On the Effect of Replication of Input Files on the Efficiency and the Robustness of a Set of Computations

Thomas Lambert

▶ **To cite this version:**

Thomas Lambert. On the Effect of Replication of Input Files on the Efficiency and the Robustness of a Set of Computations. Other [cs.OH]. Université de Bordeaux, 2017. English. NNT : 2017BORD0656 . tel-01661588

HAL Id: tel-01661588
https://theses.hal.science/tel-01661588

Submitted on 12 Dec 2017

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

THÈSE

PRÉSENTÉE À

L'UNIVERSITÉ DE BORDEAUX

ÉCOLE DOCTORALE DE MATHÉMATIQUES ET D'INFORMATIQUE

par **Thomas Lambert**

POUR OBTENIR LE GRADE DE

DOCTEUR

SPÉCIALITÉ : INFORMATIQUE

Étude de l'effet de la réplication de fichiers d'entrée sur l'efficacité et la robustesse d'un ensemble de calculs

---

Date de soutenance : 8 Septembre 2017

Devant la commission d'examen composée de :

Olivier Beaumont	Directeur de recherche, Inria Bordeaux Sud-Ouest	Directeur
Anne Benoit	Maîtresse de conférences, ENS de Lyon	Rapporteuse
Nicolas Bonichon	Maître de conférences, Université de Bordeaux	Invité
François Clautiaux	Professeur des universités, Université de Bordeaux	Examinateur
Lionel Eyraud-Dubois	Chargé de recherche, Inria Bordeaux Sud-Ouest	Encadrant
Nicolas Hanusse	Directeur de recherche, CNRS	Président
Shadi Ibrahim	Chargé de recherche, Inria Bretagne Atlantique	Examinateur
Veronika Rehn-Sonigo	Maîtresse de conférences, Université de Franche-Comté	Examinatrice
Rizos Sakellariou	Senior lecturer, University of Manchester	Rapporteur

– 2017 –

Résumé Avec l'émergence du calcul haute-performance (HPC) et des applications Big Data, de nouvelles problématiques cruciales sont apparues. Parmi elles on trouve le problème du transfert de données, c'est-à-dire des communications entre machines, qui peut génerer des délais lors de gros calculs en plus d'avoir un impact sur la consommation énergétique. La réplication, que ce soit de tâches ou de fichiers, est un facteur qui accroît ces communications, tout en étant un outil quasi-indispensable pour améliorer le parallélisme du calcul et la résistance aux pannes.

Dans cette thèse nous nous intéressons à la réplication de fichiers et à son impact sur les communications au travers de deux problèmes. Dans le premier, la multiplication de matrices en parallèle, le but est de limiter autant que possible ces réplications pour diminuer la quantité de données déplacées. Dans le second, l'ordonnancement de la phase « Map » de MapReduce, il existe une réplication initiale qu'il faut utiliser au mieux afin d'obtenir l'ordonnancement le plus rapide ou entraînant le moins de création de nouvelles copies.

En plus de la réplication, nous nous intéressons aussi à la comparaison entre stratégies d'ordonnancement statiques (allocations faites en amont du calcul) et dynamiques (allocations faites pendant le calcul) sur ces deux problèmes avec pour objectif de créer des stratégies hybrides mélangeant les deux aspects.

Pour le premier problème, le produit de matrices en parallèle, nous nous ramenons à un problème de partition de carré où l'équilibrage de charge est donné en entrée. Cet équilibrage donné, le but est de minimiser la somme des semi-paramètres des rectangles couvrant des zones ainsi créés. Ce problème a déjà été étudié par le passé et nous démontrons de nouveaux résultats. Nous proposons ainsi deux nouveaux algorithmes d'approximation, l'un fondé sur une stratégie récursive et l'autre sur l'usage d'une courbe fractale. Nous présentons également une modélisation alternative, fondée sur un problème similaire de partition de cube, dont nous prouvons la NP-complétude tout en fournissant deux algorithmes d'approximation. Pour finir, nous réalisons également une implémentation pratique du produit de matrices en utilisant nos stratégies d'allocation grâce à la librairie StarPU. Les résultats expérimentaux montrent une amélioration du temps de calcul ainsi qu'une diminution significative des transferts de données lorsqu'on utilise une stratégie statique d'allocation couplée à une technique de vol de tâches.

Pour le second problème, l'ordonnancement de la phase « Map » de MapReduce, plusieurs copies des fichiers d'entrée sont distribuées parmi les processeurs disponibles. Le but ici est de faire en sorte que chaque tâche soit attribuée à un processeur possédant son fichier d'entrée tout en ayant le meilleur temps de calcul total. Une autre option étudiée est d'autoriser les tâches non-locales (attribués à des processeurs ne possédant pas leurs fichiers d'entrée) mais d'en limiter le nombre. Dans cette thèse nous montrons premièrement qu'un algorithme glouton pour ce problème peut être modélisé par un processus de « balls-in-bins » avec choix, impliquant une surcharge (nombre de

tâches supplémentaires par rapport à la moyenne) en $O(m \log m)$ où m est le nombre de processeurs. Secondement, dans le cas où les tâches non-locales sont interdites, nous relions le problème à celui de l'orientation de graphes, ce qui permet d'obtenir des algorithmes optimaux et polynomiaux et l'existence d'une assignation presque parfaite avec forte probabilité. Dans le cas où les tâches non locales sont autorisées, nous proposons également des algorithmes polynomiaux et optimaux. Finalement, nous proposons un ensemble de simulations pour montrer l'efficacité de nos méthodes dans le cas de tâches faiblement hétérogènes.

Title On the Effect of Replication of Input Files on the Efficiency and the Robustness of a Set of Computations

Abstract The increasing importance of High Performance Computing (HPC) and Big Data applications creates new issues in parallel computing. One of them is communication, the data transferred from a processor to another. Such data movements have an impact on computational time, inducing delays and increase of energy consumption. If replication, of either tasks or files, generates communication, it is also an important tool to improve resiliency and parallelism.

In this thesis, we focus on the impact of the replication of input files on the overall amount of communication. For this purpose, we concentrate on two practical problems. The first one is parallel matrix multiplication. In this problem, the goal is to induce as few replications as possible in order to decrease the amount of communication. The second problem is the scheduling of the "Map" phase in the MapReduce framework. In this case, replication is an input of the problem and this time the goal is to use it in the best possible way.

In addition to the replication issue, this thesis also considers the comparison between static and dynamic approaches for scheduling. For consistency, static approaches compute schedules before starting the computation while dynamic approaches compute the schedules during the computation itself. In this thesis we design hybrid strategies in order to take advantage of the pros of both. First, we relate communication-avoiding matrix multiplication with a square partitioning problem, where load-balancing is given as an input. In this problem, the goal is to split a square into zones (whose areas depend on the relative speed of resources) while minimizing the sum of their half-perimeters. We improve the existing results in the literature for this problem with two additional approximation algorithms. In addition we also propose an alternative model using a cube partitioning problem. We prove the NP-completeness of the associated decision problem and we design two approximations algorithms. Finally, we implement the algorithms for both problems in order to provide a

comparison of the schedules for matrix multiplication. For this purpose, we rely on the StarPU library.

Second, in the Map phase of MapReduce scheduling case, the input files are replicated and distributed among the processors. For this problem we propose two metrics. In the first one, we forbid non-local tasks (a task that is processed on a processor that does not own its input files) and under this constraint, we aim at minimizing the makespan. In the second problem, we allow non-local tasks and we aim at minimizing them while minimizing makespan. For the theoretical study, we focus on tasks with homogeneous computation times. First, we relate a greedy algorithm on the makespan metric with a "ball-into-bins" process, proving that this algorithm produces solutions with expected overhead (the difference between the number of tasks on the most loaded processor and the number of tasks in a perfect distribution) equal to $O(m \log m)$ where m denotes the number of processors. Second, we relate this scheduling problem (with forbidden non-local tasks) to a problem of graph orientation and therefore prove, with the results from the literature, that there exists, with high probability, a near-perfect assignment (whose overhead is at most 1). In addition, there are polynomial-time optimal algorithms. For the communication metric case, we provide new algorithms based on a graph model close to matching problems in bipartite graphs. We prove that these algorithms are optimal for both communication and makespan metrics. Finally, we provide simulations based on traces from a MapReduce cluster to test our strategies with realistic settings and prove that the algorithms we propose perform very well in the case of low or medium variance of the computation times of the different tasks of a job.

Keywords Algorithmic, Scheduling, Parallel Computing, Matrix Product, MapReduce

Mots-clés Algorithmique, Ordonnancement, Calcul Parallèle, Produit de matrices, MapReduce

Laboratoire d'accueil Laboratoire Bordelais de Recherche en Informatique (LaBRI)

Remerciements

Mes premiers remerciements vont bien sûr à mes directeurs, Olivier et Lionel, qui pendant les trois années qu'ont duré cette thèse, se sont toujours montrés disponibles et attentifs, notamment dans ces moments intenses que sont la rédaction et la préparation de la soutenance. Ce fut un réel plaisir de les côtoyer et de travailler avec eux et j'espère bien pouvoir continuer à le faire dans le futur.

J'aimerais ensuite remercier mes rapporteurs, Anne Benoit et Rizos Sakellariou, pour avoir accepté de relire ma thèse et ce malgré un timing pas forcément agréable. Merci aussi à Nicolas Hanusse, Shadi Ibrahim, Veronika Rehn-Sonigo, François Clautiaux et Nicolas Bonichon de m'avoir fait l'honneur de compléter mon jury.

Parce que sans eux cette thèse n'aurait peut-être pas eu lieu, j'ai aussi une pensée pour l'équipe ROMA du LIP (ENS de Lyon), en particulier Frédéric, Loris et Bora, qui m'ont fait découvrir les joies de l'ordonnancement et du calcul parallèle au travers des cours qu'ils ont dispensé et lors de mes stages de M1 et M2 qu'ils ont encadré.

Il me faut aussi remercier les très nombreuses personnes que j'ai croisé au cours de ces trois ans. Tout d'abord mes compagnons de pause midi, anciens et passés, Bruno, poète virtuose, Pierre, dont nos conversations sur le cinéma et le sport me manquent, Zoé, skieuse de l'extrême, Alice, parisienne en exil, Elsa, dont la compassion fut une aide précieuse pendant les jours difficiles, Sami, dont nos conversations sur le cinéma et le sport, toujours courtoises malgré des désaccords de fond, vont me manquer, Jonathan, quémandeur de pokemons , et Philippe, authentique jusqu'au bout. Il me faut aussi mentionner les autres doctorants de l'IMB qui m'ont accueilli et qui sont partis depuis vers d'autres cieux, Samuel, Alan, Jean-Baptiste, Marie, Camille, Romain, Marc, Jocelyn ainsi que ceux qui sont encore là, Nicolas, Nikola, Vassilis, Thibault, Manon, Roberto, Gabriele et Sergio. Il ne me faudrait pas oublier non plus les résidents du LaBRI que j'ai eu l'occasion de côtoyer en séminaire ou lors des activés de l'AFoDIB, Mathieu, Henry, Théo, Simon, Patxi, Noël, Rémi, Claire P., Joris, Nicolas, Claire C., David J., David R., David I., Romaric, Marie, Louis-Marie, Nathan, Mohamed et Thomas. Enfin je pense aussi aux résidents de l'Inria, notamment Abdou, Samuel, Terry et Yann qui m'ont aidé à percer les arcanes de StarPU, mais aussi Emmanuel J., Emmanuel A., Matthieu,

Marc, Guillaume et Suraj.

Je n'oublie pas non plus mes vieux compagnons de l'ENS que j'ai toujours plaisir à revoir. Tout d'abord Thomas, Robin et Vincent C., mes "co-loques" du 1 avenue Debourg, fameux squat et haut-lieu d'expérimentation culinaire, musical, cinématographique, vidéo-ludique et biorythmique auquel je pense souvent la larme à l'œil, du moins jusqu'à ce que je me souvienne du saxophone de Robin, du sens de l'humour bien à lui de Thomas et du goût un brin trop prononcé de Vincent pour le piment. Je continue avec Élie, inépuisable répertoire de savoirs inutiles donc indispensables, Rémi D.J.D.V., empiriste imprévisible, Étienne, râleur d'élite, Mathias, éminent spécialiste en comédies françaises, Fred, « docteur licorne », Samantha, à qui je dois les joies de la rubrique fait-divers d'un petit journal du sud des Pouilles, Anaël, œnologue hétérodoxe, et Benjamin, à qui j'aurais dû proposer de m'écrire ces remerciements contre rémunération, ils auraient eu beaucoup plus de gueule. J'ai enfin une petite pensée pour tous les autres que j'ai pu croisé au Foyer ou ailleurs, Margaux, Damien, François, Vincent L.,Adrien, Isa, Julien, Noé, Léo, Bertrand, Agathe, Arthur, Rémi N., Aurélien, Alexis, Alvaro, Marine, Alexandre, Florine, Ronan, Sylvie...

Merci aussi à ma famille, notamment mes parents Jean-Luc et Françoise, qui, malgré une volonté évidente de ma part de m'éloigner d'eux à chaque nouvelle étape de mes études, m'ont toujours soutenu et encouragé.

J'ai aussi une petite pensée pour les deux urgentistes, les trois généralistes, les trois kinésithérapeutes, les deux radiologues et la podologue qui se sont relayés autour de mes deux chevilles et de mon dos ces trois dernières années dans l'espoir de les remettre en état (avec un succès mitigé mais des efforts méritoires).

Enfin, et ça sera ma conclusion, désolé à ceux que j'ai oublié et merci. Certains appelleront ça de la lâcheté, je préfère parler de prévoyance diplomatique.

Résumé détaillé

Avec l'émergence du calcul haute-performance (HPC) et des applications Big Data, de nouvelles problématiques cruciales sont apparues. Parmi elles on trouve le problème du transfert de données, c'est-à-dire des communications entre machines, qui peut engendrer des délais lors de gros calculs en plus d'avoir un impact sur la consommation énergétique. La réplication, que ce soit de tâches ou de fichiers, est un facteur qui accroît ces communications, tout en étant un outil quasi-indispensable pour améliorer le parallélisme du calcul et la résistance aux pannes.

Dans cette thèse nous nous intéressons à la réplication de fichiers et à son impact sur les communications au travers de deux problèmes. Dans le premier, la multiplication de matrices en parallèle, le but est de limiter autant que possible ces réplications pour diminuer la quantité de données déplacées. Dans le second, l'ordonnancement de la phase « Map » de MapReduce, il existe une réplication initiale qu'il faut utiliser au mieux afin d'obtenir l'ordonnancement le plus rapide ou entraînant le moins de création de nouvelles copies.

En plus de la réplication, nous nous intéressons aussi à la comparaison entre stratégies d'ordonnancement statiques (allocations faites en amont du calcul) et dynamiques (allocations faites pendant le calcul) sur ces deux problèmes avec pour objectif de créer des stratégies hybrides mélangeant les deux aspects.

Les trois premiers chapitres de cette thèse sont consacrés au produit de matrices parallèle. Dans le premier nous nous concentrons sur une première modélisation se basant sur un problème de partitionnement de carré. Le produit de matrices $C = A \times B$ peut en effet être considéré comme un ensemble de tâches unitaires de la forme

$$T_{i,j,k} : C_{i,j} \leftarrow C_{i,j} + A_{i,k}B_{k,j}$$

qu'il faut donc distribuer entre les processeurs disponibles. Dans un premier temps nous considérons que toutes les tâches ayant le même $C_{i,j}$ doivent être données au même processeur (afin, par exemple, d'éviter des conflits d'écriture) ; nous nous retrouvons avec la matrice résultat C à partager. Si l'on suppose la durée des tâches homogènes (une opération de type GEMM est plutôt stable en pratique), l'équilibrage se fait facilement en accordant à chaque processeur

un nombre de tâche proportionnel à sa vitesse de calcul relative. On a donc un carré (la matrice résultat que l'on supposera carrée) à partager entre plusieurs processeurs dont on connaît la surface occupée par chacun. Afin d'évaluer le nombre de communications induites par un partitionnement (et que l'on cherchera à minimiser), on considérera le nombre de $A_{i,k}$ et de $B_{k,j}$ à charger pour chaque processeur, nombre qui est proportionnel à la somme des projections sur les deux dimensions des zones attribués aux processeurs. Cette somme des projections sera donc notre fonction objectif à minimiser. Nous appelons ce problème PERI-SUM.

Le problème de décision associé à PERI-SUM ayant été prouvé NP-complet, nous nous sommes concentrés sur des algorithmes d'approximations en essayant d'améliorer la littérature préexistante. Nous avons en particulier proposé deux nouveaux algorithmes : NRRP et SFCP. Le premier est un algorithme récursif. A chaque étape le rectangle (le carré en entrée lors du premier appel) est partagé en plusieurs morceaux (au moins deux), de même que l'ensemble des surfaces (les vitesses relatives). L'algorithme est ensuite de nouveaux appelés sur chaque couple (rectangle, surfaces). Notre principal résultat de ce chapitre est la preuve que NRRP est une $\frac{2}{\sqrt{3}}$-approximation ($\frac{2}{\sqrt{3}} \simeq 1.15$), ce qui constitue le meilleur ratio d'approximation connue à ce jour. SFCP quant à lui est un algorithme qui utilise une courbe fractale, la courbe de Hilbert, pour partitionner le carré en préservant une certaine localité. Nous avons montré que SFCP est une $\frac{3\sqrt{3}}{\sqrt{11}}$-approximation ($\frac{3\sqrt{3}}{\sqrt{11}} \simeq 1.57$) ce qui constitue à notre connaissance la première analyse de cette technique classique pour ce problème. Enfin, dans ce chapitre, nous avons aussi proposé une comparaison de stratégies statiques (comme les algorithmes précédents) et dynamiques à l'aide de simulations afin notamment d'évaluer la robustesse de notre approche en cas d'évaluation moins fiable des vitesses. Ces simulations prouvent notamment que des stratégies hybrides (statiques avec une composante dynamique) permettent d'améliorer la stabilité des stratégies statiques tout en diminuant sensiblement le nombre de communications par rapport aux stratégies dynamiques, montrant ainsi l'intérêt pour ce problème des ordonnancements statiques.

Dans le second chapitre de cette thèse nous proposons une nouvelle modélisation de la multiplication parallèle de matrices. Pour cela nous nous affranchissons de la contrainte de devoir attribuer toutes les tâches mettant en jeu un même $C_{i,j}$ à un même processeur et partitionnons cette fois l'espace des tâches (qui est donc tri-dimensionnel). On a donc cette fois un problème, nommé MSCubeP, où l'on partitionne un cube en zones de volumes donnés et en minimisant la somme des projections sur les trois dimensions. Dans un premier temps nous avons montré la NP-complétude du problème de décision associé à une variante du problème, MSCuboidP (où l'on partitionne un parallélépipède rectangle au lieu d'un cube). La démonstration de ce résultat

se base sur une réduction depuis le problème de partition, une preuve similaire à celle pour la NP-complétude du problème 2D. Nous proposons ensuite deux algorithmes : 3D-NRRP et 3D-SFCP très similaires aux algorithmes du problème 2D. 3D-NRRP est ainsi basé sur le même principe de divisions successives du cube initial et nous montrons que c'est une $\frac{5}{6^{2/3}}$-approximation ($\frac{5}{6^{2/3}} \simeq 1.51$). De même 3D-SFCP utilise une courbe de Hilbert en trois dimensions pour effectuer son partitionnement et nous montrons que c'est une $\frac{7^{5/3}}{6 2^{2/3}}$-approximation ($\frac{7^{5/3}}{6 2^{2/3}} \simeq 1.64$). Enfin, pour clore ce chapitre nous proposons aussi une comparaison des deux modélisations, 2D et 3D. Il est en effet facile de transformer toute solution de PERI-SUM en solution de MSCubeP en répliquant la solution tout au long de la troisième dimension. Dans cette section nous montrons notamment, théoriquement pour certains cas, via des simulations pour les autres cas, que les solutions issues de MSCubeP sont très vite meilleures et que le ratio (solution 2D)/(solution 3D) augmente quand le nombre de processeurs augmente, avec un comportement asymptotique en $O(m^{1/6})$, où m est le nombre de processeurs.

Dans le troisième chapitre nous proposons une implémentation pratique du produit de matrices. Le but en est bien sûr de tester en conditions réelles les bons résultats théoriques prouvés auparavant ainsi que de confirmer les résultats des simulations. Cette implémentation permet aussi d'interroger le modèle, notamment certaines hypothèses réalisées, la confiance dans les estimations de vitesses, la superposition des communications avec les calculs ou l'indépendance des tâches entre elles dans le cas de la modélisation 3D. Pour la réalisation pratique nous utilisons la librairie StarPU, développée à l'Inria Bordeaux Sud-Ouest, qui permet la création de codes portables et adaptés aux architectures hétérogènes (incluant par exemple CPUs et GPUs). Dans StarPU les tâches sont soumises dès qu'elles sont disponibles à un ordonnanceur qui les distribue ensuite aux différents processeurs selon un algorithme qui peut être défini et implémenté par l'utilisateur. Nous avons donc produit un ordonnanceur hybride qui utilise NRRP ou 3D-NRRP pour distribuer statiquement les tâches dans un premier temps, puis utilise un système de vol de tâches pour corriger dynamiquement cet ordonnancement, particulièrement en cas de processeur inactif. Le vol de tâche essaye, en regardant plus ou moins de tâches, de chercher celle qui implique le moins de communications (donc qui nécessite le moins de nouvelles données du point de vue du processeur voleur). Nous comparons ces stratégies hybrides avec leurs pendants purement statiques ou dynamiques, ainsi qu'avec DMDA, une stratégie pré-implémentée dans StarPU et qui vise la minimisation du temps de calcul. Au final les résultats révèlent principalement deux choses : 1) pour éviter les conflits d'écriture lorsque le modèle utilisé est basé sur MSCubeP nous avons été obligés d'ajouter des dépendances (lorsque deux tâches partagent un $C_{i,j}$ on attend que la première soit finie pour soumettre la deuxième) ou des tâches de réduction (on crée des

versions temporaires des éléments de la matrice résultat que l'on somme à la fin), ce qui dans les deux cas dégrade significativement le temps de calcul sur la plateforme considérée, 2) néanmoins les stratégies basées sur NRRP offrent des performances plus que satisfaisantes avec à la fois une énorme diminution des communications et même une légère amélioration du temps de calcul.

Le dernier chapitre de cette thèse est lui consacré au second problème, l'ordonnancement de la phase "Map" de MapReduce. Cette dernière peut se modéliser par une distribution de tâches indépendantes avec pour contraintes un pré-placement des fichiers d'entrée à respecter. Afin d'améliorer, entre autre, le parallélisme de l'application chaque fichier d'entrée est répliqué plusieurs fois et donc chaque tâche a plusieurs processeurs candidats à son exécution. En pratique, afin d'avoir le meilleur temps de calcul possible, on autorise néanmoins les tâches non locales, c'est à dire les tâches exécutées sur un processeur ne possédant pas, lors de la distribution initiale, des fichiers d'entrée correspondants, mais cela entraîne des communications que l'on veut éviter. Nous nous retrouvons donc avec deux problèmes : un premier où l'on interdit ces communications supplémentaires et l'on cherche le meilleur temps de calcul possible (MAKESPAN-MAPREDUCE) et un second ou l'on équilibre à tout prix les tâches afin d'obtenir le meilleur temps de calcul mais en limitant autant que possible le nombre de tâches non locales à réaliser pour cela (COMMUNICATION-MAPREDUCE). Notons que dans les deux cas les tâches sont supposées homogènes (une distribution parfaite serait donc au plus la partie entière du nombre de tâches divisée par le nombre de processeurs allouée à chaque processeur) et nous représentons le problème par un graphe biparti représentant le placement des fichiers d'entrée. Dans un premier temps nous nous concentrons sur MAKESPAN-MAPREDUCE. Nous montrons notamment ses liens avec le problème de "balls-into-bins" avec choix multiple, ce qui nous permet de prouver qu'un algorithme glouton sur ce problème retourne une solution dont l'espérance du temps de calcul est $\frac{n}{m}+O(\log\log m)$ où n est le nombre de tâches et m le nombre de processeurs. Ensuite, nous montrons que ce problème est équivalent à un problème d'orientation de graphe. Il en résulte l'existence d'algorithmes polynomiaux et optimaux ainsi que l'existence, avec forte probabilité, d'une solution avec un temps de calcul $\frac{n}{m}+1$. Dans un second temps, nous nous concentrons sur COMMUNICATION-MAPREDUCE. En particulier nous proposons deux algorithmes polynomiaux et optimaux, FindAssignment et BestAssignment, basés sur des chemins alternants empruntés à la littératures des flux et des couplages. Nous montrons, par ailleurs, que ces deux algorithmes sont aussi optimaux pour MAKESPAN-MAPREDUCE, impliquant une solution commune aux deux problèmes.

Après ces résultats théoriques nous proposons une série de simulations pour comparer BestAssignment avec des deux algorithmes pré-existants. Nous proposons deux séries de simulations. Dans la première nous gardons le modèle

avec les tâches homogènes et évaluons le gain que l'on obtient avec BestAssignment (qui est optimal) par rapport aux autres algorithmes. Dans la seconde série nous utilisons des traces issues d'un serveur Hadoop pour évaluer les performances de BestAssignment en situation pratique, notamment avec des tâches qui peuvent se révéler de durées très différentes. Globalement les simulations montrent une importante diminution du nombre de tâches non-locales pour des variances faibles ou moyenne de la durée des tâches avec BestAssignment. Cependant, dans le cas d'une variance élevée et des rapports n/m trop importants les anciennes stratégies purement dynamiques semblent mieux fonctionner.

Contents

Introduction

Parallel computing is a very important field of computer science where computation resources are aggregated in order to obtain a greater computational power. The increase in the demand for high-speed computation and analysis of larger and larger datasets has reinforced parallel computing as a crucial research subject in the recent years.

Two main approaches have emerged. For a long time, parallel computing was dominated by HPC (High Performance Computing), *i.e.* big local aggregation of processors like supercomputers. HPC mainly focuses on computations, in particular scientific ones, with important hardware improvement on processors and very optimized and specific codes. However the increase of the size of data used in HPC becomes an increasingly important problem and is identified as one of the major challenges in the future of HPC, see Shalf *et al.* [2011] and Geist and Lucas [2009].

More recently, datacenters and Big Data applications have developed a more distributed approach, different from the one used in HPC. The main strength of Big Data approach is to place storage as close as possible to local computational resources, allowing fast analysis of large datasets (that can easily be split in different chunks and then distributed). This approach mainly focuses on data distribution, with dedicated file systems like HDFS (Borthakur [2008]), and uses simple programming models like MapReduce (Dean and Ghemawat [2008]).

Initially, since the two approaches went in different directions, their usages and associated challenges were different. One consequence was the existence of two ecosystems with little in common, see Figure 1. Nowadays, for economic and technical reasons, we can observe a "convergence" of both approaches. For example, at programming level, HPC becomes more accessible with tools like StarPU (Augonnet *et al.* [2011]), PaRSEC (Bosilca *et al.* [2013]) or KAAPI (Gautier *et al.* [2007]) that avoid complicated MPI programming for users. At the same time, the programming models in Cloud Computing are getting more complex to improve efficiency of specific applications. Spark (Spark [2016]), for example, is an improved version of MapReduce which provides a much more efficient management of intermediate data. Similarly, at the hardware level, the apparition of burst buffers on HPC machines make the data closer to the computational resources than before while Infiniband resources, that used to

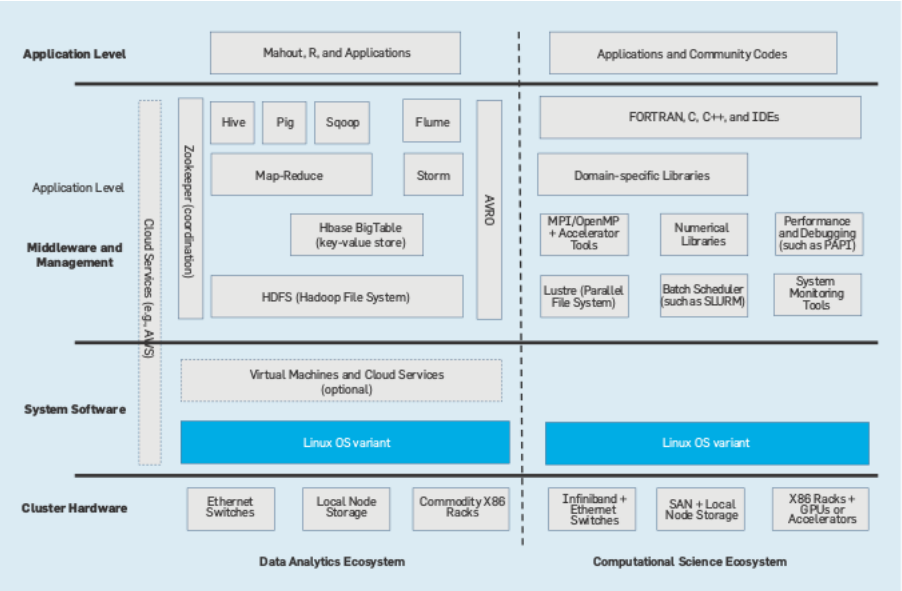


Figure 1: Ecosystems of Big Data (left) and HPC (right) (From Reed and Dongarra [2015]).

be specific to HPC architectures, are now a common component in Big Data clusters.

This convergence between HPC and Big Data implies many crucial issues (Reed and Dongarra [2015]). A major aspect is data management and many sub-questions result from this problematic: how to minimize data movement? How to improve data availability? How to make the best use of data known to be read-only?....

Among all these questions, this thesis considers specifically the replication problematic. Today, replication is a very classical tool in parallel computing. Two kinds of replication schemes can be distinguished: task replication and data replication.

In the case of tasks, replication provides an interesting tool to bring resilience to computation with a large set of tasks. More precisely, the replication of a task, *i.e.* the multiple execution of the same task, improves the chances of this task to finish and to be correct. Such an operation can be done in a reactive way, if there is a suspicion of a failed or a very slow task. This is, for example, the way that MapReduce (Dean and Ghemawat [2008]) deals with stragglers, *i.e.* abnormally slow tasks, see Zaharia *et al.* [2008], Fu *et al.* [2015] or Guo and Fox [2012] for studies on the problem of managing such a mechanism. Task replication can also be done in prevision. In this case, the scheduler assumes that there will be a failure during the execution and therefore launches several duplicates of the same task to increase the chances of

finishing this task, see Wang *et al.* [1995] or Litke *et al.* [2007], or to check the correctness of a task to avoid silent errors, see Lyons and Vanderkulk [1962] or Fiala *et al.* [2012]. Task replication is beyond the scope of this thesis. However we briefly consider task replication in Chapter 1 and Chapter 4.

The other type of replication, data replication, is a crucial component in distributed computing. It can be used to avoid data loss (if the storage node becomes down) and, more importantly, to improve the accessibility of data. Indeed, data is notably used as input files. To be processed on a given processor, a task needs its input data to be gathered on this processor. Therefore increasing the replication of input files allows more processors to be able to schedule a task without data movement during the computation. This technique has widely proved its efficiency, see Chervenak *et al.* [2000], Stockinger *et al.* [2001] or Ranganathan and Foster [2001]. This led some storage systems, like for example HDFS (Borthakur [2008]), to replicate every input data chunk three times by default. In this thesis we also consider replication performed during the computation of a set of tasks, *i.e.* data movement (or communication). A very classical behaviour in parallel computing is to never let a processor idle. Therefore, when a processor has no local input files of a ready task, it downloads these files from another processor, thus creating new replicates.

However replication has a cost. The first cost, of course, is storage. If a data is replicated d times, this data alone will be using d times more disk capacity. In the case of Google or Amazon clusters, for example, the user pays in function of used capacity and thus heavy replication significantly increases the storage price. In addition, replication implies data movement and communication between nodes. Communications are an important issue in HPC, for the following reasons. First of all, data movement is not instantaneous. Hoemmen [2010] provides, in its introduction, an excellent review of this issue. Communication time mostly depends on two parameters, latency and bandwidth. Technical progress improves these values regularly, but these improvements are nevertheless slow in comparison to the increase of computational power. For example Graham *et al.* [2005] point that, from 1988 to 2004, the flops of commodity processors increased by 59% while DRAM latency only decreased by 5.5% (and the trend goes on). Therefore the overlap of communication with task computation, a classical solution to avoid idle time while loading data, could reach its limits if there are too many (or too large) objects to move. In addition, communications also have an impact on energy consumption, as underlined by Shalf *et al.* [2011] who place, among others, communication reduction as one of the challenges to solve for the future of HPC.

In this thesis we study two particular problems under this communication-avoiding angle. In Chapter 1, Chapter 2 and Chapter 3 we consider a very classical operation, namely matrix multiplication. In these three chapters we aim at minimizing the data movement (assuming there is no data replication at the beginning) while ensuring good makespan. In this case, replication is

treated as an objective function to minimize. In Chapter 4 we consider a second problem, namely scheduling of the Map phase in MapReduce. In this case we suppose that input files are replicated at the beginning and we want to make use of these replications in order to have the best possible makespan, or to avoid the creation of new duplicates. In this case, data replication is an input of the optimisation problem on which we have no influence and that we want to use in the best possible way.

Another aspect of this thesis is the opposition between static and dynamic strategies, the two main scheduling approaches.

A scheduling strategy is said to be static if the schedule is determined before the computation. To do so, static strategies rely on model and evaluation of different parameters (computation time, size of the inputs, ...). On the other hand, a scheduling strategy is said to be dynamic if its schedule is determined during the computation. To do so, dynamic strategies use their knowledge of the actual state of the different components (tasks, processors, input files, ...). Both have been widely used and studied, static (Baptiste *et al.* [2012], Sen and Gupta [1984] or Semar Shahul and Sinnen [2010]) and dynamic (Ouelhadj and Petrovic [2009] or Blumofe and Leiserson [1999]).

In the dynamic case, we consider two kinds of strategies: resource-centric and task-centric approaches. In the first case, resource-centric scheduling, decisions are made when resources are ready, in general when a processor becomes idle. Such strategies are designed to optimize the use of the available resources. An example of resource-centric scheduling strategy is the work-stealing technique where idle processors are allowed to steal tasks from other processors if they are able to finish them earlier (Blumofe and Leiserson [1999] or Bleuse *et al.* [2015]). On the other hand, task-centric schedulers make their decisions when tasks are ready (for example when they are submitted or when all tasks they depend on are finished). In this case the goal is to optimize the cost of each task, in general to complete it as early as possible. A strategy like MCT (Minimum Completion Time) that is close to the HEFT strategy (Heterogeneous Earlier Finishing Time, Topcuoglu *et al.* [2002]) and which allocates each task on the processor that can finish it the earliest, is a good example of such an approach.

The pros and cons of dynamic and static approaches are well-known. Static scheduling relies on a larger vision of the problem, allowing scheduling decisions that use knowledge from future tasks and DAG structure to provide schedules that can be proved to be more efficient that the ones from dynamic strategies (Albers [2003]). This holds true even if associated decision problems are often NP-complete, requiring the search of approximation algorithms. However, the good quality of static strategies relies on the knowledge they have on the input. If this knowledge is unreliable, the scheduling can be based on false and not up-to-date information (due to wrong estimations of processing and transfer

times for instance), leading to bad decisions. Meanwhile, dynamic strategies base their decisions on more accurate information, what reduces the chance to making really bad choices, and are designed to react to unexpected events. Therefore, dynamic strategies are often seen as a more reliable technique in HPC because they are less dependent on accurate evaluation and able to adapt from unexpected behaviours (task failures, broken links between machines, ...).

In this thesis, we mainly focus on static scheduling, in particular in Chapter 1, Chapter 2 and Chapter 4, providing theoretical analysis. For both problems, parallel matrix multiplication and Map phase of MapReduce scheduling, we try to make use of the strengths of static approach while minimizing its weaknesses by augmenting it with dynamic techniques (mostly work-stealing) in order to create hybrid approaches.

The thesis outline goes as follows.

In Chapter 1, we introduce the issue of communication-avoiding parallel matrix multiplication with a focus on heterogeneous platforms. After introducing the PERI-SUM problem, that is the theoretical square partitioning problem to work on, and presenting the different previous studies, we propose comparison between static and dynamic approaches for this problem. We prove that, with the addition of a work-stealing strategy, static approaches can be reliable for this particular case. In addition, we also design NRRP and SFCP, two approximation algorithms for PERI-SUM with proved approximation ratios. This work has been partially realized with Abdou Guermouche from HiePACS team at Inria Bordeaux Sud-Ouest and has been partially published in SBAC-PAD'15 (Beaumont *et al.* [2015]) and IPDPS'16 (Beaumont *et al.* [2016b]).

In Chapter 2, we pursue the study of parallel matrix multiplication with a new model, the MSCubeP problem, which replaces the square partitioning problem by a cube partitioning problem. Shortly, with PERI-SUM the goal is to partition the output matrix while with MSCubeP we partition the entire set of elementary tasks. In this chapter, we provide an NP-completeness proof of MSCuboidP, a more general variant of MSCubeP, and two approximation algorithms, 3D-NRRP and 3D-SFCP. In addition, we propose a comparison between the PERI-SUM and the MSCubeP models, showing that the algorithms for MSCubeP provide significantly better solutions than the ones from PERI-SUM (in terms of induced communications), except for small instances where the second performs slightly better. This work has been partially published in Euro-Par'16 (Beaumont *et al.* [2016c]).

In Chapter 3, we propose a practical implementation of the algorithms from the two previous chapters. For this purpose, we use the StarPU library (Augonnet *et al.* [2011]) and propose experimentations on the computational platform PlaFRIM2. From these experiments, we provide trace analysis and

show a small speed-up and a significant decrease of data movement when a static scheduler is used in place of the pre-implemented dynamic scheduler. This recent work is still to be published.

In Chapter 4, we introduce the problem of scheduling during the Map phase of MapReduce, a very popular paradigm of distributed computing. We propose in this chapter two close models for this problem, MAKESPAN-MAPREDUCE and COMMUNICATION-MAPREDUCE, which differ on the objective function (communication versus makespan), and a survey of scheduling techniques on this issue. We also relate MAKESPAN-MAPREDUCE with the probabilistic model Balls-In-Bins and to GRAPH-ORIENTIABILITY, a problem of edge orientation in graph, allowing us to use the existing literature for these two problems. Finally, we propose optimal polynomial-time algorithms (FindAssignment and BestAssignment) for COMMUNICATION-MAPREDUCE and MAKESPAN-MAPREDUCE in the case of homogeneous durations and propose a simulation-based analysis of their behaviour with online and heterogeneous settings. In this case, the results are very encouraging: we observe a decrease in the number of non-local tasks (tasks that are processed on nodes that do not own their input files) when the variance between the computation times is small enough. This work has been realized with Loris Marchal from Roma team at ENS de Lyon and Bastien Thomas, intern in the REALOPT team at Inria Bordeaux Sud-Ouest and is planned to be submitted for publication shortly.

Chapter 1

Square Partitioning for Communication-Avoiding Parallel Matrix Multiplication

1.1 Introduction

In the case of homogeneous resources, the problem of partitioning data for Linear Algebra kernels in order to both balance the makespan throughout the computation and to minimize communications is well understood. 2D block-cyclic distributions, for instance, have been introduced in Scalapack Choi *et al.* [1996] in order to achieve this goal. More recently, the problem has received a lot of attention in Communication Avoiding algorithms design (see Hoemmen [2010]; Anderson *et al.* [2011] and Solomonik and Demmel [2011]; Ballard *et al.* [2012] for Matrix Multiplication specifically). In this context, the goal is to partition the set of elementary computations to be performed into a minimal number of zones, each zone being able to be processed in local memory (*i.e.* both input, intermediate and output data). This corresponds to maximizing the volume of computations that can be processed with a given amount of memory.

In this chapter (and in Chapter 2 and Chapter 3), we concentrate on dense Matrix Multiplication algorithms and more specifically on Matrix Multiplication algorithms that involve N^3 elementary operations of type $C_{i,j} \leftarrow C_{i,j} + A_{i,k}B_{k,j}$, *i.e.* we ignore variants such as Strassen (Strassen [1969], Ballard *et al.* [2012] also propose a communication avoiding algorithm for this case) or Coppersmith-Winograd (Coppersmith and Winograd [1990]). Note that throughout this chapter, and during Chapter 2 and Chapter 3, we will assume that matrices are partitioned into blocks, whose size is chosen so as to be well adapted to all types of resources (typically CPUs and GPUs). On the other hand, we consider a fully heterogeneous platform where nodes may have different processing capacities, and we address the most general problem,

where several partially aggregated copies of C can reside simultaneously in memory, such as in 2.5D algorithms Solomonik and Demmel [2011]. In this context, the problem consists in partitioning the computational domain (the cube of N^3 points) into sub-domains allocated to the different resources. In order to balance the load between the processing units, each unit should receive a volume of computation proportional to its processing speed and the overall amount of communication, that corresponds to the overall boundary area between the zones, should be minimized.

We target a heterogeneous computing platform with m computing resources labelled $M_1, M_2, \ldots, M_m$ and we denote by w_l the time necessary to process an elementary block matrix product on M_l. For the sake of simplicity and in order to stick with more traditional scheduling models and notations, we will assume that there exists a master node M_0 that sends out chunks to workers over a network. In practice, data initially resides in the memory and will be sent from there to the different processing resources. Therefore, main memory will act as the master node.

In order to have a clear and precise objective function, we will consider a model for communications where communications can be overlapped with computations, what is a reasonable assumption in the case of dense blocked matrix product. Thus, communication time will not be explicitly taken into account in the makespan, but the overall number of elements sent from the main memory to the processing units and between the processing units themselves will be our target, and can be seen either as a measure of possible congestion or as a measure of communication energy. Note that we do not forget the makespan and evaluate communication volume assuming the makespan is optimal by maintaining a perfect load balancing.

1.1.1 Formal Definition of the Problem

The standard matrix multiplication $C = A \times B$ can be seen as a set of N^3 (for square matrices) tasks

$$T_{i,j,k} : C_{i,j} \leftarrow C_{i,j} + A_{i,k}B_{k,j}$$

for $\{i, j, k\} \in [1, N]^3$. Each task $T_{i,j,k}$ needs exactly three elements, $C_{i,j}$, $A_{i,k}$ and $B_{k,j}$ to be computed and each of these elements are also needed by other tasks. For example if two tasks $T_{i,j,k}$ and $T_{i,j,k'}$ are allocated to two different processors, respectively M_1 and M_2, then replicates of $C_{i,j}$ must reside in the local memory of M_1 and M_2. Note that what we call elements, the $A_{i,k}$s, $B_{k,j}$s and $C_{i,j}$s, are blocks, *i.e.* sub-matrices of the original A, B and C. As stated before, our goal here is to avoid as much as possible these replications, under the assumption that we aim for an optimal makespan (that is achieved simply by giving to each processor a number of $T_{i,j,k}$s proportional to its relative computation speed).

As a first step we allocate tasks by bags. For a fixed k we allocate $T_{i,j,k}$ between the different processors and reproduce this scheduling for each k, limiting the problem to only two dimensions (the $A_{i,k}$'s and the $B_{k,j}$'s) to simplify it. In addition, note that each $C_{i,j}$ is allocated to a single processor, avoiding the problem of concurrent writing. This technique is the one mainly used for High-Performance Matrix-Multiplication and introduced by ScaLAPACK (Blackford *et al.* [1997], and Goto and Geijn [2008] for a more general survey). This case can be modelled by the square partitioning problem PERI-SUM. We define the following notations. Let Z be a zone included into a unary square $S = [0,1] \times [0,1]$. We define by $s(Z)$ its area (formally $\iint_Z dxdy$). Let $\Pi_1(Z) = \{x,\ \exists (x,y) \in Z\}$ and $\Pi_2(Z) = \{y,\ \exists (x,y) \in Z\}$ be the projections of Z on both dimensions and denote by $\pi_1(Z)$ and $\pi_2(Z)$ their sizes ($\pi_1(Z) = |\Pi_1(Z)|$ and $\pi_2(Z) = |\Pi_2(Z)|$). Then we define the half-perimeters of Z as $p(Z) = \pi_1(Z) + \pi_2(Z)$, see Figure 1.1.

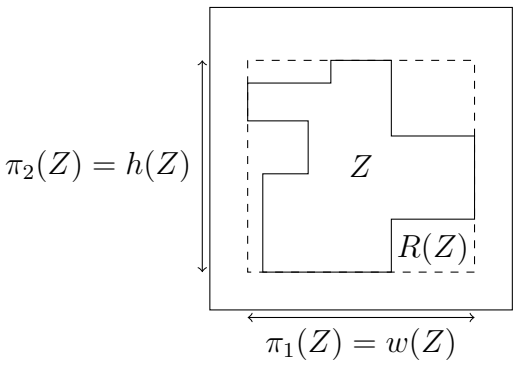
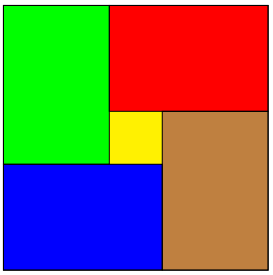


Figure 1.1: Illustration of the definition of $\pi_1(Z)$, $\pi_2(Z)$ and $R(Z)$

Problem 1.1 (PERI-SUM)**.** Given a set of m strictly positive rational numbers $\{s_1, \ldots, s_m\}$ such that $\sum s_k = 1$, and the square $S = [0,1] \times [0,1]$, find for each s_k a zone $Z_k \in S$ such that the surface of Z_k is s_k, $\bigcup Z_k = S$, and such that $\sum p(Z_k)$ is minimized.



$$\begin{aligned} & 2/5 + 3/5 \\ + \ & 3/5 + 2/5 \\ + \ & 3/5 + 2/5 \\ + \ & 2/5 + 3/5 \\ + \ & 1/5 + 1/5 \\ \sum p(Z) \ = \ & 22/5 \end{aligned}$$

Figure 1.2: Illustration of PERI-SUM.

An illustration of PERI-SUM can be found on Figure 1.2. For the particular case of continuous zones (the most common case in this chapter, except in Section 1.5) we also define the notion of covering rectangle and denote it by $R(Z)$. $R(Z)$ is the smallest rectangle such that $Z \subseteq R(Z)$. If $R(Z) = [x_1, x_2] \times [y_1, y_2]$, then let us define the width of Z by $w(Z) = x_2 - x_1$ and the height of Z by $h(Z) = y_2 - y_1$. Note that if Z is connected, then $\pi_1(Z) = w(Z)$ and $\pi_2(Z) = h(Z)$, see Figure 1.1. In this case we also define $\rho(Z) = \frac{\max(h(Z),w(Z))}{\min(h(Z),w(Z))}$ as its aspect ratio.

This problem has been firstly introduced by Kalinov and Lastovetsky [2001] and proven NP-Complete by Beaumont *et al.* [2002], see Section 2.2.2 for more detail.

In PERI-SUM the square to be partitioned represents the result matrix C (or more precisely one step of its computation) with its different elements, the $C_{i,j}$s. The set $\{s_1, \ldots, s_m\}$ is the relative speed of the processors ($\frac{w_i}{\sum w_j}$) used to perform the parallel matrix multiplication: if a processor is faster than another, it then has to compute more $C_{i,j}$s. To calculate $C_{i,j}$, $A_{i,k}$ and $B_{k,j}$ are needed. Therefore, to compute a subset Z of element of C, a processor has to load $\{A_{i,k},\ C_{i,j} \in Z\} \sim \Pi_1(Z)$ and $\{B_{k,j},\ C_{i,j} \in Z\} \sim \Pi_Z(Z)$ and thus the amount of communication for this processor is the half-perimeter as defined above.

Note that PERI-SUM can also be adapted to be used as a building block for many dense linear algebra kernels. For instance, it has been extended to LU factorization and other dense linear algebra kernels in Beaumont *et al.* [2001b,a]. In this case, block cyclic principle is combined with the initial partitioning in order to obtain 2D-cyclic ScaLAPACK solutions (Blackford *et al.* [1997]), where the load is balanced throughout the whole computation. These partitionings have also been adapted to distributed hierarchical and highly heterogeneous platforms by Clarke *et al.* [2012], where the partitioning is applied at two levels (intra-node and inter-node), based on sophisticated performance models. The same partitioning has also been extended to finite-difference time-domain (FDTD) method to obtain numerical solutions of Maxwell's equations by Shams and Sadeghi [2011].

In this model two assumptions are made :

- The communication cost for each processor is the same.
- Each processor has an unlimited local memory.

Ballard *et al.* [2011a] propose, for example, a different modelling where the memory is limited and the bandwidth and writing-on-cache times are heterogeneous among processors. In this case theses values are also used to compute the load balancing between processors (the s_is). However, as the local memories are regularly flushed, the locality of input data is not as important as in our model and the problem is more on maintaining a good load balancing and

assuring that each load fills the memory cache (*i.e.* during each load trying to compute square sub-matrices of the output matrix).

For completeness, we point the existence of a variant of PERI-SUM, PERI-MAX, where the objective function is to minimize $\max_i\ p(Z_i)$. This problem is also NP-complete, Beaumont *et al.* [2002], and two approximation algorithms for PERI-SUM can be shown to be $\frac{2}{\sqrt{3}}$-approximations for PERI-MAX, Beaumont *et al.* [2002]; Nagamochi and Abe [2007].

1.2 Related Work

In this section we present some important results on PERI-SUM that will be extended in the other sections, or used as comparison points.

1.2.1 Lower Bound

A lower bound to PERI-SUM is known, based on Loomis-Whitney inequality, see Ballard *et al.* [2011b]. Given $\{s_1, \ldots, s_m\}$, there is no better allocation than one where all zones are squares (what is not always possible). Then, for all i, $p(Z_i) \geq 2\sqrt{s_i}$ and thus

$$C_{opt} \geq 2\sum_i \sqrt{s_i} \tag{1.1}$$

with C_{opt} the optimal solution for a given instance of PERI-SUM.

1.2.2 Column-Based

Beaumont *et al.* [2002] propose a dynamic-programming-based solution for PERI-SUM: ColumnBased. More precisely they optimally solve COL-PERI-SUM, a variant of PERI-SUM where the partitioning is forced to be made of columns.

To solve COL-PERI-SUM, ColumnBased computes the function $f_C(q)$ defined as the cost of the best possible partitioning of a rectangle of size $(\sum_{i=1}^{q} s_i)\times 1$, in rectangles of sizes $\{s_1, \ldots, s_q\}$, in C columns and under the assumption that if $i \leq j$, then the column of s_i is not to the right of the column of s_j.

Let C_i be a column of k_i rectangles. Its height is the same than the unary square, thus 1, and its width is equal to the sum of the surfaces of the rectangle in the column, $\sum_{j=i'}^{i'+k_i-1} s_j$. Trivially the sum of the heights of the rectangle is the height of the column and their individual widths are the same, the one of the column. Thus the cost of a column is $1 + k_i \sum_{j=i'}^{i'+k_i-1} s_j$. Then

$$f_C(q) = \begin{cases} 1 + q * \sum_{i=1}^{q} s_i & \text{if } C = 1 \\ \min_{1\leq r\leq q-C+1}(1 + r \times \sum_{i=q-r+1}^{q} s_i + f_{C-1}(q-r)) & \text{otherwise.} \end{cases}$$

In addition to f_C, ColumnBased also computes f_C^{cut} such that $f_C^{cut}(q)$ contains the optimal number of rectangles in the $C-1$ first columns in a partitioning with the q first elements. This value allows ColumnBased to quickly compute the number of elements on each column of the final partitioning. See Algorithm 1.1 for a formal description of ColumnBased.

Algorithm 1.1 : ColumnBased $(R, \{s_1, \ldots, s_m\})$

Input : A set of positive values $\{s_1, \ldots, s_m\}$ such that $\sum s_i = s(R)$ and $s_1 \leq s_2 \leq \ldots \leq s_m$

Output : A number of columns C and $\{k_1, \ldots, k_C\}$ the number of rectangle in each column to optimally solve COL-PERI-SUM.

$s = 0$;
for $q = 1$ **to** m **do**
  $s = s + s_q$;
  $f_1(q) = 1 + s \times q$;
  $f_1^{cut}(q) = 0$;
for $C = 2$ **to** m **do**
  for $q = C$ **to** m **do**
    $f_C(q) = \min_{1 \leq r \leq q-C+1}(1 + r \times \sum_{i=q-r+1}^{q} s_i + f_{C-1}(q-r))$;
    $f_C^{cut}(q) = q - r_{min}$;
$q = m$;
$C_{min} = C$ such that $f_C(m) = \min_{1 \leq C \leq m} f_C(m)$;
for $C = C_{min}$ **downto** 2 **do**
  $k_C = q - f_C^{cut}(q)$;
  $q = f_C^{cut}(q)$;
$k_1 = q$;
return $C_{min}, \{k_1, \ldots, k_{C_{min}}\}$;

Note that ColumnBased is optimal given an order of the $\{s_1, \ldots, s_m\}$. Beaumont *et al.* [2002] show that if $s_1 \leq s_2 \leq \ldots \leq s_m$ then the partitioning returned by ColumnBased is optimal among all the possible orders.

Finally, a naive implement of ColumnBased runs in $O(m^3)$ operations as it has to compute $f_C(q)$ for every $1 \leq C \leq q \leq m$, and for each of these values it needs to evaluate $(1 + \sum_{i=q-r+1}^{q} s_i + f_{C-1}(q-r))$ for every $r \in [1, q-C+1]$. However, using convexity property of f_C allows dichotomous search for this last part and thus the actual complexity of ColumnBased is $O(m^2 \log m)$.

1.2.3 Rectangular Recursive Partitioning

Nagamochi and Abe [2007] propose a recursive algorithm based on a Divide-and-Conquer approach: RRP (Rectangular Recursive Partitioning).

First note that the problem considered in Nagamochi and Abe [2007],PERI-SUM-PERFECT-RECTANGLES, is slightly different. Indeed it only looks for partitioning into perfect rectangles (for each Z_k, $Z_k = R(Z_k)$).

Problem 1.2 (PERI-SUM-PERFECT-RECTANGLES)**.** Given a set of m strictly positive rational numbers $\{s_1, \ldots, s_m\}$ such that $\sum s_k = 1$, and the square $S = [0,1] \times [0,1]$, find for each s_k a rectangle $R_k \in S$ such that the area of R_k is s_k, $\bigcup R_k = S$, and such that $\sum p(R_k)$ is minimized.

A solution of PERI-SUM-PERFECT-RECTANGLES is of course a valid solution of PERI-SUM but the contrary is not true, see further for examples.

RRP is mostly based on a simple routine: `Guillotine`. Given a composed zone R and a rational number $\alpha \in [0,1]$, $Guillotine(R, \alpha)$ splits R along the largest dimension into two rectangles of respective areas $\alpha s(R)$ and $(1-\alpha)s(R)$ (see Figure 1.3 for a visualisation and Algorithm 1.2 for a formal definition). In some cases, we may need to perform two `Guillotine` calls in sequence, and we denote it with an additional input parameter. For future utilisation (in particular in Section 1.4) and to reduce the space required by `Guillotine`, we allow it to have more than one input parameter. More specifically, if $(R_1, R_2) = Guillotine(R, \alpha)$, then $Guillotine(R, \alpha, \beta) = (Guillotine(R_1, \beta), R_2)$.

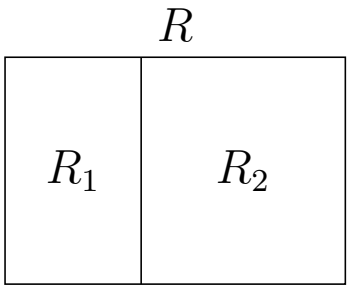


Figure 1.3: Illustration of `Guillotine` routine.

The principle of RRP is to recursively apply `Guillotine` on the unitary square and the rectangles that come from a previous application of `Guillotine`. The splitting of the input surfaces into two sets is made such that each set has at least, if possible, one third of the total surface (*i.e.* we split with $Guillotine(R, \alpha)$ with $\alpha \geq \frac{1}{3}$). For this purpose, surfaces are sorted in increasing order and aggregated one by one until a third of the total is reached. Lemma 1.1 shows that if at least one item remains, then the split is valid. Otherwise, this means that $\sum_{i=1}^{m-1} s_i < \frac{s}{3}$, and hence s_m is significantly larger than the other values, and then RRP splits the rectangle into two rectangles with one only containing s_m. See Algorithm 1.3 for a more formal definition.

Lemma 1.1 (Nagamochi and Abe [2007])**.** *Let $\{s_1, \ldots, s_m\}$ be a set of positive values such that $s_1 \leq \ldots \leq s_m$ and k be the smallest k such that $\sum_{i=1}^{k} s_i \geq \frac{s}{3}$ ($s = \sum_{i=1}^{m} s_i$). Then, if $k < m$, $\sum_{i=k+1}^{m} s_i \geq \frac{s}{3}$.*

Algorithm 1.2 : *Guillotine*(R, α)

Input : A rectangle $R = [x_1, x_2] \times [y_1, y_2]$, $\alpha \in [0, 1]$
Output : Two zones of surfaces $\alpha S(R)$, $(1 - \alpha)S(R)$
$w = w(R)$;
$h = h(R)$;
if $h \leq w$ **then**
  $R_1 = [x_1, x_1 + \alpha w] \times [y_1, y_2]$
else
  $R_1 = [x_1, x_2] \times [y_1, y_1 + \alpha h]$
$R_2 = R \setminus R_1$;
return R_1, R_2

Proof. By definition of k, $\sum_{i=1}^{k-1} s_i < \frac{s}{3}$. Let us assume that $\sum_{k+1}^{m} s_i < \frac{s}{3}$ for the search of a contradiction. In this case we obtain that $s_k \geq \frac{s}{3}$. As $s_{k+1} \leq \sum_{i=k+1}^{m} s_i < \frac{s}{3}$, we have $s_k > s_{k+1}$ which is a contradiction with the hypothesis $s_1 \leq \ldots \leq s_m$. □

Lemma 1.2 (Nagamochi and Abe [2007])**.** *Let R be a rectangle with $\rho(R) \leq 3$, $\alpha \in [0, 1]$ and $R_1, R_2 = Guillotine(R, \alpha)$.*

- *If $\alpha \geq \frac{1}{3}$ then $\rho(R_1) \leq 3$.*
- *If $(1 - \alpha) \geq \frac{1}{3}$ then $\rho(R_2) \leq 3$.*

Proof. Let us assume without loss of generality that $h = h(R) \leq w(R) = w$, and denote $\rho = \rho(R) = \frac{w}{h}$. Then $\rho(R_1) = \min(\frac{\alpha w}{h}, \frac{h}{\alpha w})$. We have $\frac{\alpha w}{h} \leq \frac{w}{h} = \rho \leq 3$ and $\frac{h}{\alpha w} = \frac{1}{\alpha \rho} \leq \frac{3}{\rho} \leq 3$ (under the assumption $\alpha \geq \frac{1}{3}$). Therefore, $\alpha \geq \frac{1}{3}$ implies $\rho(R_1) \leq 3$ and for the same reason, $(1-\alpha) \geq \frac{1}{3\rho}$ implies $\rho(R_2) \leq 3$. □

RRP can be proven to be a $\frac{5}{4}$-approximation of PERI-SUM-PERFECT-RECTANGLES. The proof relies on two main ingredients:

- If the aspect ratio of an output rectangle is below 3, then Lemma 1.3 applies (and $\frac{2}{\sqrt{3}} \leq \frac{5}{4}$).
- Otherwise, thanks to a sophisticated charge transfer technique, it can be proven that there exists a surface significantly larger than some others (the second mode of RRP has been used, otherwise Lemma 1.2 would apply) and the good aspect ratio of this zone compensates the bad aspect ratios of the smaller rectangles.

Lemma 1.3 (Nagamochi and Abe [2007])**.** *Let R be a rectangle. Then:*

$$\frac{p(R)}{2\sqrt{s(R)}} = \frac{1 + \rho(R)}{2\sqrt{\rho(R)}}.$$

Algorithm 1.3 : RRP $(R, \{s_1, \ldots, s_m\})$

Input : A rectangle R, a set of positive values $\{s_1, \ldots, s_m\}$ such that $\sum s_i = s(R)$ and $s_1 \leq s_2 \leq \ldots \leq s_m$
Output : For each $1 \leq i \leq m$, a rectangle R_i such that $s(R_i) = s_i$ and $\bigcup R_i = R$

```
if m = 1 then
    return R
else
    ρ = ρ(R) ;
    s = s(R) ;
    k = the smallest k such that Σ_{i=1}^{k} s_i ≥ s/3 ;
    s' = Σ_{i=1}^{k} s_i ;
    if k < m then
        R_1, R_2 = Guillotine(R, s'/s) ;
    else
        k = m − 1 ;
        R_1, R_2 = Guillotine(R, 1 − s_m/s) ;
    return RRP(R_1, {s_1, ..., s_k}) + RRP(R_2, {s_{k+1}, ..., s_m})
```

In particular, if $\rho(R) \leq 3$ *then,*

$$\frac{p(R)}{2\sqrt{s(R)}} \leq \frac{2}{\sqrt{3}}.$$

Proof. Let us assume without loss of generality that $h = h(R) \leq w = w(R)$ and denote $\rho = \rho(R) = \frac{w}{h}$. Then $p(R) = h + w = h(1 + \rho)$. In addition $s = s(R) = hw = \rho h^2$. Therefore

$$\frac{w + h}{2\sqrt{s}} = \frac{h(1 + \rho)}{2\sqrt{\rho h^2}} = \frac{1 + \rho}{2\sqrt{\rho}}.$$

Furthermore, one can prove that $x \mapsto \frac{1+x}{2\sqrt{x}}$ is an increasing function on $[1, 3]$ and then, for $\rho \leq 3$:

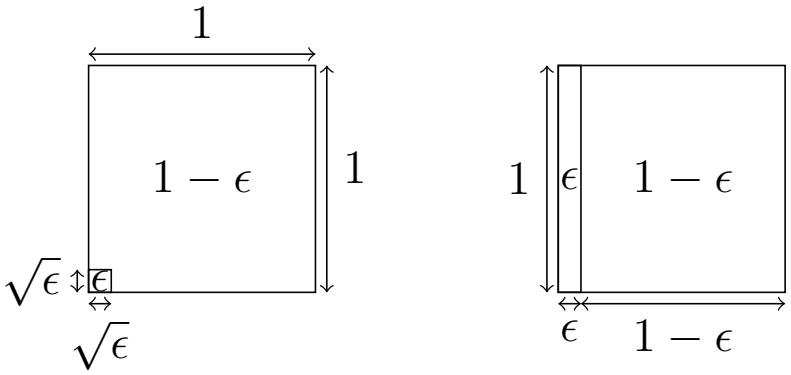
$$\frac{w + h}{2\sqrt{s}} \leq \frac{1 + 3}{2\sqrt{3}} = \frac{2}{\sqrt{3}}.$$

□

First note that if only the first mode of RRP has been used, then the approximation ratio can be lowered to $\frac{2}{\sqrt{3}} \simeq 1.15$ as Fügenschuh *et al.* [2014] pointed. In particular they propose the following sufficient condition: for all $i \in 1, m-1$, $s_{i+1} \leq 2s_i$.

Secondly it is important to clarify that the $\frac{5}{4}$-approximation is only valid for PERI-SUM-PERFECT-RECTANGLES whose worst cases differ from those of

PERI-SUM. Indeed, let us consider an instance with two values ϵ and $1-\epsilon$. If ϵ is small enough, the optimal partitioning is the one depicted on Figure 1.4(a): the communication cost is $2\sqrt{\epsilon}$ for the zone of surface ϵ and 2 for the other zone, what yields a total of $2(1+\sqrt{\epsilon})$. The partitioning returned by RRP is shown on Figure 1.4(b) (see Section 1.2.4). In this case, the communication cost is $1+\epsilon$ for the smallest zone and $2-\epsilon$ for the largest one, which gives a total communication cost of 3. It results that $\frac{Cost(RRP)}{C_{opt}} = \frac{3}{2(1+\epsilon)} \underset{\epsilon \to 0}{\longrightarrow} \frac{3}{2}$. Therefore RRP is at most a $\frac{3}{2}$-approximation of PERI-SUM.



(a) (b)

Figure 1.4: Two partitions of a square with the instance $\{\epsilon, 1-\epsilon\}$.

1.2.4 Optimal Solutions for 2 and 3 processors

Several studies (see below) have attempted to find optimal solutions for PERI-SUM. The problem is heavily combinatorial and thus only solutions for two particular cases, $m = 2$ and $m = 3$, exist. In both cases the same technique is applied, the "push technique", that consists in starting from an arbitrary partitioning and pushing elements on the edge of a zone toward its center, so as to improve the global cost. Eventually the push technique leads to a limited number of cases, which are candidates for optimal solutions.

In the $m = 2$ case, DeFlumere *et al.* [2012] prove there are two dominant cases, the square corner case and the straight line case, see Figure 1.5.

Depending on the ratio between the two processors, both of these solutions may be optimal cases.

Theorem 1.4 (DeFlumere *et al.* [2012])**.** *Let $\{s_1, s_2\}$ be an instance of PERI-SUM such that $s_1 \leq s_2$. Then:*

- *if $\frac{s_2}{s_1} \geq 3$ then the optimal solution for this instance is the Square Corner Partitioning.*

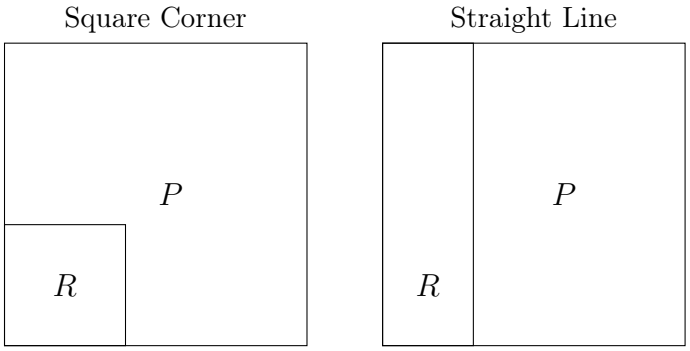


Figure 1.5: The dominant cases with $m = 2$.

- *Otherwise the optimal solution for this instance is the Straight Line Partitioning.*

In the $m = 3$ case, DeFlumere and Lastovetsky [2014] find that there are six dominant cases, see Figure 1.6.

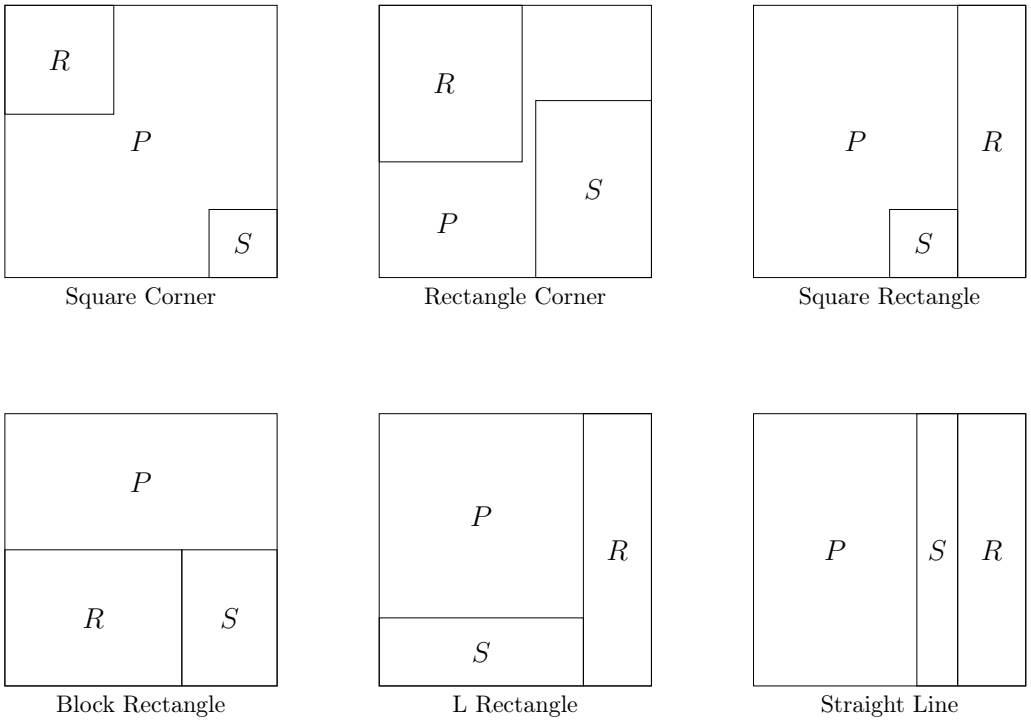


Figure 1.6: The dominant cases with $m = 3$.

However three of the cases never dominate the others. Therefore there are always three choices and depending on the size of the input surfaces, one of these shapes is the optimal case.

Theorem 1.5 (DeFlumere and Lastovetsky [2014]). *The optimal data partition shape for PERI-SUM with $m = 3$ is always one of three shapes: the Square Corner, the Square Rectangle, or the Block Rectangle.*

Note that these results have been extended to different processors topologies. In our modelling we consider that all processors are linked to a central memory and input data are uploaded from it. Other topologies with links between processors have been considered, like star topology (DeFlumere [2014]) or fully connected processors (DeFlumere *et al.* [2014]).

1.2.5 Dynamic Strategies

If dynamic strategies are very commonly used in scheduling for parallel applications, there are few studies for parallel matrix multiplication and its communication avoiding version that is the main subject of this chapter. Beaumont *et al.* [2013] point that simply considering matrix multiplication without data locality (and simply as a set of tasks under the Divisible Load Theory (DLT)) may imply huge communication costs. Beaumont and Marchal [2014] propose a resource-centric dynamic scheduling strategy specific to communication avoiding matrix multiplication, more precisely the outer-product *i.e.* the computation of one layer of C. Here the tasks are of the form

$$T_{i,j} : C_{i,j} \leftarrow C_{i,j} + A_i \times B_j.$$

The strategy, denoted by DynamicOuter, applies the following procedure for every idle processor M that finished its tasks: if $I = \{i,\ M \text{ owns } A_i\}$ and $J = \{j,\ M \text{ owns } B_j\}$, pick i and j at random such that $i \notin I$ and $j \notin J$ and allocate to M all the $T_{i,j'}$s and $T_{i',j}$s such that $i' \in I$ and $j' \in J$, see Algorithm 1.4.

Algorithm 1.4 : DynamicOuter

while *There is unprocessed task* **do**
- Wait for an idle processor M ;
- $I = \{i,\ M \text{ owns } A_i\}$;
- $J = \{j,\ M \text{ owns } B_j\}$;
- $i, j =$ two random integers such that $i \notin I$, $j \notin J$;
- Allocate $\{T_{i,j}$ if unprocessed $\} \cup \{$unprocessed $T_{i,j'},\ j' \in J\} \cup \{$unprocessed $T_{i',j},\ i' \in I\}$ to M ;

The basic principle of DynamicOuter is of course to make each load as profitable as possible by allocating all "free" tasks to the processor that is loading new input data. Beaumont and Marchal [2014] also provide theoretical analysis of DynamicOuter and a notable improvement. Indeed they show theoretically

and experimentally that if more than a certain percent of the whole set of tasks has been computed (98.5% for the outer-product, around 95% for the whole matrix multiplication) then having a second phase of scheduling where tasks are randomly chosen without data locality consideration decreases the amount of communication. This improvement comes from the fact that at the end of DynamicOuter there is high probability that loading a random A_i may provide no free unprocessed tasks.

1.3 Comparison Between Static, Dynamic and Hybrid Strategies in Static and Dynamic Settings

In this section, we are interested in the comparison between pre-existing static and dynamic strategies. Our main concern is to prove that even with the risk of instability and the lack of information on the performance of our resources, the search for efficient static strategies (see Sections 1.4 and 1.5 and Chapter 2) may improve the efficiency of the application. To do this we propose hybrid strategies that use the best of the two, dynamic and static approaches. In this section we only use simulations, practical implementation of the problem will be discussed in Chapter 3.

1.3.1 Presentation of the Targeted Platforms

For the sake of realism, we analyse the behaviour of several algorithms (see Section 1.3.3) on four different target platforms.

The two first platforms, which we denote by Homogeneous-5-CPUs and Homogeneous-20-CPUs, are homogeneous platforms consisting respectively of 5 and 20 CPUs. Note that due to non-determinism of processing times, these platforms will turn out to be heterogeneous in practice, but it corresponds well to the homogeneous machines that can be found either in multicore HPC systems or in datacenters.

In addition, we consider the Heterogeneous-1-GPU and the Heterogeneous-4-GPUs heterogeneous platforms, that correspond well to machines consisting of a few accelerators, such as GPU units, and a few multicore nodes. More specifically, Heterogeneous-1-GPU consists in 1 GPU and 4 CPUs and Heterogeneous-4-GPUs consists in 4 GPUs and 16 CPUs. Timings used for the relative performance of GPUs and CPUs are a coarse approximation of what can be achieved with regular matrix product operations on both devices: in our simulations, we assume a GPU is 50 times faster than a regular CPU core (in Beaumont *et al.* [2016a], for example the ratio is close to 30, the ratio we choose is then slightly more important than in this particular case but with

Name	Distribution	Parameters
UNIFORM-0.80	Uniform in $[a, b]$	$a = 0.8, b = 1.2$
UNIFORM-0.95		$a = 0.95, b = 1.05$
GAUSSIAN-0.1	Truncated Gaussian $\max(\mathcal{N}(\mu, \sigma^2), 0)$	$\mu = 1, \sigma = 0.1$
GAUSSIAN-0.5		$\mu \simeq 0.97, \sigma = 0.5$
GAUSSIAN-1		$\mu \simeq 0.48, \sigma = 1$
TWOMODES-2	v_1 with prob. 0.99 v_2 with prob. 0.01	$v_1 = 1/1.01, v_2 = 2/1.01$
TWOMODES-10		$v_1 = 1/1.09, v_2 = 10/1.09$

Table 1.1: Probability distributions for execution times. The processing time of a task processed by M_l is Xw_l, where X is drawn with the corresponding distribution. All distributions have an expected value of 1.

similar order of magnitude).

As already stated, our goal is also to model non-determinism in resource performance. More precisely, the processing time of the $i-$th task on resource M_l will be given by a random function depending on a specific probability distribution. In order to do a fair comparison of makespan and communication, we fix a value w_{CPU}, which is the expected processing time for the CPUs for all the probability distributions studied here. The expected processing time of the GPUs is set to $\frac{1}{50}w_{CPU}$ (in our model, a GPU is expected to be 50 time faster than a CPU). For a processor M_l this expected duration is denoted w_l. We consider three classes of probability distributions:

- UNIFORM-0.80 and UNIFORM-0.95: the computation time w_i is drawn uniformly on an interval $[w_l - x, w_l + x]$ with $x = 0.2w_l$ for UNIFORM-0.80 and $x = 0.05w_l$ for UNIFORM-0.95.

- GAUSSIAN-0.1, GAUSSIAN-0.5, GAUSSIAN-1: the computation time w_i is drawn under Gaussian law centred on w_l. To avoid sub-zero computation time the Gaussian distribution is truncated and therefore the center of the distribution is in fact lower than w_l in order to keep w_l as the expected value.

- TWOMODES-2, TWOMODES-10: the computation time w_i may take two values, the normal one, close to w_l, with probability 0.99, and a degraded one, close to w_l multiplied by a constant factor, with probability 0.01.

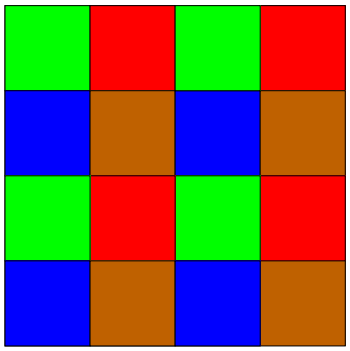
Parameters are described in Table 1.1. Throughout the whole set of simulations, the estimations are obtained with the same algorithm. Each resource processes 5 tasks, whose execution times are chosen at random according to the relevant distribution. Then, the estimated processing time will be taken as the mean value of these 5 measurements. This typically corresponds to classical algorithms used in runtime systems.

Before presenting the strategies we use in our simulations, we provide a small discussion on an important issue that comes from the differences between the PERI-SUM model and the practical matrix multiplication.

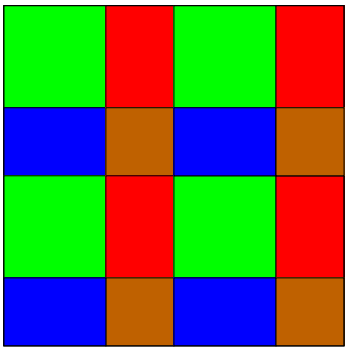
1.3.2 Discrete Aspect of Matrix Partitioning

In PERI-SUM we are considering the partitioning of the unitary square and allow any continuous split. However the application we are considering needs a partitioning for a matrix that is in fact discrete. Therefore the application of a solution of PERI-SUM to partition a matrix may imply roundings that could degrade load balancing.

As an illustration, let us consider a platform with 16 identical processors (with speed 1) and a matrix of size $N = 10$. With this input, RRP and ColumnBased return the same optimal partition, 16 squares each with a half-perimeter of 2.5, see Figure 1.7(a). After rounding, in the discrete partition (depicted in Figure 1.7(b)), some processors are assigned $3 \times 3 = 9$ tasks while other processors are only assigned $2 \times 2 = 4$ tasks. In this case, the rational optimal makespan would be $100/16 = 6.25$, but the makespan of the partition returned after the rounding is 9.



(a) Partition from RRP and ColumnBased



(b) Partition from RRP and ColumnBased after rounding

Figure 1.7: Comparison between the results of the theoretical continuous partitioning and its discrete version, for 16 identical processors and a matrix of size 10.

Let us denote RRP-Rounded and ColumnBased-Rounded the rounded versions of RRP and ColumnBased. Table 1.2 provides the makespan and communication ratios in the case $N = 50$ (the parameter we use later for our simulations) for the different platforms (and constant processing times). The communication ratio is indeed good (much better, as expected, than the worst case bound $\frac{5}{4}$), however the ratio for the makespan, that would be 1 if rounding was not used, is much worse than expected. In particular, on heterogeneous platforms, the makespan ratio can be as high as 1.38 for the platform

HETEROGENEOUS-4-GPUS. Moreover, we can observe that the situation gets worse when new processors are added.

	ColumnBased-Rounded		RRP-Rounded	
	Makespan	Communication	Makespan	Communication
Homo-5	1.02	1.02	1.02	1.02
Homo-20	1.04	1.01	1.08	1.03
Hetero-1	1.02	1.04	1.12	1.06
Hetero-4	1.04	1.04	1.38	1.00

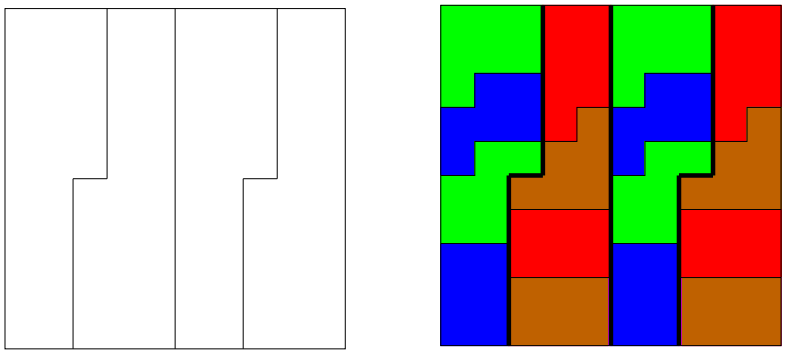
Table 1.2: Ratio between the the result of ColumnBased-Rounded or RRP-Rounded and the lower bounds in the case of constant and well-estimated speeds

As stated before, the initial matrix, that may be huge, is split into blocks whose size is chosen so as to be well adapted to all types of resources and these blocks are the elementary items we want to allocate. For our current application the block size needs to represent a good trade-off between large granularity to fully exploit accelerators like GPUs and fine granularity to have good behaviour on regular cores. Thus block sizes of order 1000 are required and the number of blocks is therefore relatively small: for our test platform, realistic problem sizes correspond to square matrices of several tens of thousands of order. Thus we consider rather small number of blocks ($N = 50$ which corresponds to a matrix size of 50000×50000), so that the effect of rounding errors is important and cannot be neglected.

In order to deal with rounding errors, we have implemented two new algorithms, ColumnBased-Accurate and RRP-Accurate, that allow the assignment of non-rectangular zones, while remaining close to the partition provided by ColumnBased or RRP. This is achieved with the following procedure. By design, the output of ColumnBased is divided in a certain number k of columns, $C_1, \ldots, C_k$, and each processor is assigned to a certain column. The idea is to redefine the frontier between these columns so that each column has the correct surface (what is not the case in Figure 1.7(b), where the first column has 30 tasks instead of 25). To do so, we go through the matrix column by column until the target number of tasks for this column is reached (see Figure 1.8(a)), and we later proceed similarly with rows (see Figure 1.8(b)) so that each cell contains exactly $\lceil \frac{s_k N^2}{\sum_{k'} s_{k'}} \rceil$ or $\lfloor \frac{s_k N^2}{\sum_{k'} s_{k'}} \rfloor$ tasks.

RRP-Accurate is designed along the same ideas and is illustrated in Figure 1.9.

Table 1.3 depicts the results achieved by ColumnBased-Accurate and RRP-Accurate under the same conditions as in Table 1.2. We can observe that RRP-Accurate is more efficient in terms of makespan than RRP-Rounded and achieves similar results with respect to data exchanges.



(a) Previous split for Column-Based

(b) Partition from ColumnBased-Accurate

Figure 1.8: Illustration of ColumnBased-Accurate

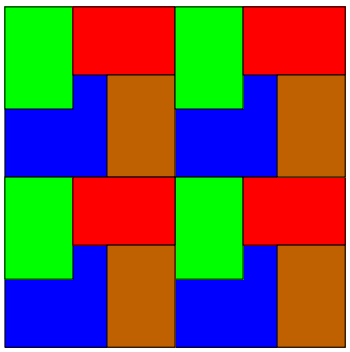


Figure 1.9: Illustration of RRP-Accurate

As RRP-Accurate and ColumnBased-Accurate may "add" some piece of rectangles of width or height 1 to each of the rectangles obtained with RRP or ColumnBased, the worst case additional cost in terms of communication is $O(m)$ where m is the number of processors (to compare with the $2N \times \sum_{i=1}^{m} \sqrt{s_i}$ that is a lower bound for PERI-SUM for a square of size $N \times N$).

In general the discrete variant of PERI-SUM has not been studied to the best of our knowledge. The SFCP algorithm introduced in Section 1.5 is the only known static algorithm for this problem.

1.3.3 Presentation of the strategies

In this section, we provide a short summary of the strategies that will be used in the simulations. They are divided in three categories, static strategies that are based on pre-allocation of the tasks, dynamic strategies that are based on allocation during the execution and hybrid strategies that rely on both techniques.

	ColumnBased-Accurate		RRP-Accurate	
	Makespan	Communication	Makespan	Communication
Homo-5	1	1.03	1	1.03
Homo-20	1	1.04	1	1.05
Hetero-1	1.02	1.07	1.02	1.08
Hetero-4	1.04	1.12	1.04	1.05

Table 1.3: Ratio between the the result of ColumnBased-Accurate and RRP-Accurate and the lower bounds in the case of constant and well-estimated speeds.

Static

As discussed in the previous section, we adapt ColumnBased and RRP to make them usable for discrete partitioning. We then have four strategies, ColumnBased-Rounded, ColumnBased-Accurate, RRP-Rounded and RRP-Accurate. We showed in Section 1.3.2 than ColumnBased-Rounded and RRP-Rounded may fail to reach optimal makespan. However, by coupling them with dynamic strategies (see Hybrid strategies subsection) this default can be avoided and it is interesting in this case to compare the total communication cost of this strategy with ones based on optimal-makespan partitioning.

Dynamic

For these simulations, we decide to use two dynamic strategies, a resource-centric one and a task-centric one. Let us recall that in a resource-centric strategy, processors choose their tasks (often when one is idle); whereas in a task-centric approach, tasks choose the processor on which they will be processed (often when a new task is ready or submitted).

The task-centric strategy, that we denote MCT (Min Cost Time), is an adaptation of the HEFT heuristic, Topcuoglu *et al.* [2002]. Basically, to allocate a task, the MCT scheduler simply chooses the processor that will finish this task the earliest. The estimation is made using the current load of the different processors and the performance estimation that is regularly updated.

The resource-centric strategy we use is MinCost that is close from DynamicOuter that is introduced in Section 1.2.5. More precisely, instead of choosing a row and a column at random (which is the DynamicOuter procedure), an idle processor in MinCost strategy picks a task with the minimum cost (number of row/column to load, *i.e.* 0, 1 or 2) and then loads, if they are not already in its local memory, the corresponding row and column. All the available tasks that become free (with cost 0) when loading these column and/or row are then attributed to this processor. In our simulations we also decide to use another natural feature of resource-centric algorithms, *replication*: at the end of the computation, when a resource becomes idle but no task is available, an already started (and unfinished) task is replicated on this resource, in the hope

that this will allow to finish the task earlier. Such strategies are quite rare in HPC scenarios, but have been extensively used in the Grid Computing community Cirne *et al.* [2007]. The consensus is that the most efficient approach is WQR, which replicates all running tasks equally up to a certain limit on the number of replications. In our experiments, only the fast resources (*i.e.* GPUs) are allowed to duplicate tasks, in order to make sure that most replications actually do improve the finish time of the task.

Hybrid

The goal of hybrid strategies is to retain the best of static and dynamic strategies: the far-sight and the good theoretical quality output by static approaches, the adaptability of dynamic approaches.

Basically, our hybrid strategies are composed of two waves. During the first one, a static strategy (chosen among the ones presented above) is applied to build a first allocation before the beginning of the execution. Then, each time a processor is idle and there are unprocessed tasks, MinCost is applied. Idle processors are allowed to perform tasks allocated to other processors. This heuristic is close to the work-stealing strategy presented in Lima *et al.* [2012], where the resource selects a victim to steal from. The stealing criterion may be driven by locality to reduce data movements, as in Bueno *et al.* [2012]. These strategies are used in runtime systems like PaRSEC (Bosilca *et al.* [2013]), SMPSs (Badia *et al.* [2009]) or KAAPI (Gautier *et al.* [2007]).

We thus have four hybrid strategies, one for each static strategy and denote them Hybrid-RRP-Rounded, Hybrid-RRP-Accurate, Hybrid-ColumnBased-Rounded and Hybrid-ColumnBased-Accurate.

1.3.4 Experimental Results

In this section we present the results of the simulations for the different algorithms presented above. In a first subsection we describe the case where the settings are completely static, i.e the speeds do not change during the execution. Dynamic settings are presented in the second subsection. Let us recall that there are two purely dynamic strategies, MCT (task-centric strategy) and MinCost (resource-centric strategy) and 4 variants of static (resp. hybrid) algorithms. Finally, for each heuristic and platform, we perform 50 runs. In order to facilitate the comparison between the different strategies, the results have been normalized using the expected communication cost (resp. makespan) based on the lower bound from (1.1) (respectively on $\frac{s_i}{\sum s_j} N^2$).

Static Settings

When the performance of the resources is constant over time and well estimated (the execution time of a task is always w_{CPU} for a CPU and w_{GPU} for a GPU),

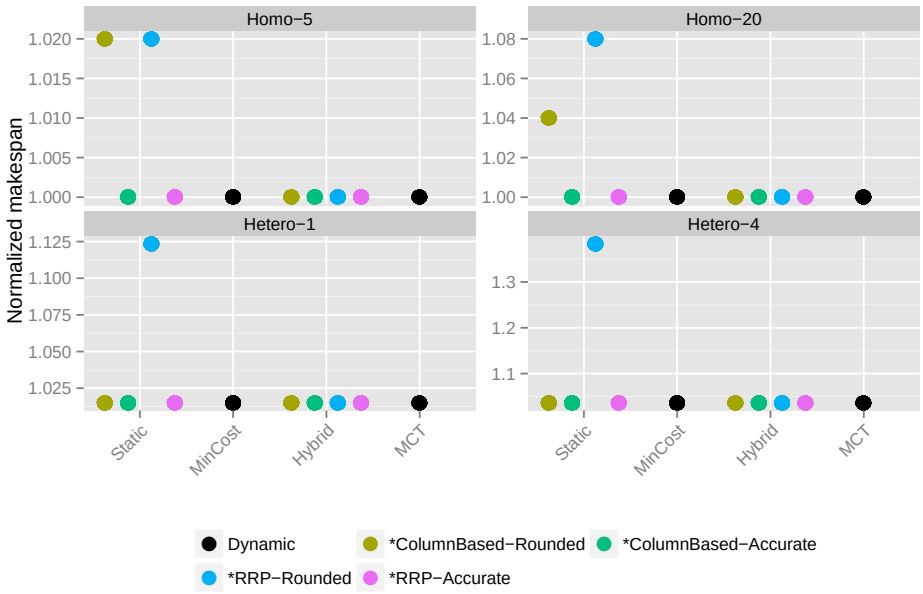


Figure 1.10: Makespan for different algorithms for the four platforms and for static setting.

static heuristics are based on fully accurate previsions. In practice, one can notice on Figure 1.10 that, except ColumnBased-Rounded and RRP-Rounded because of rounding errors, all algorithms are close to the optimal. The small difference for the heterogeneous platforms comes from the fact that some last tasks may be allocated to a CPU instead of GPU (which was busy at that time working on another task). This small difference can be solved with task duplication (see Figure 1.11). The small difference from 1 in both cases is due to imperfect load-balancing. For example, the lower bound we are considering for HETEROGENEOUS-1-GPU is $\frac{N^2}{54} w_{CPU}$ ($54 = 50+1+1+1+1$, the number of tasks performed by unit of time for the whole platform) and to reach this lower bound, the GPU needs, to execute $\frac{50}{54}N^2$ tasks, $\frac{1}{54}N^2$ tasks for the CPUs. However, as $N = 50$, these values are not integer ($\frac{1}{54}N^2 \simeq 46,296$) and the lower bound is not achievable.

As for the communication costs, we can see in Figure 1.12 (and Figure 1.13) that static heuristics perform better, as expected, and the "accurate" versions do not create a significant extra cost. At the same time, hybrid strategies perform well and the ratio with the optimal is always below 1.5. These good results can be explained by the fact that very few job stealing operations take place (except for Hybrid-ColumnBased-Rounded on HETEROGENEOUS-4-GPUS where the task balancing is quite odd, GPUs have a relatively unbalanced repartition of tasks, which does not degrade the makespan, but implies many jobs stealing). For purely dynamic strategies, the communication cost is

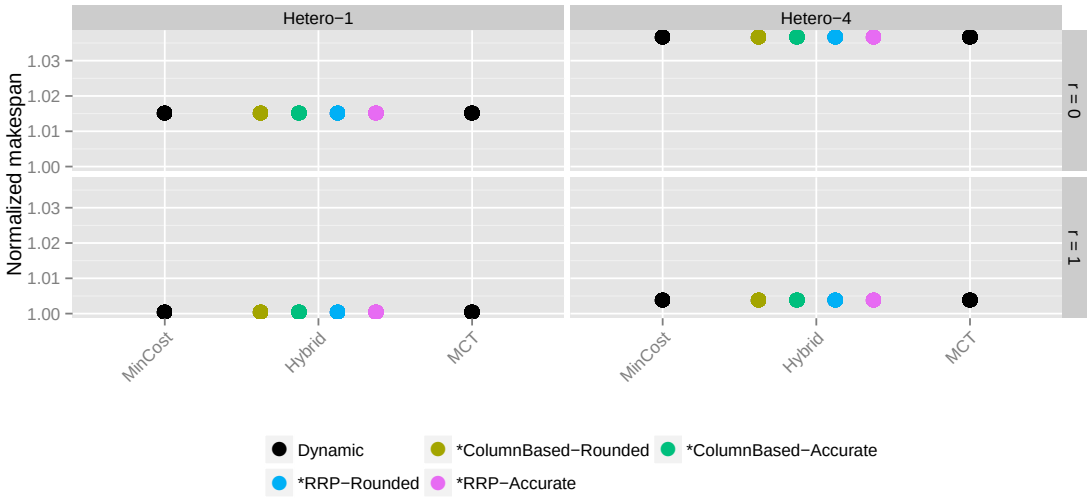


Figure 1.11: Makespan for different algorithms for HETEROGENEOUS-1-GPU and HETEROGENEOUS-4-GPUS for static setting with zero or one allowed replication per task.

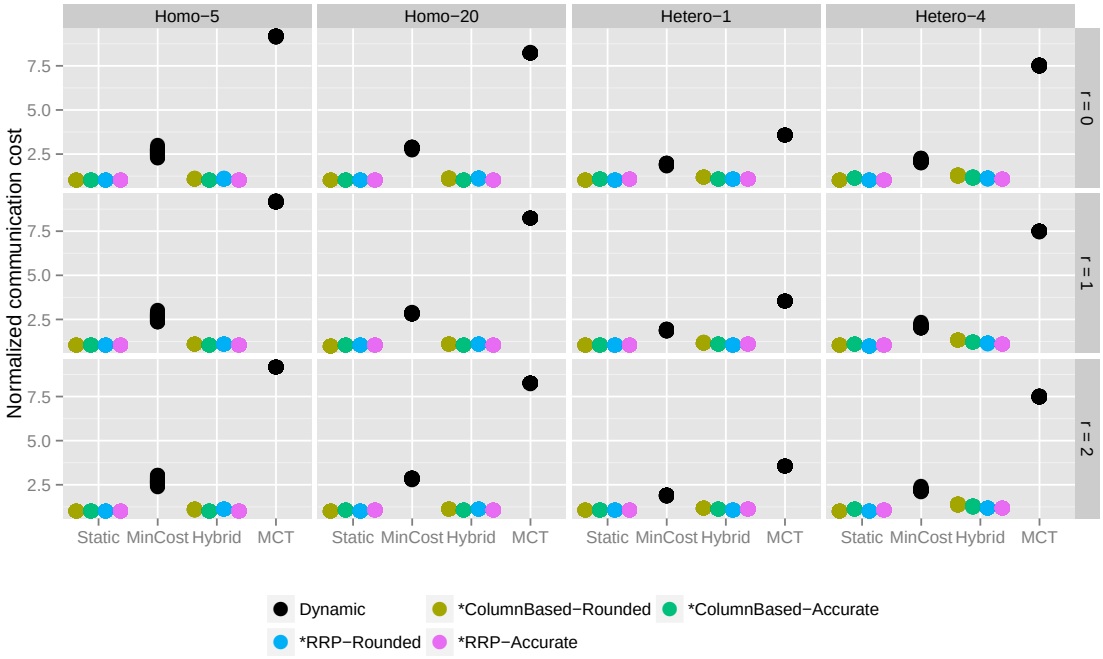


Figure 1.12: Communication cost of static, dynamic and hybrid strategies for static setting

larger and thus they have been excluded from Figure 1.13 to have a better view on the performance of hybrid strategies. For MinCost, the ratio with the optimal value is close to 2 for heterogeneous platforms and 2.5 for homogeneous ones. For MCT, it is between 7.5 and 9, depending on the platform.

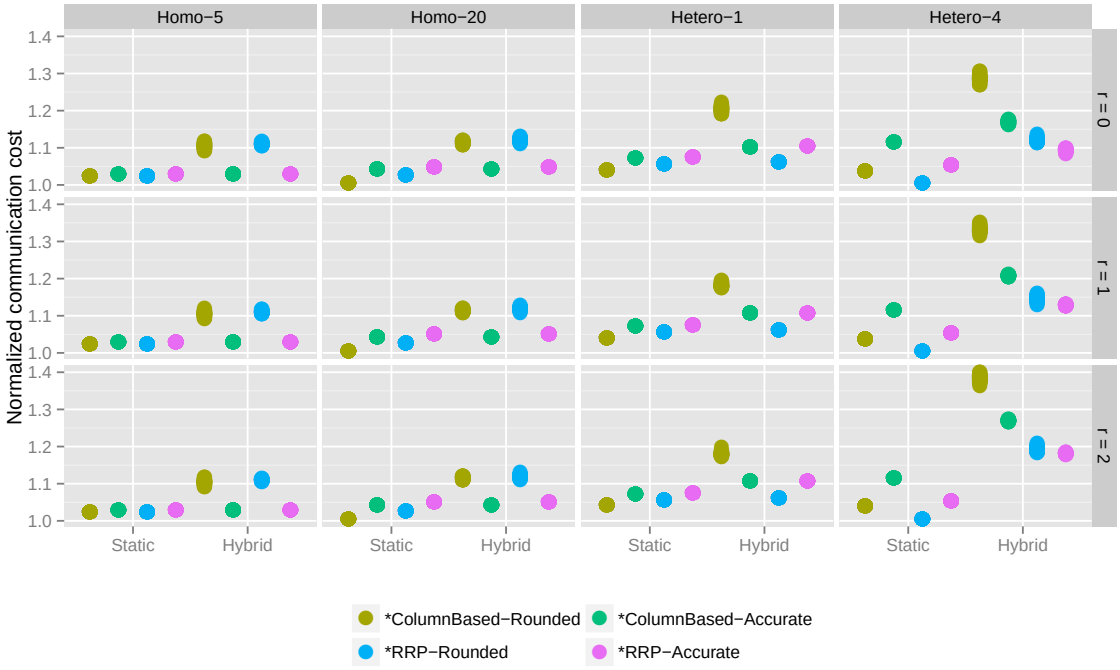


Figure 1.13: Communication cost of static, dynamic and hybrid strategies for static setting

Dynamic Settings

In practice, performance is rarely constant over time and perfectly known, something that explains the practical success of dynamic strategies.

Let us first consider makespan, since indeed our goal is to minimize the execution time. In this setting, static heuristics fail to achieve this objective, and results get worse when the variance of speeds increases, like for Gaussian-0.5, Gaussian-1 or TwoModes-10, where the ratio on the makespan can be larger than 2, see Figure 1.14. This can be explained by the bad estimations of the speeds and by the fact that one processor can sometimes have a really bad execution (a behaviour that is represented with TwoModes-10). In the case of smaller variance, in particular Uniform-0.95, the results are better and close to the expected time of "accurate" versions, but the ratios increase with the number of processors, the risk of a bad estimation increasing in this case. For the heuristics with a dynamic part (see Figure 1.15), replication is compulsory to achieve a good makespan, in particular when the variance

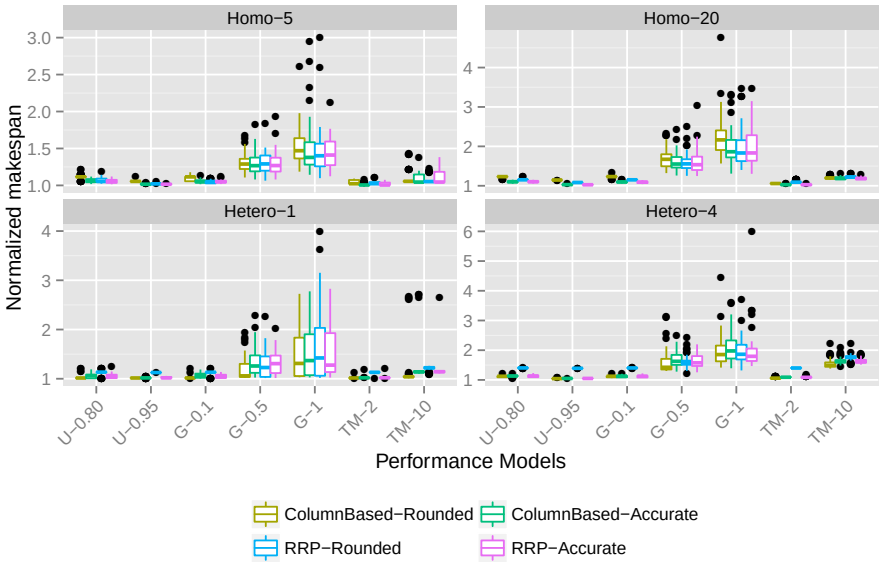


Figure 1.14: Makespan without replication

is large. However, one replication per task is enough. The reason of such a bad execution time (1.6 in the worst case for Heterogeneous-4-GPUs) when there is no replication is that CPUs are really slower than GPUs, so that assigning a task to a CPU instead of a GPU makes a large difference on the overall processing time. The replication mechanism fixes this problem. For homogeneous platforms the optimal is closer, because there is no "wrong" attribution.

For the communication cost results (Figure 1.16 and Figure 1.17), static algorithms are excluded since they achieve similar performance than in the static case, but their makespan is very large. We also excluded MCT in Figure 1.17 order to keep it readable, since the communication cost induced by MCT is really higher than those of the other algorithms (it is larger than 8 for Heterogeneous-4-GPUs or Homogeneous-20-CPUs). For the other algorithms, MinCost and the hybrid ones, we only consider the case where we allow one replication per task, as explained previously. First, we can notice that even if MinCost has the worst ratio of these five algorithms, it stays below 3. We can also notice that MinCost exhibits a better robustness against the variance of the platform.

Hybrid strategies suffer more of an increase in the variance since their static assignment becomes less effective, in particular because of the poor reliability of speed estimations. However the ratio is less than 1.5 (even less than 1.25 for Heterogeneous-1-GPU) for small variance (Gaussian-0.1, TwoModes-2, Uniform-0.80 and Uniform-0.95) and less than 2 in most cases even in presence of large variance, what is always better than MinCost. One can notice

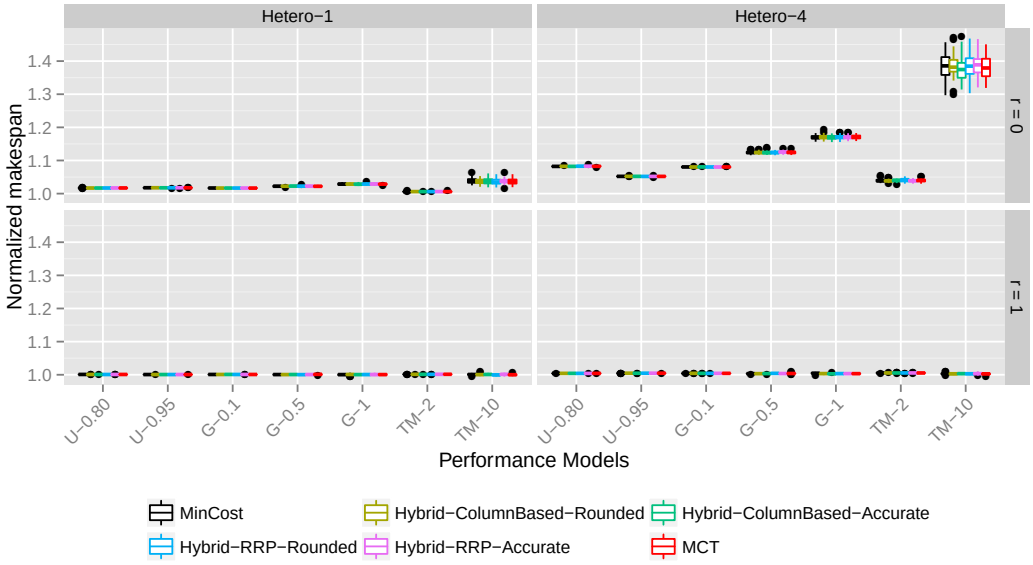


Figure 1.15: Makespan for different levels of replication and different algorithms

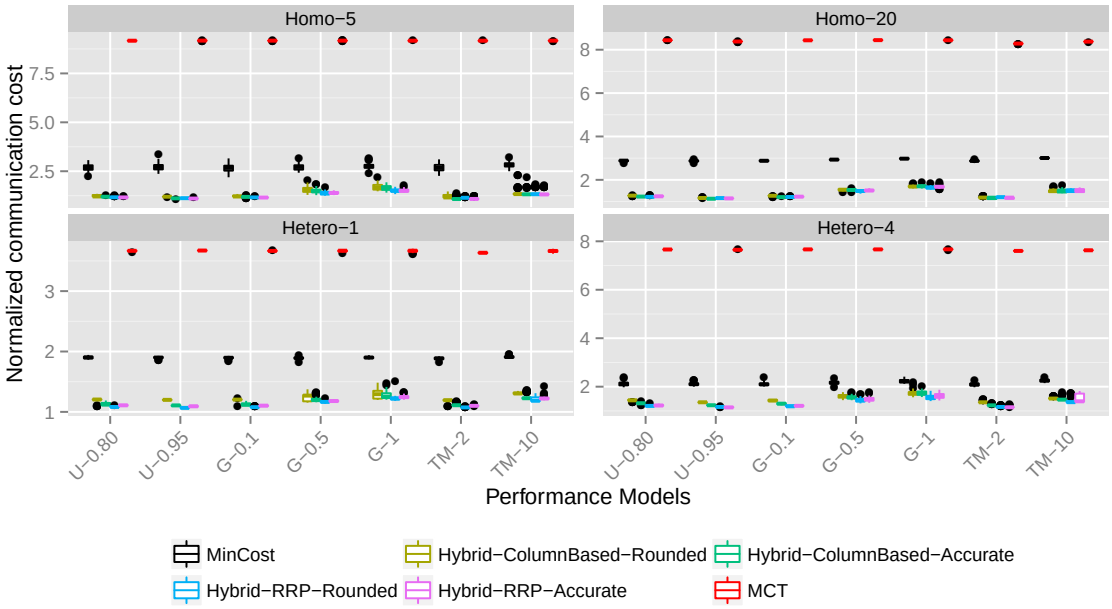


Figure 1.16: Communication cost of dynamic and hybrid strategies for dynamic setting.

that Hybrid-RRP-Accurate, which achieves a better balance, is more effective for low variance since there is less job stealing. In the case of high variance, the dispersion of the results makes it difficult to have a clear hierarchy, even if Hybrid-RRP-Rounded is a little better because it has a cheaper initial static repartition.

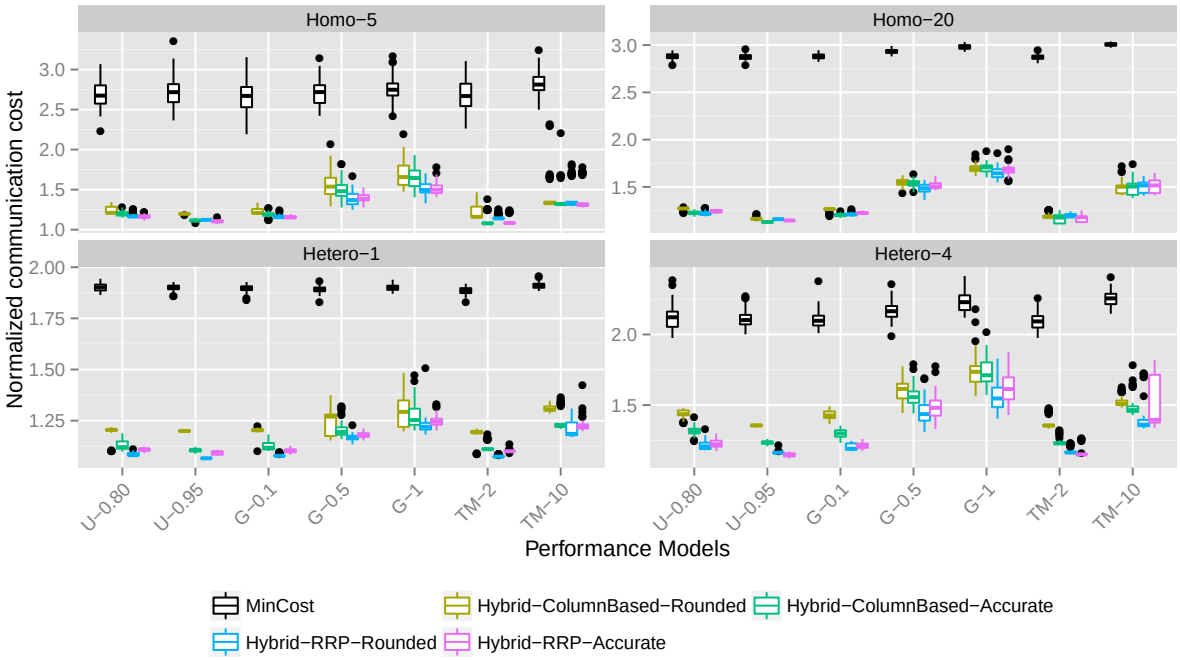


Figure 1.17: Communication cost of MinCost and hybrid strategies for dynamic setting.

Discussion

From these simulations we can retain several facts.

(i) Resource-based strategies are more efficient when we want to minimize communications, and they can be time-optimal for parallel matrix multiplication.

(ii) Purely static partitioning is not reliable enough to be used in practice, but adding a dynamic part to them may be both cost-efficient and time-optimal.

(iii) Hybrid strategies are significantly better, in particular when the variance is low, with an efficient static heuristic. This point justifies our search for new algorithms to solve PERI-SUM.

(iv) The practical implementation might be difficult (in particular it requires to be able to send messages to others processors saying that the task they may be working on has been finished by another processor) but task duplication brings a real improvement to the makespan, in particular on heterogeneous platforms.

1.4 NRRP (Non-Rectangular Recursive Partitioning)

In this section we focus on divide-and-conquer-type algorithms, using RRP as a basis and improving it by adding some new routines and subcases. The main result, presented in Section 1.4.2, is the design of NRRP, a $\frac{2}{\sqrt{3}}$-approximation algorithm.

1.4.1 SNRRP (Simple Non-Rectangular Recursive Partitioning)

As a first step to improve RRP, we use the insight from the optimal solution of Section 1.2.4: whenever a value is significantly larger than the others, it is best to avoid splitting into two rectangles. We thus slightly adapt RRP by adding a new routine in addition to `Guillotine`: `Square`, depicted in Figure 1.18. `Square` is inspired from the work of DeFlumere *et al.* [2012] already presented in Section 1.2.4. Given a rectangle R and a rational number $\alpha \in [0, 1]$, $Square(R, \alpha)$ returns a square R_1 of area $\alpha s(R)$ and a zone Z_2 which corresponds to the initial rectangle R punched by square R_1. The covering rectangle of Z_2 is R and Z_2 will always be used to host an output zone, see Algorithm 1.5 for a formal definition of `Square`.

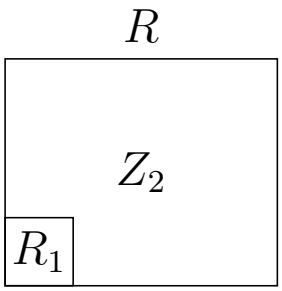


Figure 1.18: Illustration of `Square` routine.

Algorithm 1.5 : $Square(R, \alpha)$

Input : A rectangle $R = [x_1, x_2] \times [y_1, y_2]$, $\alpha \in [0, 1]$
Output : Two zones of area $\alpha S(R)$, $(1 - \alpha)S(R)$
$s = s(R)$;
$R_1 = [x_1, x_1 + \sqrt{\alpha s}] \times [y_1, y_1 + \sqrt{\alpha s}]$;
$Z_2 = R \setminus R_1$;
return R_1, Z_2

SNRRP

With `Guillotine` and `Square` we propose an improved algorithm: SNRRP (Simple Non-Rectangular Recursive Partitioning), given in Algorithm 1.6. The basic idea is the following: according to an appropriate condition, the current rectangle is either split into two well-shaped rectangles (lines 7-9), or into a square and its complement, the latter being assigned to a single processor (lines 12-13).

Algorithm 1.6 : SNRRP $(R, \{s_1, \ldots, s_m\})$

Input : A rectangle R, a set of values $\{s_1, \ldots, s_m\}$ such that $\sum s_i = s(R)$ and $s_1 \leq s_2 \leq \ldots \leq s_m$
Output : For each $1 \leq i \leq m$, a zone Z_i such that $s(Z_i) = s_i$ and $\bigcup Z_i = R$

```
1  if m = 1 then
2      return R
3  else
4      ρ = ρ(R) ;
5      s = s(R) ;
6      k = the smallest k such that Σ_{i=1}^{k} s_i ≥ s/(3ρ) ;
7      s' = Σ_{i=1}^{k} s_i ;
8      if k < m then
9          R_1, R_2 = Guillotine(R, s'/s) ;
10         return SNRRP(R_1, {s_1, ..., s_k}) + SNRRP(R_2, {s_{k+1}, ..., s_m})
11     else
12         R_1, Z_2 = Square(R, (s − s_m)/s) ;
13         SNRRP(R_1, {s_1, ..., s_{m−1}}) + Z_2
```

This algorithm can be proven to be a $\sqrt{\frac{3}{2}}$-approximation ($\sqrt{\frac{3}{2}} \simeq 1.22$) and thus be seen as a significant improvement of RRP. The proof relies on one major invariant (that also ensures the correctness of the algorithm): each rectangle on which the function is called has an aspect ratio below 3, as is

depicted on Theorem 1.8. In order to prove it, we need the following lemmas that are generalisations of Lemma 1.1 and Lemma 1.2.

Lemma 1.6. *Let $\{s_1, \ldots, s_m\}$ be a set of positive values sorted in non-decreasing order, $\rho \geq 1$, and $s = \sum_i s_i$. Let us assume that there exists an index k such that $\sum_{i=1}^{k-1} s_i \geq \frac{s}{3\rho}$, and let us consider the smallest such integer. Then $\sum_{i=k}^{m} s_i \geq \frac{s}{3\rho}$.*

Proof. By definition of k, $\sum_{i=1}^{k-2} s_i < \frac{s}{3\rho}$. Therefore, if we assume that $\sum_{i=k}^{m} s_i < \frac{s}{3\rho}$, we obtain $s_{k-1} = s - \sum_{i=1}^{k-2} s_i - \sum_{i=k}^{m} s_i \geq \frac{s}{3\rho}$ (since $\rho \geq 1$). Since $s_k \leq \sum_{i=k}^{m} s_i < \frac{s}{3\rho}$, we have $s_{k-1} > s_k$ which is a contradiction with the fact that the s_i's are sorted in non-decreasing order. □

Lemma 1.7. *Let R be a rectangle with $\rho(R) \leq 3$, $\alpha \in [0, 1]$ and $R_1, R_2 = Guillotine(R, \alpha)$.*

- *If $\alpha \geq \frac{1}{3\rho(R)}$ then $\rho(R_1) \leq 3$.*
- *If $(1 - \alpha) \geq \frac{1}{3\rho(R)}$ then $\rho(R_2) \leq 3$.*

Proof. Let us assume without loss of generality that $h = h(R) \leq w(R) = w$, and denote $\rho = \rho(R) = \frac{w}{h}$. Then $\rho(R_1) = \min(\frac{\alpha w}{h}, \frac{h}{\alpha w})$. We have $\frac{\alpha w}{h} \leq \frac{w}{h} = \rho \leq 3$ and $\frac{h}{\alpha w} = \frac{1}{\alpha\rho} \leq \frac{3\rho}{\rho} \leq 3$ (under the assumption $\alpha \geq \frac{1}{3\rho}$). Therefore, $\alpha \geq \frac{1}{3\rho}$ implies $\rho(R_1) \leq 3$ and for the same reason, $(1 - \alpha) \geq \frac{1}{3\rho}$ implies $\rho(R_2) \leq 3$. □

Theorem 1.8 (Correctness)**.** *When executing SNRRP $(R, \{s_1, \ldots, s_m\})$ with $\rho(R) \leq 3$, all the recursive calls to SNRRP $(R', \{s'_1, \ldots, s'_k\})$ are performed on a rectangle area R' such that $\rho(R') \leq 3$.*

Proof. There are two cases where there are rectangles on which SNRRP is called, at line 9 and at line 12. In the second case the rectangle is a square produced by `Square` and then its aspect ratio is below 3. In the first case, Lemma 1.6 ensures that Lemma 1.7 applies for both rectangles. □

In order to complete the approximation proof we note that the zones returned by the algorithm are produced at line 2 or 12. In the first case we know that the zone is a rectangle with an aspect ratio below 3 (Theorem 1.8) and a simple use of Lemma 1.3 proves that its half-perimeter is below $\frac{2}{\sqrt{3}}$ times the lower-bound (and $\frac{2}{\sqrt{3}} \leq \sqrt{\frac{3}{2}}$). In the case of line 12, the resulting zone is no longer rectangular and we need an additional lemma.

Lemma 1.9. *Let R be a rectangle such that $\rho(R) \leq 3$, $\alpha \in [0, 1]$ and $R', Z = Square(R, \alpha)$. If $\alpha \leq \frac{1}{3\rho(R)}$ then $\frac{p(Z)}{2\sqrt{s(Z)}} \leq \sqrt{\frac{3}{2}}$.*

Proof. Let us denote $\rho(Z) = \rho$ and $s(Z) = s$. Let us suppose that $h = h(R) \leq w = w(R)$ without loss of generality. Note that the covering rectangle of Z is R and therefore $w = w(Z)$ and $h = h(Z)$. In this case $w = \rho h$, so that $p(Z) = (1+\rho)h$. By definition of `Square`, $s = (1-\alpha)s(R)$ and $s(R) = hw = \rho h^2$. Therefore $s = (1-\alpha)\rho h^2 \geq (1 - \frac{1}{3\rho(Z)})\rho h^2$ and hence

$$\begin{aligned}\frac{p(Z)}{2\sqrt{S(Z)}} &\leq \frac{(1+\rho)h}{2\sqrt{(1-1/3\rho)\rho h^2}} \\ &\leq \frac{\sqrt{3}(1+\rho)}{2\sqrt{(3\rho-1)}}.\end{aligned}$$

One can prove that the function $x \longmapsto \frac{\sqrt{3}(1+x)}{2\sqrt{3x-1}}$ reaches its maximum on $[1,3]$ for $x = 1$ or $x = 3$ and this maximum is $\sqrt{\frac{3}{2}}$. Then $\frac{p(Z)}{2\sqrt{S(Z)}} \leq \sqrt{\frac{3}{2}}$. □

Lemma 1.9 covers the second case of zone creation. Indeed, the hypothesis of the lemma on the aspect ratio of R is ensured by Theorem 1.8 and the hypothesis on α comes from the fact that, at line 11 of Algorithm 1.6, $s - s_m = \sum_{i=1}^{n-1} s_i = \sum_{i=1}^{k-1} s_i < \frac{s}{3\rho(R)}$ by definition of k at line 5.

All of the above proves that if $\{Z_1, \ldots, Z_m\} = SNRRP(R, \{s_1, \ldots, s_m\})$, then for all $i \in [1,m]$, $\frac{p(Z_i)}{2\sqrt{s_i}} \leq \sqrt{\frac{3}{2}}$. Therefore we deduce $\frac{\sum_{i=1}^{m} p(Z_i)}{\sum_{i=1}^{m} 2\sqrt{s_i}} \leq \sqrt{\frac{3}{2}}$. As $\sum_{i=1}^{m} 2\sqrt{s_i}$ is the lower bound from (1.1), we obtain Theorem 1.10.

Theorem 1.10. *SNRRP is a $\sqrt{\frac{3}{2}}$-approximation for PERI-SUM.*

Note that cases exist where SNRRP returns a partition such that the ratio between the sum of perimeters and the lower bound based on (1.1) is indeed $\sqrt{\frac{3}{2}}$. Let us define for $1 < k \leq m$, $s_k = 2/3^{m-k+1}$ and $s_1 = 1/3^{m-1}$. One can notice that $s_k = \frac{2}{3}\sum_{i=1}^{k} s_i$. Hence each step of $SNRRP([0,1]^2, \{s_1, \ldots, s_m\})$ corresponds to the case of line 11-12 and therefore the `Square` routine is called. In addition, this corresponds the extremal position of the case of line 11-12 and we can notice in the proof of Lemma 1.9 that in this case the bound is tight. Hence, if $Z_1, \ldots, Z_m = NRRP([0,1]^2, \{s_1, \ldots, s_m\})$, $p(Z_1) = 2\sqrt{s(Z_1)}$ and for $k > 1$, $p(Z_k) = 2\sqrt{\frac{3}{2}}\sqrt{s(Z_k)}$. Thus,

$$\lim_{m\to\infty} \frac{\sum_{i=1}^{m} p(Z_i)}{2\sum_{i=1}^{m} \sqrt{s(Z_i)}} = \sqrt{\frac{3}{2}}$$

An illustration of this case is depicted in Figure 1.19.

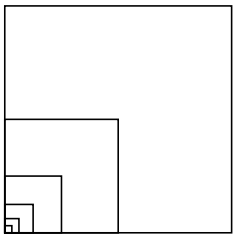


Figure 1.19: Worst case scenario for SNRRP.

Possible Improvement of SNRRP

Further improvements over SNRRP and its approximation ratio can be obtained by changing an invariant. In both RRP and SNRRP, the aim is to obtain a rectangle with an aspect ratio below 3. This can be generalized by aiming for a rectangle with an aspect ratio below μ where μ is a parameter whose value will be discussed. The objective is to use values of $\mu < 3$, so that the zones produced are as close to squares as possible. With values of μ different from 3, above proofs have to be adapted at two places.

First note that Lemma 1.1 is conserved for $\mu \geq 3$ and can be rewritten in Lemma 1.11.

Lemma 1.11. *For $\mu \geq 3$, let $\{s_1, \ldots, s_m\}$ be a set of positive values such that $s_1 \leq \ldots \leq s_m$ and k be the smallest k such that $\sum_{i=1}^{k} s_i \geq \frac{s}{\mu}$. Then, if $k < m$, $\sum_{i=k+1}^{m} s_i \geq \frac{s}{\mu}$.*

However, in the case $\mu < 3$, the lemma has to be significantly weakened, as expressed in Lemma 1.12.

Lemma 1.12. *Let $q > 1$ be an integer and let μ be in $[\frac{2q+1}{q}, \frac{2q-1}{q-1}[$. Let $\{s_1, \ldots, s_m\}$ be a set of positive values such that $s_1 \leq \ldots \leq s_m$ and k be the smallest k such that $\sum_{i=1}^{k} s_i \geq \frac{s}{\mu}$. Then, if $k \leq m - q$, $\sum_{i=k+1}^{m} s_i \geq \frac{s}{\mu}$.*

Proof. By definition of k, $\sum_{i=1}^{k-1} s_i < \frac{s}{\mu}$. Let us assume $\sum_{i=k+1}^{m} s_i < \frac{s}{\mu}$ for the search of a contradiction. In this case we obtain a $s_k \geq \frac{\mu-2}{\mu} s$. In addition $\sum_{i=k+1}^{m} s_i \geq \sum_{i=k+1}^{m} s_{k+1} = (m-k)s_{k+1}$. Therefore $s_{k+1} < \frac{s}{(m-k)\mu}$. From the assumption $k \leq m-q$ we deduce $s_{k+1} < \frac{s}{q\mu}$. Furthermore, $\mu - 2 \geq \frac{2q+1}{q} - 2 \geq \frac{1}{q}$. Hence $s_k \geq \frac{s}{q\mu} > s_{k+1}$ what is a contradiction with $s_1 \leq \ldots \leq s_m$. □

Both lemmas provide a sufficient condition for the possibility to split the list of values such that each part sums to a fraction at least $\frac{1}{\mu}$ of the total, allowing to perform a recursive call on each sublist. When the condition is not met, the implication in the case $\mu \geq 3$ is that there exists a single large value $s_m \geq s(1-\frac{1}{\mu})$, for which a large zone can be accommodated with low communication cost by `Square`. When $\mu < 3$ however, this guarantee is weakened to the

existence of at most $q = \lfloor \frac{1}{\mu-2} \rfloor$ values whose sum is at least $s(1 - \frac{1}{\mu})$. It is thus necessary to design routines that can accommodate q zones with low communication cost.

The conclusion is that using lower values for μ requires the consideration of additional cases when it is not possible to obtain two parts that contain more than $\frac{1}{\mu}$ of the total. The number of terminal zones to accommodate is $q = \lfloor \frac{1}{\mu-2} \rfloor$, and the complexity of designing the corresponding partitioning increases significantly with q.

Changing the value of μ has also an effect on the approximation ratios for the individual zones returned by the algorithm. For the case where the algorithm returns a rectangle, Lemma 1.3 becomes Lemma 1.13, with an identical proof.

Lemma 1.13. *Let R be a rectangle. If $\rho(R) \leq \mu$ then:*

$$\frac{p(R)}{2\sqrt{s(R)}} \leq \frac{\mu+1}{2\sqrt{\mu}}.$$

This function is increasing for the interesting values of μ, and thus decreasing μ allows us to improve the approximation ratio in the case where all the produced zones are rectangles. However, as shown above, this is not the limiting worst-case; indeed, $\frac{2}{\sqrt{3}}$ ($\frac{\mu+1}{2\sqrt{\mu}}$ for $\mu = 3$) is smaller than $\sqrt{\frac{3}{2}}$. Hence one could expect that increasing slightly this part of the bound with a larger value of μ would allow to lower the approximation ratios obtained for zones returned as complements of a square, given in Lemma 1.9. However the generalized version, Lemma 1.14, does not allow this.

Lemma 1.14. *Let R be a rectangle such that $\rho(R) \leq \mu$, $\alpha \in [0, 1]$ and $R', Z = Square(R, \alpha)$. If $\alpha \leq \frac{1}{\mu\rho(R)}$ then $\frac{p(Z)}{2\sqrt{s(Z)}} \leq \max(\sqrt{\frac{\mu}{\mu-1}}, \frac{\sqrt{\mu(\mu+1)}}{2\sqrt{\mu-1}})$.*

Proof. We denote $\rho(Z) = \rho$ and $s(Z) = s$. We suppose that $h = h(R) \leq w = w(R)$ without loss of generality. Note that the covering rectangle of Z is R and therefore $w = w(Z)$ and $h = h(Z)$. In this case $w = \rho h$, which implies $p(Z) = (1 + \rho)h$. By definition of `Square`, $s = (1 - \alpha)s(R)$ and $s(R) = hw = \rho h^2$. Therefore $s = (1 - \alpha)\rho h^2 \geq (1 - \frac{1}{\mu\rho(Z)})\rho h^2$ and hence

$$\begin{aligned}\frac{p(Z)}{2\sqrt{S(Z)}} &\leq \frac{(1+\rho)h}{2\sqrt{(1-1/\mu\rho)\rho h^2}} \\ &\leq \frac{\sqrt{\mu}(1+\rho)}{2\sqrt{(\mu\rho-1)}}.\end{aligned}$$

One can prove that the function $x \longmapsto \frac{\sqrt{\mu}(1+x)}{2\sqrt{\mu x-1}}$ is decreasing on $[1, 1 + \frac{2}{\mu}]$ and increasing on $[1 + \frac{2}{\mu}, \mu]$ and then $\frac{\sqrt{\mu}(1+x)}{2\sqrt{\mu x-1}} \leq \max(\sqrt{\frac{\mu}{\mu-1}}, \frac{\sqrt{\mu(\mu+1)}}{2\sqrt{\mu-1}})$. Then, $\frac{p(Z)}{2\sqrt{S(Z)}} \leq \max(\sqrt{\frac{\mu}{\mu-1}}, \frac{\sqrt{\mu(\mu+1)}}{2\sqrt{\mu-1}})$. □

If $\mu < 3$, then $\max(\sqrt{\frac{\mu}{\mu-1}}, \frac{\sqrt{\mu(\mu+1)}}{2\sqrt{\mu-1}}) = \sqrt{\frac{\mu}{\mu-1}}$. With such a value of μ, the dominant worst case is when the rectangle R is a perfect square (see Figure 1.20(a)) and the low value of μ implies the possibility for the square to take a large portion of the rectangle. If $\mu > 3$, then $\max(\sqrt{\frac{\mu}{\mu-1}}, \frac{\sqrt{\mu(\mu+1)}}{2\sqrt{\mu-1}}) = \frac{\sqrt{\mu(\mu+1)}}{2\sqrt{\mu-1}}$. With such a value of μ, the dominant worst case is when the rectangle R has an aspect ratio of μ (see Figure 1.20(b)) and the large value of μ implies a huge deformation of the rectangle which increases the approximation ratio.

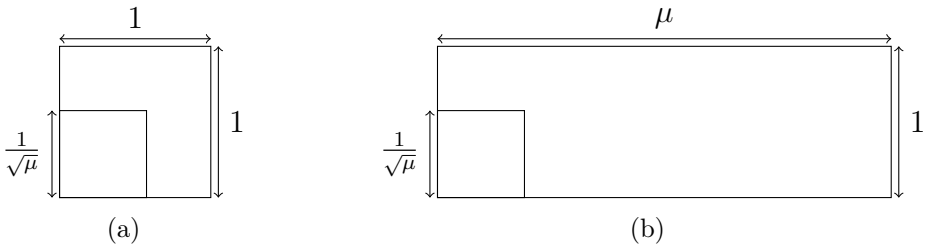


Figure 1.20: Dominant worst case in function of μ for the `Square` routine.

In summary, $x \mapsto \sqrt{\frac{x}{x-1}}$ is decreasing on $]1, 3]$ and $\frac{\sqrt{x(x+1)}}{2\sqrt{x-1}}$ is increasing on $[3, +\infty[$. Therefore, with this technique of proof and with no change on SNRRP, the value $\mu = 3$ yields to the best approximation ratio.

Therefore, to improve SNRRP, we need two things:

- A new routine for splitting in addition to `Guillotine` to deal with rectangles with aspect ratio $\mu < 3$.
- New subcases for pathological cases when `Square` routine does not provide the targeted approximation ratio.

1.4.2 NRRP

In this section, we describe NRRP, a new approximation algorithm to solve PERI-SUM that provides a $\frac{2}{\sqrt{3}}$-approximation ($\frac{2}{\sqrt{3}} \simeq 1.15$). As for RRP and SNRRP, NRRP is based on a divide and conquer paradigm. At each step, it tries to split the actual rectangle into two parts (more in few cases), and is applied recursively on each part. As stated previously, we now aim for a μ smaller than 3 and choose $\mu = \frac{5}{2}$ what is the smallest value such that $q = \lfloor \frac{1}{\mu-2} \rfloor \leq 2$.

As the number of cases increases and to add clarity in the following, we now use the terms **simple** and **composed** zones. Simple zones are terminal and are allocated to a single processor. In what follows, they will be denoted using letter Z. Composed zones are unions of simple zones that are encountered

during the algorithm. In what follows, they will be denoted using letter R (they are always rectangle).

Let us first describe the complete algorithm and the different routines it uses, in addition of the already known `Guillotine` and `Square`. We recall that to reduce space required, we allow `Guillotine` to have more than one input parameter. More specifically, if $(R_1, R_2) = Guillotine(R, \alpha)$, then $Guillotine(R, \alpha, \beta) = (Guillotine(R_1, \beta), R_2)$.

The first new routine is `Tripartition`, depicted in Figure 1.21 and Algorithm 1.7. In some (rare) cases, neither `Guillotine` nor `Square` routines are able to provide either simple zones or composed zones consisting of a rectangle whose aspect ratio is smaller than 5/2. In this case, we use $Tripartition(R, \alpha, \beta)$, that returns three rectangles R_1, Z_2 and Z_3 (in practice, rectangles Z_2 and Z_3 will always be used to host simple zones) of respective areas $\alpha s(R)$, $\beta s(R)$ and $(1 - \alpha - \beta)s(R)$. The difference with the result given by $Guillotine(R, \alpha + \beta, \alpha/(\alpha + \beta)$ is that we do not perform the second split along the largest dimension of R'.

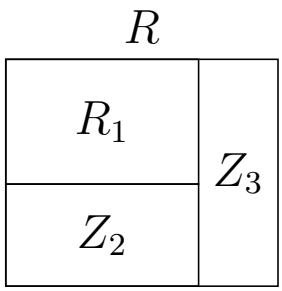


Figure 1.21: Illustration of `Tripartition` routine.

Algorithm 1.7 : $Tripartition(R, \alpha, \beta)$

Input : A rectangle $R = [x_1, x_2] \times [y_1, y_2]$, $\alpha, \beta \in [0, 1]$
Output : Three zones of respective areas $\alpha S(R)$, $\beta S(R)$, $(1 - \alpha - \beta)S(R)$

$w = w(R)$; $h = h(R)$;
if $h \leq w$ **then**
  $w_1 = (\alpha + \beta)w$; $h_1 = \frac{\alpha}{\alpha+\beta}h$;
  $R_1 = [x_1, x_1 + w_1] \times [y_1, y_1 + h_1]$; $Z_2 = [x_1, x_1 + w_1] \times [y_1 + h_1, y_2]$
else
  $w_1 = \frac{\alpha}{\alpha+\beta}w$; $h_1 = (\alpha + \beta)h$;
  $R_1 = [x_1, x_1 + w_1] \times [y_1, y_1 + h_1]$; $Z_2 = [x_1 + w_1, x_2] \times [y_1, y_1 + h_1]$
$Z_3 = R \setminus (R_1 \cup R_2)$;
return R_1, Z_2, Z_3

The second routine is `Superposition` depicted in Figure 1.22 and Algorithm 1.8. $Superposition(R, \alpha, \epsilon)$ returns three zones, R_1, Z_2 and Z_3 of re-

spective areas $\epsilon s(R)$, $(\alpha - \epsilon)s(R)$ and $(1 - \alpha)s(R)$. R_1 is a square that can be placed in the upper left corner, Z_2 a rectangle which is placed under R_1 in the bottom left corner and Z_3 is the remaining zone, *i.e.* R punched by both R_1 and Z_2. In practice, Z_2 and Z_3 will always be used to host simple zones.

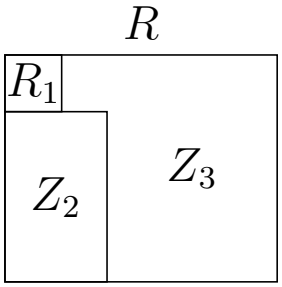


Figure 1.22: Illustration of `Superposition` routine.

Algorithm 1.8 : *Superposition*(R, α, ϵ)

Input : A rectangle $R = [x_1, x_2] \times [y_1, y_2]$, $\alpha, \epsilon \in [0, 1]$
Output : Three zones of respective areas $\epsilon S(R)$, $(\alpha - \epsilon)S(R)$, $(1 - \alpha)S(R)$
$w = w(R)$; $h = h(R)$; $s = s(R)$;
$l = \sqrt{\epsilon s}$; $R_1 = [x_1, x_1 + l] \times [y_1, y_1 + l]$;
if $h \leq w$ **then**
  $w_1 = \frac{(\alpha-\epsilon)hw}{h-l}$; $Z_2 = [x_1, x_1 + w_1] \times [y_1 + l, y_2]$
else
  $h_1 = \frac{(\alpha-\epsilon)hw}{w-l}$; $Z_2 = [x_1 + l, x_2] \times [y_1, y_1 + h_1]$
$Z_3 = R \setminus (R_1 \cup Z_2)$;
return R_1, Z_2, Z_3

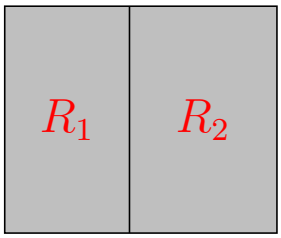
Finally let us define `Packing`. Given a list $\{s_1, \dots s_k\}$ sorted in increasing order, two rational values s and s' and a rectangle R such that $s(R) = \sum s_i$, $Packing(\{s_1, \dots, s_k\}, s, s', R)$ returns a list of couples (R_j, S_j) where R_j is a rectangle and S_j is a subset of $\{s_1, \dots, s_k\}$ such that $\bigcup R_j = R$, $s(R_j) = \sum_{i \in S_j} s_i$ and $s \leq s(R_j) \leq s'$. For shortness, we allow s_k to be larger than s' and in this case $Packing(\{s_1, \dots, s_k\}, s, s', R) = Packing(\{s_1, \dots, s_{k-1}\}, s, s', R_1) + (R_2, s_k)$, where $R_1, R_2 = Guillotine(R, (s(R) - s_k)/s(R))$. We will explicitly discuss the existence of such a function in the condition we use it. We will denote by $Map(NRRP, L)$, with $L = Packing(\{s_1, \dots, s_k\}, s, s', R)$ the function that applies NRRP to each of the R_js.

Now, all necessary ingredients to describe NRRP are there, see Algorithm 1.9. For a more visual approach Figure 1.23 provides all the subcases with the associated lines. Each part and subcase will be discussed and described during the correctness and the approximation ratio proofs.

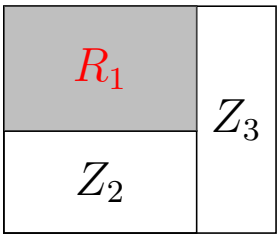
Algorithm 1.9 : NRRP $(R, \{s_1, \ldots, s_m\})$

Input : A rectangle R, a set of values $\{s_1, \ldots, s_m\}$ such that $\sum s_i = s(R)$ and $s_1 \leq s_2 \leq \ldots \leq s_m$
Output : For each $1 \leq i \leq m$, a zone Z_i such that $s(Z_i) = s_i$ and $\bigcup Z_i = R$

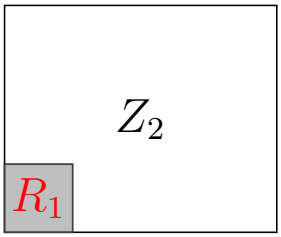
```
1  if m = 1 then
2      return R
3  else
4      ρ = ρ(R) ;
5      k = the smallest k such that Σ_{i=1}^{k} s_i ≥ 2s/5ρ ;
6      s' = Σ_{i=1}^{k} s_i ;
7      if k < m then
8          if s − s' ≥ 2s/5ρ then
9              R_1, R_2 = Guillotine(R, s'/s) ;
10             return NRRP(R_1, {s_1, ..., s_k}) + NRRP(R_2, {s_{k+1}, ..., s_m})
11         else
12             R_1, R_2, R_3 = Tripartition(R, (s' − s_{m−1})/s, (s_{m−1})/s)) ;
13             return NRRP(R_1, {s_1, ..., s_{m−2}}) + Z_2 + Z_3
14     else
15         s' = Σ_{i=1}^{k−1} s_i ;
16         if s'/s ≤ 1 − 3(ρ+1)^2/16ρ then
17             R_1, Z_2 = Square(R, s'/s) ;
18             return NRRP(R_1, {s_1, ..., s_{m−1}}) + Z_2
19         else
20             if m > 2 then
21                 s'' = s' − s_{m−1} ;
22                 if s'' ≥ 2ρs'^2/5s then
23                     if s'' ≤ 5ρs'^2/2s then
24                         R_1, Z_2, Z_3 = Guillotine(R, s'/s, s''/s') ;
25                         return NRRP(R_1, {s_1, ..., s_{m−2}}) + Z_2 + Z_3
26                     else
27                         s''' = s'' − s_{m−2} ;
28                         if s''' ≥ 2ρs'^2/5s then
29                             R_temp, Z_2 = Guillotine(R, s'/s) ;
30                             L = Packing({s_1, ..., s_{m−1}}, 2ρs'^2/5s, 5ρs'^2/2s, R_temp) ;
31                             return Map(NRRP, L) + Z_2
32                         else
33                             if s'''/s ≤ (1−√(1−ρ∗s'/s))^2/ρ then
34                                 Z_2, R_temp, Z_4 = Guillotine(R, s'/s, (s''' + s_{m−1})/s') ;
35                                 R_1, Z_3 = Square(R_temp, s'''/(s''' + s_{m−1})) ;
36                                 return NRRP(R_1, {s_1, ..., s_{m−3}}) + Z_2 + Z_3 + Z_4
37                             else
38                                 R_1, R_temp, Z_4 = Superposition(R, s'/s, s'''/s) ;
39                                 Z_2, Z_3 = Guillotine(R_temp, s_{n−2}/(s' − s''')) ;
40                                 return NRRP(R_1, {s_1, ..., s_{m−3}}) + Z_2 + Z_3 + Z_4
41                 else
42                     if s''/s ≤ (1−√(1−ρ∗s'/s))^2/ρ then
43                         R_temp, Z_3 = Guillotine(R, s'/s) ;
44                         R_1, Z_2 = Square(R_temp, s''/s') ;
45                         return NRRP(R_1, {s_1, ..., s_{m−2}}) + Z_2 + Z_3
46                     else
47                         R_1, Z_2, Z_3 = Superposition(R, s'/s, s''/s) ;
48                         return NRRP(R_1, {s_1, ..., s_{m−2}}) + Z_2 + Z_3
49             else
50                 Z_1, Z_2 = Guillotine(R, s'/s) ;
51                 return Z_1 + Z_2 ;
```



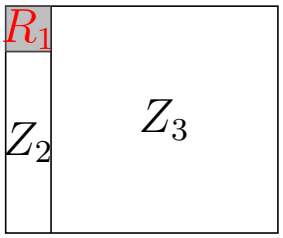
(a) **(A1)**, l9-l10



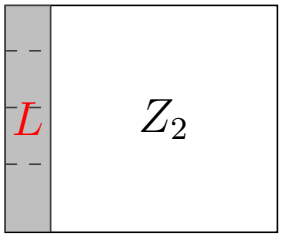
(b) **(A2)**, l12-l13



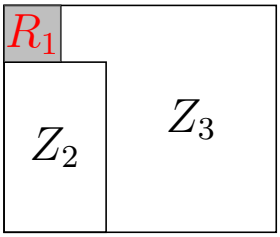
(c) **(B1)**, l17-l18



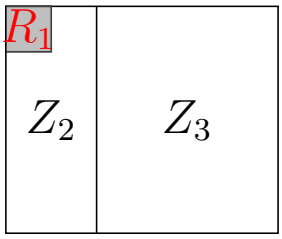
(d) **(B2-a1)**, l24-l25



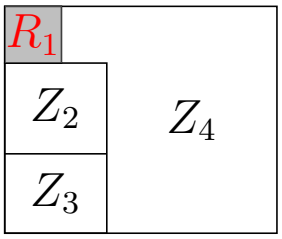
(e) **(B2-a2′)**, l29-l31



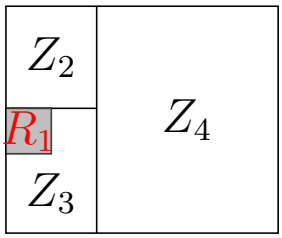
(f) **(B2-a2″-1)**, l34-l36



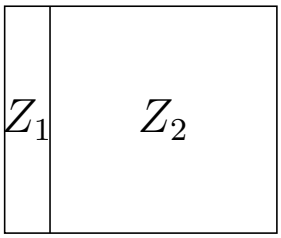
(g) **(B2-a2″-2)**, l38-l40



(h) **(B2-b1)**, l43-l45



(i) **(B2-bZ)**, l47-l48



(j) **(B2-c)**, l50-l51

Figure 1.23: All the cases in NRRP. The gray rectangles are the ones on which the recursive calls are made, the white ones are returned zones.

The proof that NRRP is indeed a $\frac{2}{\sqrt{3}}$-approximation algorithm is decomposed into two parts.

In order to keep the proofs relatively simple, we enforce that NRRP can only be applied to a (simple or composed) zone R that fulfills the following properties: R must be (i) a rectangle (ii) whose aspect ratio is less or equal than 5/2 (the μ we are aiming for). Proving that these two properties hold true in NRRP is done in the Correctness subsection.

Unfortunately, it is not always possible to partition the unit square into rectangles whose aspect ratios are all smaller than 5/2, and NRRP may create such zones under the following two conditions: (i) these zones have to be terminal (ii) each time NRRP creates such a set of zones, the $\frac{2}{\sqrt{3}}$-approximation ratio must be fulfilled for the set of zones (maybe not for each individual zone, but for the set as a whole). Proving that these two properties hold true in NRRP is done in the Approximation Ratio subsection.

In both cases (Correctness and Approximation Ratio subsections), proofs are rather technical and involve many subcases, but all these subcases are required to enforce the claimed approximation ratio.

Correctness

In order to prove the correctness of NRRP, we need to prove the following theorem, that states that all the composed rectangles (on which the algorithm is recursively applied) have an aspect ratio lower than 5/2, since this property is crucial in order to establish the approximation ratio. In all the following, we consider that the list of s_i values is sorted in increasing order, $s_1 \leq s_2 \leq \ldots \leq s_m$. Furthermore, we assume that rectangles R are composed zones, *i.e.* $m > 1$.

Theorem 1.15 (Correctness of NRRP)**.** *When executing $NRRP(R, \{s_1, \ldots, s_m\})$ with $\rho(R) \leq 5/2$, all the recursive calls to $NRRP(R', \{s'_1, \ldots, s'_k\})$ are performed on a rectangle area R' such that $\rho(R') \leq 5/2$.*

Proof. The first step of the algorithm finds k, the smallest index such that $s' = \sum_{i=1}^{k} s_i \geq \frac{2s}{5\rho}$, where $s = s(R)$ and $\rho = \rho(R)$ (Line 5). Depending on the value of k, there are two cases (Line 7).

Case (A) (lines 8-13) corresponds to the case $k < m$, that is split into two subcases.

Case (A1) (lines 9-10) corresponds to the case where $s - s'$ is also larger than $\frac{2s}{5\rho(R)}$. Then, with $\alpha = s'/s$, Lemma 1.16 applies.

Lemma 1.16. *Let R be a rectangle with $\rho(R) \leq 5/2$, $\alpha \in [0, 1]$ and $R_1, R_2 = Guillotine(R, \alpha)$.*

- *If $\alpha \geq \frac{2}{5\rho(R)}$ then $\rho(R_1) \leq 5/2$.*

- *If* $(1-\alpha) \geq \frac{2}{5\rho(R)}$ *then* $\rho(R_2) \leq 5/2$.

Proof. Let us assume without loss of generality that $h = h(R) \leq w(R) = w$, and denote $\rho = \rho(R) = \frac{w}{h}$. Then $\rho(R_1) = \min(\frac{\alpha w}{h}, \frac{h}{\alpha w})$. We have $\frac{\alpha w}{h} \leq \frac{w}{h} = \rho \leq 5/2$ and $\frac{h}{\alpha w} = \frac{1}{\alpha\rho} \leq \frac{5\rho}{2\rho} \leq 5/2$ (under the assumption $\alpha \geq \frac{2}{5\rho}$). Therefore, $\alpha \geq \frac{2}{5\rho}$ implies $\rho(R_1) \leq 5/2$ and for the same reason, $(1-\alpha) \geq \frac{2}{5\rho}$ implies $\rho(R_2) \leq 5/2$. □

Therefore, in case (A1), $Guillotine(R, \alpha)$ returns two rectangles R_1, R_2 whose aspect ratios are smaller than $5/2$ and we can apply NRRP on each of them.

Case (A2) (lines 12-13) corresponds to the case where $s' = \sum_{i=1}^{k} s_i \geq \frac{2s}{5\rho}$, $s - s' < \frac{2s}{5\rho}$ and $k < m$. In this case, we rely on the `Tripartition` routine and the following lemma states that if $R_1, Z_2, Z_3 = Tripartition(R, (s' - s_{m-1})/s, s_{m-1}/s)$, then $\rho(R_1) \leq 5/2$ and Z_2 and Z_3 are simple.

Lemma 1.17. *If* $s' = \sum_{i=1}^{k} s_i \geq \frac{2s}{5\rho(R)}$, $k < m$ *and* $s - s' < \frac{2s}{5\rho(R)}$, *then* $k = m-1$ *and* $\rho(R) < 6/5$. *In addition, if* $R_1, Z_2, Z_3 = Tripartition(R, (s' - s_{n-1})/s, s_{n-1}/s)$, *then* $\rho(R_1) \leq 5/2$.

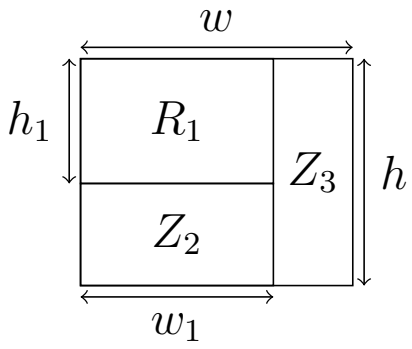


Figure 1.24: **Case (A2)**

Proof. Let us assume without loss of generality that $h = h(R) \leq w(R) = w$ and define $\rho = \rho(R)$. By definition of k, we know that $s'' = \sum_{i=1}^{k-1} s_i < \frac{2s}{5\rho}$. Therefore

$$
\begin{aligned}
s_k &= s' - s'', \\
s_k &= s - (s - s') - s'', \\
s_k &> s - \frac{2s}{5\rho} - \frac{2s}{5\rho}, \\
s_k &> s(1 - \frac{4}{5\rho}) = s\frac{4\rho - 4}{5\rho}.
\end{aligned}
$$

Since the s_i values are sorted, $s - s' = \sum_{i=k+1}^{m} s_i \geq (m-k)s_{k+1} \geq (m-k)s_k$. Hence $\frac{2s}{5\rho} > (m-k)s_k > (m-k)s\frac{5\rho-4}{5\rho}$. This implies that $m - k < \frac{2}{5\rho-4} \leq 2$ since $\rho \geq 1$, and then $k \geq m-1$. Thus, the only possible value for k is $m-1$ (remember that $k < m$). Therefore, $s = s'' + s_{m-1} + s_m < 3\frac{2s}{5\rho}$ and $\rho < 6/5$.

Let us denote $\alpha = s''/s$, $\beta = s_{m-1}/s$ and $\gamma = s_m/s$, $w_1 = w(R_1)$ and $h_1 = h(R_1)$ (see Figure 1.24). We want to prove that both w_1/h_1 and h_1/w_1 are smaller than $5/2$, which is equivalent to proving $2/5 \leq w_1/h_1 \leq 5/2$. First, let us recall that $w_1 = (\alpha+\beta)w$ and $h_1 = \frac{\alpha}{\alpha+\beta}h$.

Let us now establish lower bounds on α, β and γ (the following bounds hold true $\alpha < \frac{2}{5\rho}$ and $\beta \leq \gamma < \frac{2}{5\rho}$). Let us notice that $1 - \alpha = \beta + \gamma < \frac{4}{5\rho}$. Therefore $\alpha > \frac{5\rho-4}{5\rho}$, and similarly $\beta > \frac{5\rho-4}{5\rho}$. For γ, noticing that $2\gamma \geq \beta + \gamma = 1 - \alpha > \frac{5\rho-2}{5\rho}$, we obtain $\gamma > \frac{5\rho-2}{10\rho}$.

Since $\alpha + \beta = 1 - \gamma$, then

$$\frac{5\rho-2}{5\rho} < \alpha + \beta < \frac{5\rho+2}{10\rho}.$$

Moreover, since $w_1/h_1 = \rho\frac{(\alpha+\beta)^2}{\alpha}$, then

$$\rho\frac{(5\rho-2)^2}{25\rho^2} \times \frac{5\rho}{2} < \frac{w_1}{h_1} < \rho\frac{(5\rho+2)^2}{100\rho^2} \times \frac{5\rho}{5\rho-4}$$

and

$$\frac{(5\rho-2)^2}{10} < \frac{w_1}{h_1} < \frac{(5\rho+2)^2}{20(5\rho-4)}.$$

Trivially, $\frac{(5\rho-2)^2}{10} \geq 9/10 > 2/5$. Moreover, $\frac{(5\rho+2)^2}{20(5\rho-4)} \leq 49/20 < 5/2$ since $x \mapsto \frac{(5x+2)^2}{20(5x-4)}$ is a decreasing function over $[1, 6/5]$. Hence, we prove $2/5 \leq w_1/h_1 \leq 5/2$ and therefore, $\rho(R_1) \leq 5/2$. □

The above lemmas prove the correctness of the calls to NRRP in cases (A1) and (A2) (lines (10) and (13) of Algorithm 1.9). Case (B) (lines 15-51) corresponds to the case where $k = m$, which happens when s_m is significantly larger than the other values. Let us denote $s' = s - s_m$ (Line 15). Depending on the value of s', several subcases can occur.

Case (B1) (lines 17 and 18) corresponds to a small s', *i.e.* $s' \leq s(1 - \frac{3(\rho+1)^2}{16\rho})$. In this case, `Square` is called and generates R_1 that is a square (with aspect ratio is $1 < 5/2$) and the simple zone Z_2.

Let us note that in all remaining cases, $\rho - \frac{3(\rho+1)^2}{16} < \frac{\rho s'}{s} < 2/5$ (where $\rho = \rho(R)$). In addition, $13/64 \leq \rho - \frac{3(\rho+1)^2}{16}$ for $\rho \in [0, 5/2]$ and thus,

$$13/64 \leq \rho - \frac{3(\rho+1)^2}{16} < \rho\frac{s'}{s} < 2/5. \tag{1.2}$$

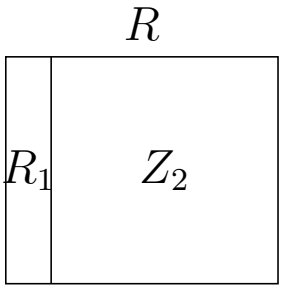


Figure 1.25: **Case (B2)**

Case (B2) (lines 21 to 48) The situation is depicted in Figure 1.25, where Z_2 is simple (and such that $\rho(Z_2) \leq 5/2$). Unfortunately, $\rho(R_1) > 5/2$ so that NRRP cannot directly be called on R_1, which needs to be further split into several rectangles with acceptable aspect ratio.

Let us denote $s'' = s' - s_{m-1}$.

Case (B2-a) (Lines 22-40) corresponds to the case $\frac{2\rho s'^2}{5s} \leq s''$.

Case (B2-a1) (Lines 24-25) correspond to the case $\frac{2\rho s'^2}{5s} \leq s'' \leq \rho\frac{5s'^2}{2s}$. Lemma 1.19 proves that if $R_1, Z_2, Z_3 = \mathit{Guillotine}(R_1, s'/s, s''/s')$ (see Figure 1.26), then $\rho(R_1) \leq 5/2$ and that Z_2 and Z_3 are simple so that the call $NRRP$ on R_1 at line 24 is valid.

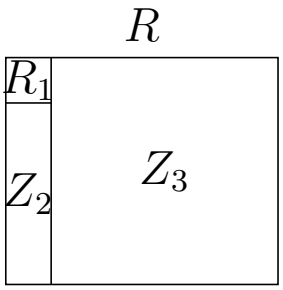


Figure 1.26: **Case (B2-a1)**

Case (B2-a2) (lines 27-40) corresponds to the case where $s'' > \rho\frac{5s'^2}{2s}$. It is again split into several subcases depending on the value of $s''' = s'' - s_{m-2}$. Let us first note that $s''' > 0$. Indeed, thanks to (1.2), we know that $s''/s' > \rho\frac{5s'}{2s} > 65/128 > 1/2$. Therefore $s_{m-2} \leq s_{m-1} < s'(1-1/2) < s'/2 < s''$.

Case (B2-a2′) (Lines 29-31) corresponds to the case $s''' \geq \rho\frac{2s'^2}{5s}$. Then, the conditions of Lemma 1.18 hold true (with $\alpha = \rho\frac{s'}{s}$ and the s of Lemma 1.18 be the current s') and we can build $\mathit{Packing}(\{s_1, \ldots, s_{m-1}\}, \frac{2\rho s'^2}{5s}, \frac{5\rho s'^2}{2s}, R_1)$. By definition of `Packing`, each rectangle of this list (except possibly at one simple rectangle) have a surface between $\frac{2\rho s'^2}{5s}$ and $\frac{5\rho s'^2}{2s}$. Therefore, with x being the surface of the different rectangles from $\mathit{Packing}(\{s_1, \ldots, s_{m-1}\}, \frac{2\rho s'^2}{5s}, \frac{5\rho s'^2}{2s}, R_1)$,

we are in the condition of Lemma 1.19 (the rectangle R in Lemma 1.19 is R_1 here). Thus we can call NRRP on each element of the list, since all the rectangles of the list, except, again, possibly at one simple rectangle, have an aspect ratio smaller than 5/2 (see Figure 1.27).

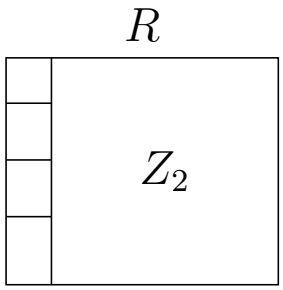


Figure 1.27: **Case (B2-a2′)**

Lemma 1.18. *Let R be a rectangle such that $\rho(R) = 1/\alpha$ and $\{s_1, \ldots, s_m\}$ an ordered list such that $\sum s_i = s(R) = s$. Then, if $m \geq 3$, $\sum_{i=1}^{m-2} s_i \geq \frac{2\alpha s}{5}$, $\sum_{i=1}^{m-1} s_i > \frac{5\alpha s}{2}$ and $13/64 < \alpha < 2/5$, then we can build $Packing(R, \{s_1, \ldots, s_m\}, \frac{2\alpha s}{5}, \frac{5\alpha s}{2})$ in linear time.*

Proof. Let us denote $s = s(R)$, $s' = s - s_m$ and $s'' = s' - s_{m-1}$ (we know that $s'' \geq \frac{2\alpha s}{5}$). Note that $\frac{5\alpha}{2} > 1/2$.

- Let us first assume that $s_m + s_{m-1} > \frac{5\alpha s}{2}$. In this case, $2s_m \geq s_m + s_{m-1} > \frac{5\alpha s}{2}$ and therefore $s_m > \frac{5\alpha s}{4} > \frac{2\alpha s}{5}$. In addition, $s'' \leq \frac{5\alpha s}{2}$ (otherwise $s = s'' + s_{m-1} + s_m > s/2 + s/2$). Therefore, let us consider R_1, where $R_1, R_2 = Guillotine(R, s'/s)$. Let j be such that $\sum_{i=1}^{j} s_i \leq s' - \frac{2\alpha s}{5}$ and $\sum_{i=1}^{j+1} s_i > s' - \frac{2\alpha s}{5}$. We have two cases:

  (i) If $s_{m-1} \geq \frac{2\alpha s}{5}$, then $j = m - 2$. Therefore, $\frac{2\alpha s}{5} \leq s'' \leq \frac{5\alpha s}{2}$ and $\frac{2\alpha s}{5} \leq s_{m-1} \leq \frac{5\alpha s}{2}$ ($s_{m-1} > \frac{5\alpha s}{2}$ implies $s_m > \frac{5\alpha s}{2}$ and for the same reason, since $s'' \leq \frac{5\alpha s}{2}$, this is impossible). Therefore, if $R_1, R_3 = Guillotine(R_1, s''/s')$, $[(R_1, \{s_1, \ldots, s_{m-2}\}), (R_3, \{s_{m-1}\}), (R_2, \{s_m\})]$ is a valid output of $Packing(R, \{s_1, \ldots, s_m\}, \frac{2\alpha s}{5}, \frac{5\alpha s}{2})$.

  (ii) If $s_{m-1} < \frac{2\alpha s}{5}$, then $\forall i \leq m-1$, $s_i < \frac{2\alpha s}{5}$ and $j < m - 1$. Let us denote $s_{bis} = \sum_{i=1}^{j} s_i$ and $s_{ter} = \sum_{i=j+1}^{m-1} s_i$. $s_{bis} < s'' \leq \frac{5\alpha s}{2}$ and

  $$s_{bis} + s_{j+1} > s' - \frac{2\alpha s}{5} > \frac{5\alpha s}{2} - \frac{2\alpha s}{5}.$$

  Hence

  $$s_{bis} > \frac{5\alpha s}{2} - \frac{2\alpha s}{5} - \frac{2\alpha s}{5} = (5/2 - 4/5)\alpha s \geq \frac{2\alpha s}{5}.$$

Moreover, $s' - s_{bis} = s_{ter}$ implies that $s_{ter} \geq s' - (s' - \frac{2\alpha s}{5})$ and therefore $s_{ter} \geq \frac{2\alpha s}{5}$, and $s_{ter} \leq s' - (s' - \frac{4\alpha s}{5}) < \frac{5\alpha s}{2}$. Therefore, if $R_1, R_3 = Guillotine(R_1, (s_{bis} + s_{ter})/s', s_{ter}/(s_{bis} + s_{ter}))$,

$[(R_1, \{s_1, \ldots, s_j\}), (R_3, \{s_{j+1}, \ldots, s_{m-1}\}), (R_2, \{s_m\})]$ is a valid output of $Packing(R, \{s_1, \ldots, s_m\}, \frac{2\alpha}{5}, \frac{5\alpha}{2})$.

- Otherwise, $s_m + s_{m-1} \leq \frac{5\alpha s}{2}$. Since $s_{m-1} \leq s_m$, we know that $s_{m-1} \leq \frac{5\alpha s}{4}$ and therefore $\forall i \leq m-1$, $s_i \leq \frac{5\alpha s}{4}$. Let us apply the following procedure:
  - If $s_i \geq \frac{2\alpha s}{5}$, then, since $s_i \leq s_m \leq \frac{5\alpha s}{2}$, we can leave s_i alone since it fulfills the condition.
  - Else, let j_i be such that $\sum_{j=j_i}^{i} s_j \geq \frac{2\alpha s}{5}$ and $\sum_{j=j_i+1}^{i} s_j < \frac{2\alpha s}{5}$. Note that

$$\sum_{j=j_i}^{i} s_j = s_{j_i} + \sum_{j=j_i+1}^{i} s_j < s_i + \sum_{j=j_i+1}^{i} s_j < \frac{4\alpha s}{5} < \frac{5\alpha s}{2}$$

    Then, the set $\{s_{j_i}, \ldots, s_i\}$ fulfills the condition.

  After the execution of the above algorithm, there may exist a k such that $\sum_{j=1}^{k} s_j < \frac{2\alpha s}{5}$. If $s_{k+1} \geq \frac{2\alpha s}{5}$, then

$$\sum_{j=1}^{k+1} s_j \leq \frac{2\alpha s}{5} + s_{m-1} \leq \frac{2\alpha s}{5} + \frac{5\alpha s}{4} = \frac{33\alpha s}{20} < \frac{5\alpha s}{2}$$

  and trivially $\sum_{j=1}^{k+1} s_j \geq \frac{2\alpha s}{5}$. Therefore the set $\{s_1, \ldots, s_{k+1}\}$ fulfills the condition. Otherwise, there exists i such that $k = j_i$. Then

$$\sum_{j=1}^{i} s_j \leq \frac{2\alpha s}{5} + \frac{4\alpha s}{5} = \frac{6\alpha s}{5} < \frac{5\alpha s}{2}$$

  and trivially $\sum_{j=1}^{i} s_j \geq \frac{2\alpha s}{5}$. Therefore, the set $\{s_1, \ldots, s_i\}$ fulfills the condition.

Then, in any possible case, we have built a valid result for $Packing(R, \{s_1, \ldots, s_m\}, \frac{2\alpha s}{5}, \frac{5\alpha s}{2})$ (in linear time). □

Lemma 1.19. *Let R be a rectangle such that $\rho(R) = 1/\alpha$ and $R', R'' = Guillotine(R, x)$. If $13/64 \leq \alpha \leq 2/5$ and $\frac{2\alpha}{5} \leq x \leq \frac{5\alpha}{2}$, then $\rho(R') \leq 5/2$.*

Proof. Let us suppose without loss of generality that $h = h(R) \leq w(R) = w$. In this case, $h(R') = h$ and $w(R') = xw$ with $x \in [0, 1]$. $s(R') = xw \times h = xS(R)$. Therefore $\frac{2\alpha}{5} \leq x \leq \frac{5\alpha}{2}$. Since $\rho(R') \leq 5/2 \Leftrightarrow 2/5 \leq w(R')/h(R') \leq 5/2$ is equivalent to $2/5 \leq x/\alpha \leq 5/2$ and $\frac{2\alpha}{5} \leq x \leq \frac{5\alpha}{2}$, which is true by construction. □

Case (B2-a2″) (Lines 33-40) corresponds to the case $s''' < \frac{2\rho s'^2}{5s}$. Note that $s''' < \frac{2\rho s'^2}{5s}$ and $s'' > \frac{5\rho s'^2}{2s}$ is a possible situation, for example with $s_{m-1} = s_{m-2} = \frac{21\rho s'^2}{10s} + \epsilon$. In this case, either we successively apply `Guillotine` and `Square` on R or we use `Superposition` on R (the choice will be discussed in the Approximation Ratio subsection) before using `Guillotine` on one of the rectangle. Both cases are depicted in Figure 1.28 and in all cases, at most one rectangle (R_1) is not simple, and since it is shaped as a square, then its aspect ratio is less than 5/2, and the calls to NRRP at lines 37 and 39 are both valid.

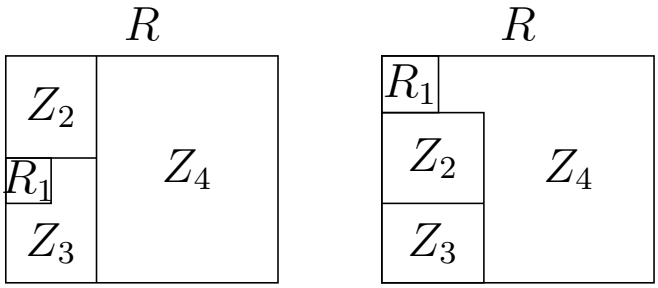


Figure 1.28: **Case (B2-a2″)**

Case (B2-b) (lines 42 to 48) corresponds to the case $s'' < \frac{2\rho s'^2}{5s}$. In this case, we rely on the technique described for Case (B2-a2″). We successively apply `Guillotine` and `Square` on R or we use `Superposition` on R (the choice will be discussed in the Approximation Ratio subsection) and are depicted in Figure 1.29.

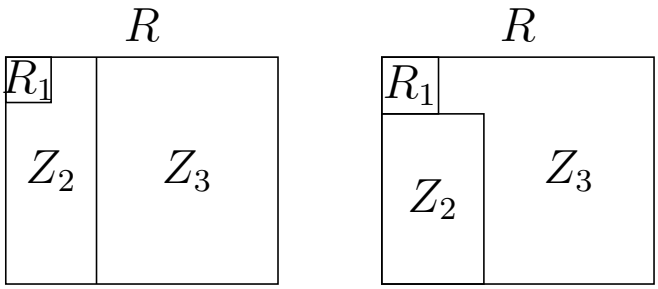


Figure 1.29: **Case (B2-b)**

Case (B2-c) (lines 50 and 51) corresponds to the case where $m = 2$, and thus neither subcases in (B2-a) nor (B2-b) are possible, and s_1 is too big to use case (B1). In this case NRRP simply uses `Guillotine` and returns the two resulting rectangles. Therefore there are only simple zones in this case.

This ends the proof of Theorem 1.15

□

To finish the correctness aspect of NRRP, we also prove that it is always possible to perform the `Square` routine, *i.e.* that the edge length of the produced square is indeed smaller than the lengths of the edges of the rectangle that contains it. We have two main cases that we recall on Figure 1.30. The left figure represents **case (B1)**, the right figure represents **case (B2-a2″)** and **case (B2-b)**.

For the first case, we know that $s(R_1) \leq \frac{2}{\rho(R)5} s(R)$. In addition $s(R) = \sqrt{\rho(\min(h(R), w(R)))^2}$. Thus,

$$h(R_1) = w(R_1) = \sqrt{s(R_1)} \leq \sqrt{\frac{2}{5}} \min(h(R), w(R))$$

and therefore $h(R_1) = w(R_1) \leq \min(h(R), w(R))$.

In the second case, we know that $s(R_1 \cup Z_2) \leq \frac{2}{\rho(R)5} s(R)$ and $s(R_1) \leq \frac{2\rho s(R_1 \cup Z_2)^2}{5s(R)}$ (in both cases, **(B2-a2″)** and **(B2-b)**, $s(R_1 \cup Z_2)s'$, $s(R_1) = s'''$ in case **(B2-a2″)** and $s(R_1) = s''$ in case **(B2-b)**). Let us suppose without loss of generality that $w = w(R) \geq h(R) = h$ and let us denote $\alpha = \frac{s(R_1 \cup Z_2)}{s(R)}$. Therefore $w(Z_2) = \alpha w$ and $h(Z_2) = h$ and

$$s(R_1 \cup Z_2) = w(Z_2)h(Z_2) = \alpha\rho h^2.$$

In addition $w(R_1) = h(R_1) = \sqrt{s(R_1)}$. Thus,

$$\begin{aligned} w(R_1) \leq \sqrt{\frac{2\rho s(R_1 \cup Z_2)^2}{5s(R)}} \leq \alpha\sqrt{\frac{2\rho wh}{5}} &= \alpha\rho \left(\sqrt{\frac{2}{5}}\right) h \\ &= \alpha \left(\sqrt{\frac{2}{5}}\right) w. \end{aligned}$$

Therefore $w(R_1) \leq \left(\sqrt{\frac{2}{5}}\right) w(Z_2) \leq w(Z_2)$ and $w(R_1) \leq \alpha\rho \left(\sqrt{\frac{2}{5}}\right) h(Z_2)$. Since $\alpha\rho \leq \frac{2}{5}$, $w(R_1) \leq h(Z_2)$ also holds true what finishes the correctness proof of NRRP.

Approximation Ratio

We now prove our claim that NRRP is a $\frac{2}{\sqrt{3}}$-approximation for PERI-SUM (Theorem 1.20).

Theorem 1.20. *NRRP is a $\frac{2}{\sqrt{3}}$-approximation for PERI-SUM.*

Proof. We will extensively rely on the following lemma in order to prove Theorem 1.20.

Lemma 1.21. *Let A, B, C and D denote 4 positive rational numbers, then if $\frac{A}{B} \leq \alpha$ and $\frac{C}{D} \leq \alpha$, then $\frac{A+C}{B+D} \leq \alpha$.*

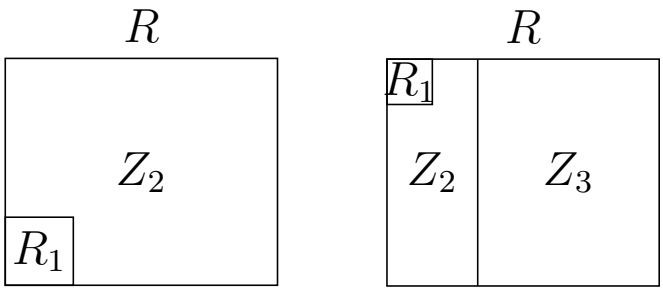


Figure 1.30: Possible uses of `Square` routine.

Proof. Let A, B, C and D be 4 positive rational numbers such that $\frac{A}{B} \leq \alpha$ and $\frac{C}{D} \leq \alpha$. Then $A \leq \alpha B$ and $C \leq \alpha D$. Thus $A + C \leq \alpha(B + D)$ and finally $\frac{A+C}{B+D} \leq \alpha$. □

Lemma 1.21 plays a crucial role in the following proof. Indeed let us partition the s_ks into I subsets $S_1, \ldots, S_I$. Then, by Lemma 1.21, if

$$\forall i, \frac{\sum_{k \in S_i} p(Z_k)}{2 \sum_{k \in S_i} \sqrt{s_k}} \leq \frac{2}{\sqrt{3}}, \text{ then } \frac{\sum_{k=1}^{m} p(Z_k)}{2 \sum_{k=1}^{m} \sqrt{s_k}} \leq \frac{2}{\sqrt{3}}.$$

If NRRP is called with $m = 1$, then it returns the input rectangle (line 2). We have proved in the Correctness subsection that all composed rectangles (on which NRRP is recursively called) have an aspect ratio less than 5/2. This is enough to prove our result since a such a rectangle satisfies, by Lemma 1.3

$$\frac{p(A_k)}{2\sqrt{s_k}} \leq \frac{7\sqrt{2}}{4\sqrt{5}} < \frac{2}{\sqrt{3}}.$$

To establish the approximation ratio, using Lemma 1.21, we can therefore consider those rectangles independently.

Unfortunately, there are simple Z zones for which the 5/2 aspect ratio does not hold true, and it is not true in all cases that $\frac{p(Z_k)}{2\sqrt{s(Z_k)}} \leq \frac{2}{\sqrt{3}}$. Nevertheless, for each case depicted in Figures 1.24, 1.25, 1.26, 1.27 ,1.28, 1.29, if we group all the terminal simple zones Z_1, Z_2 (and possibly Z_3), then we can prove that $\frac{\sum p(Z_k)}{2 \sum \sqrt{s(Z_k)}} \leq \frac{2}{\sqrt{3}}$, so that the bound holds globally if it is not the case for individual zones. Then, we can conclude with Lemma 1.21 that the $\frac{2}{\sqrt{3}}$ bound holds true since it is enough to exhibit one partitioning of the s_ks such that the bounds holds for each individual group of the partition.

The rest of the proof is rather technical and simply proves that the above bound holds true for all possible subcases of NRRP.

Let us now check the different possible results of NRRP.

First if $m = 1$, then a single rectangle R is returned, and by construction (see Correctness subsection) (Line 2) $\rho(R) \leq 5/2$, what ends the proof.

Case (A1) (lines 9-10): NRRP returns no simple areas.

Case (A2) (lines 12-13) corresponds to the case described in Figure 1.31 with 2 simple zones Z_2 and Z_3. Lemma 1.23 proves that $p(Z_2) + p(Z_3) \leq \frac{2}{\sqrt{3}}(2(\sqrt{s_{m-1}} + \sqrt{s_m}))$, what ends the proof of this case. Lemma 1.22 is a technical Lemma that is needed to prove Lemma 1.23.

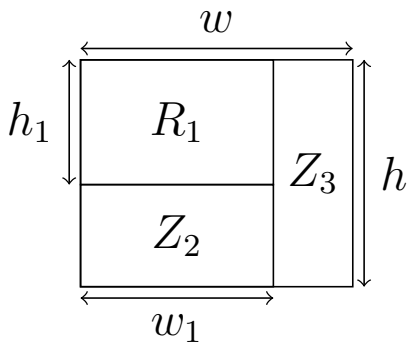


Figure 1.31: **Case (A2)**

Lemma 1.22. *Let* $x \in [1, 6/5]$, $z \in [\frac{5x-2}{10x}, \frac{2}{5x}]$ *and* $y \in [\frac{5x-4}{5x}, z]$ *and* $f(x, y, z) = \frac{x+1+\frac{y}{1-z}}{2\sqrt{x}(\sqrt{y}+\sqrt{z})}$. *If* $\alpha = 1 - z - y \geq 1/5$ *and* $xz \leq 2/5$, *then* $f(x, y, z) \leq f(1, 1/5, 3/10) = \frac{16\sqrt{5}}{7(2+\sqrt{6})} < \frac{2}{\sqrt{3}}$

Proof. First let us prove that f is decreasing in y and z.

$$\frac{\partial f}{\partial y}(x, y, z) = \frac{2\sqrt{y}(\sqrt{y} + \sqrt{z}) - (1 - z)(x + 1) - y}{4\sqrt{xy}(1 - z)(\sqrt{(}y) + \sqrt{z})^2}$$

Therefore, $\frac{\partial f}{\partial y}(x, y, z) \leq 0$ is equivalent to $2\sqrt{y}(\sqrt{y}+\sqrt{z})-(1-z)(x+1)-y \leq 0$.

$$\begin{aligned} 2\sqrt{y}(\sqrt{y} + \sqrt{z}) - (1 - z)(x + 1) - y &= 2y + 2\sqrt{yz} - x - y - 1 + xz + z \\ &= y + z + 2\sqrt{yz} + zx - x - 1 \\ &= 2\sqrt{yz} - (1 - \gamma)x - \alpha \end{aligned}$$

As $x \geq 1$ and $y \leq z$, we have:

$$\begin{aligned} 2\sqrt{y}(\sqrt{y} + \sqrt{z}) - (1 - z)(x + 1) - y &\leq 2z - +z - \alpha \\ &\leq 3z - 1 - \alpha \end{aligned}$$

Yet $z \leq \frac{2}{5\rho} \leq 6/5$ and $\alpha \geq 1/5$, hence,

$$2\sqrt{y}(\sqrt{y} + \sqrt{z}) - (1 - z)(x + 1) - y \leq 6/5 - 1 - 1/5 = 0$$

and thus $\frac{\partial f}{\partial y}(x,y,z) \leq 0$.

$$\frac{\partial f}{\partial z}(x,y,z) = \frac{2z^{3/2}(\sqrt{y}+\sqrt{z}) - (1-z)^2(x+1) - (1-z)y}{2\sqrt{xz}(1-z)^2(\sqrt{(}y)+\sqrt{z})^2}$$

So $\frac{\partial f}{\partial z}(x,y,z) \leq 0$ is equivalent to $2z^{3/2}(\sqrt{y}+\sqrt{z})-(1-z)^2(x+1)-(1-z)y \leq 0$.

$$\begin{aligned} 2z^{3/2}(\sqrt{y}+\sqrt{z}) - (1-z)^2(x+1) - (1-z)y &= 2z^2 + 2z\sqrt{yz} - x - 1 + 2zx \\ &\quad + 2z - xz^2 - z^2 - y + xy \\ &\leq 4z^2 - x - 1 + 2xz + 2z - xz^2 - z^2 - y - z^2 \\ &\leq 4z^2 - x - 1 + 2xz + 2z - xz^2 - y \\ &\leq 3z^2 - 2 + 2xz + 2z - y \end{aligned}$$

Since $xz \leq 2/5$,

$$\begin{aligned} 2z^{3/2}(\sqrt{y}+\sqrt{z}) - (1-z)^2(x+1) - (1-z)y &\leq 3z^2 - 2 + 4/5 + 2z - y \\ &\leq 12/25 + 4/5 - 1/5 - 6/5 \\ &\leq -3/25 \end{aligned}$$

and thus $\frac{\partial f}{\partial z}(x,y,z) \leq 0$.

Then $f(x,y,z) \leq f(x, \frac{5x-4}{5x}, \frac{5x-2}{10x})$. Let us denote $g(x) = f(x, \frac{5x-4}{5x}, \frac{5x-2}{10x})$ so that

$$g(x) = \frac{\sqrt{5}(x(5x+17)-6)}{(5x+2)(2\sqrt{5x-4}+\sqrt{10x-4})}$$

We are interested in proving that $g(x)$ is decreasing with x in $[1, 6/5]$.

First,

$$\begin{aligned} g'(x) = &\frac{\sqrt{5}(10x+17)}{(5x+2)(2\sqrt{5x-4}+\sqrt{10x-4})} \\ &- \frac{5\sqrt{5}(x(5x+17)-6)}{(5x+2)^2(2\sqrt{5x-4}+\sqrt{10x-4})} \\ &- \frac{\sqrt{5}(x(5x+17)-6)(\frac{5}{\sqrt{10x-4}}+\frac{5}{\sqrt{5x-4}})}{(5x+2)(2\sqrt{5x-4}+\sqrt{10x-4})^2} \end{aligned}$$

so that $g'(x) \leq 0 \iff$

$$(25x^2+20x+64) - \frac{5(x(5x+17)-6)(5x+2)(\frac{1}{\sqrt{10x-4}}+\frac{1}{\sqrt{5x-4}}))}{(2\sqrt{5x-4}+\sqrt{10x-4})} \leq 0$$

Moreover $5x+2$ is increasing with x in the interval $[1, 6/5]$, so that $5x+2 \geq 7$. To finish the proof, let us denote $A(x) = \frac{(\frac{1}{\sqrt{10x-4}}+\frac{1}{\sqrt{5x-4}})}{(2\sqrt{5x-4}+\sqrt{10x-4})}$ Clearly, A(x) is

decreasing with x in the interval $[1, 6/5]$, so that in particular in the interval $[1, 11/10]$, $A(x) \geq A(11/10)$ and in the interval $[11/10, 6/5]$, $A(x) \geq A(6/5)$.

Therefore, in the interval $[1, 11/10]$,

$$B(x) = (25x^2+20x+64) - \frac{(5(x(5x+17)-6)(5+2)(\frac{1}{\sqrt{10*11/10-4}} + \frac{1}{\sqrt{5*11/10-4}}))}{(2\sqrt{5*11/10-4} + \sqrt{10*11/10-4}))} \leq 0$$

$$\Longrightarrow g'(x) \leq 0.$$

Moreover, B is a polynomial of degree 2 that tends to $-\infty$ when x tends to $+\infty$, that is equal to $40\sqrt{42} - 146 > 0$ in 0 and to $669 - 320\sqrt{14/3} < 0$ in 1 so that $B(x)$ is negative in the interval $[1, 11/10]$

Similarly, in the interval $[11/10, 6/5]$,

$$C(x) = (25x^2+20x+64) - \frac{(5(x(5x+17)-6)(5+2)(\frac{1}{\sqrt{10*6/5-4}} + \frac{1}{\sqrt{5*6/5-4}}))}{(2\sqrt{5*6/5-4} + \sqrt{10*6/5-4})} \leq 0$$

$$\Longrightarrow g'(x) \leq 0.$$

As previously, C is a polynomial of degree 2 that tends to $-\infty$ when x tends to ∞, that is equal to $827/8 > 0$ in 0 and to $-435/64 < 0$ in $11/10$ so that $B(x)$ is negative in the interval $[11/10, 6/5]$.

Therefore $f(x, y, z) \leq g(x) \leq g(1) = \frac{16\sqrt{5}}{7(2+\sqrt{6})} < \frac{2}{\sqrt{3}}$ and we have our result. □

Lemma 1.23. *Let us suppose that* $s' = \sum_{i=1}^{k} s_i \geq \frac{2s}{5\rho(R)}$, $k = m - 1$ *and* $s - s' < \frac{2s}{5\rho(R)}$. *Then, if* $R_1, Z_2, Z_3 = Tripartition(R, (s' - s_{n-1})/s, s_{m-1}/s)$, $\frac{p(Z_2)+p(Z_3)}{2(\sqrt{s(Z_2)}+\sqrt{s(Z_3)})} \leq \frac{2}{\sqrt{3}}$

Proof. We use the same notations as for Lemma 1.17: without loss of generality, $w = w(R) \geq h(R) = h$ and denote $\rho = \rho(R)$, $\alpha = s''/s$, $\beta = s_{n-1}/s$ and $\gamma = s_n/s$. Let us also denote $w_1 = w(R_1)$ and $h_1 = h(R_1)$ (see Figure 1.31). Remember that we have proved in Lemma 1.17 that

$$\begin{aligned} \frac{5\rho - 4}{5\rho} \leq \alpha \leq \frac{2}{5\rho} \\ \frac{5\rho - 4}{5\rho} \leq \beta \leq \gamma \\ \frac{5\rho - 2}{10\rho} \leq \gamma \leq \frac{2}{5\rho} \\ 1 \leq \rho \leq 6/5 \end{aligned}$$

and that $w_1 = (\alpha + \beta)w$ and $h_1 = \frac{\alpha}{\alpha+\beta}h$. With these notations, $p(Z_2) = w_1 + h - h_1$ and $p(Z_3) = w - w_1 + h$. Then $p(Z_2) + p(Z_3) = w + 2h - h_1 =$

$\rho h + 2h - \frac{\alpha}{\alpha+\beta}h = h(\rho + 2 - \frac{\alpha}{\alpha+\beta})$. Since $\alpha+\beta = 1-\gamma$, $\frac{\alpha}{\alpha+\beta} = \frac{1-\gamma-\beta}{1-\gamma} = 1 - \frac{\beta}{1-\gamma}$. Then, $p(Z_2) + p(Z_3) = h(\rho + 1 + \frac{\beta}{1-\gamma})$. Moreover, $s(Z_2) = wh\beta = \beta\rho h^2$ and $s(Z_3) = wh\gamma = \gamma\rho h^2$. Thus, with f defined as in Lemma 1.22,

$$\frac{p(Z_2) + p(Z_3)}{2(\sqrt{s(Z_2)} + \sqrt{s(Z_3)})} = \frac{\rho + 1 + \frac{\beta}{1-\gamma}}{2\sqrt{\rho}(\sqrt{\beta} + \sqrt{\gamma})} = f(\rho, \beta, \gamma)$$

Since $1 - \beta - \gamma = \alpha > 1/5$ and $\gamma\rho = \rho\frac{s-s'}{s} \leq 2/5$, we can apply Lemma 1.22 and obtain our result. □

Let us now move to Case (B), *i.e.* $k = m$.

Case (B1) (lines 17 and 18): NRRP returns a single simple zone Z_2 defined by $R_1, Z_2 = Square(R, s'/s)$. In this case, Lemma 1.24 proves that $p(Z_2) \leq \frac{2}{\sqrt{3}} \times 2\sqrt{s(Z_2)}$.

Lemma 1.24. *Let R be a rectangle. If $\alpha \leq 1 - \frac{3(\rho(R)+1)^2}{16\rho(R)}$ and $R_1, Z_2 = Square(R, \alpha)$, then $\frac{p(Z_2)}{2\sqrt{Z_2}} \leq \frac{2}{\sqrt{3}}$.*

Proof. Let us suppose, without loss of generality that $w = w(R) \geq h(R) = h$ and then $\rho = \rho(R) = w/h$. If $R_1, Z_2 = Square(R, \alpha)$, then $R(Z_2) = R$ and $p(Z_2) = w + h = (\rho+1)h$. In the same time, $s(Z_2) = (1-\alpha)s(R) = \rho(1-\alpha)h^2$. Then,

$$\frac{p(Z_2)}{2\sqrt{Z_2}} = \frac{\rho + 1}{2\sqrt{\rho(1-\alpha)}}.$$

As $x \mapsto \frac{1}{\sqrt{1-x}}$ is a increasing function on $[0, 1[$ and $\alpha \leq 1 - \frac{3(\rho(R)+1)^2}{16\rho(R)}$, we have

$$\frac{p(Z_2)}{2\sqrt{Z_2}} = \frac{\rho + 1}{2\sqrt{\rho(1-\alpha)}} \leq \frac{\rho + 1}{2\sqrt{\rho(1 - 1 + \frac{3(\rho+1)^2}{16\rho})}}.$$

Therefore,

$$\frac{p(Z_2)}{2\sqrt{Z_2}} \leq \frac{\rho + 1}{2\sqrt{\frac{3(\rho+1)^2}{16}}} = \frac{2}{\sqrt{3}}.$$

□

In the rest of the cases, (1.2) applies and

$$13/64 \leq \rho - \frac{3(\rho+1)^2}{16} < \rho\frac{s'}{s} < 2/5.$$

Let us first start with **Case (B2-b) (lines 42 to 48)**, we will show later that other cases are dominated by these ones.

Lemma 1.25 justifies the choice between performing successively `Guillotine` and `Square` (Lines 42 to 45) or `Superposition` (Lines 47 and 48). In both cases, there are exactly two simple zones: Z_2 and Z_3 in the first case, Z_2' and Z_3' in the second one (see Figure 1.32).

Lemma 1.25. *Let R be a rectangle, $\rho = \rho(R)$, α such that $1 - \frac{3(\rho+1)^2}{16\rho} < \alpha < \frac{2}{5\rho}$ and ϵ be such that $0 \leq \epsilon \leq \frac{2\rho\alpha^2}{5}$. Let $R', Z_3 = Guillotine(R, \alpha)$, $R_1, Z_2 = Square(R', \epsilon/\alpha)$ and $R'_1, Z_2, Z'_3 = Superposition(R, \alpha, \epsilon)$. Then,*

- *If $\epsilon \leq \frac{(1-\sqrt{1-\rho\alpha})^2}{\rho}$ then $p(Z_2) + p(Z_3) \leq p(Z'_2) + p(Z'_3)$.*
- *Otherwise $p(Z_2) + p(Z_3) \geq p(Z'_2) + p(Z'_3)$.*

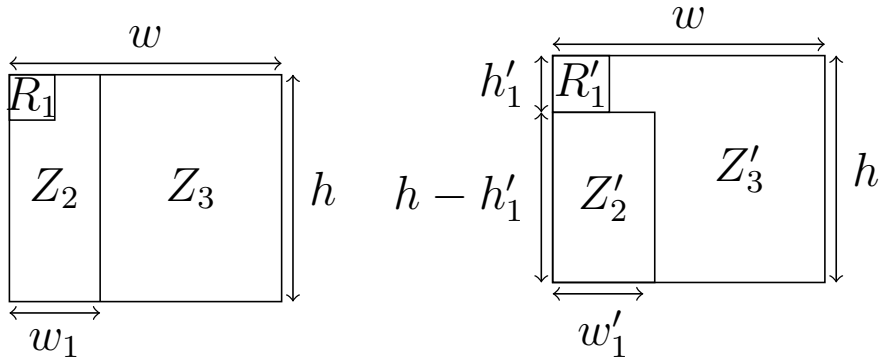


Figure 1.32: **Case (B2-b)**

Proof. Let us suppose, without loss of generality, that $w = w(R) \geq h(R) = h$. Let us denote $w_1 = w(Z_2)$, $w'_1 = w(Z'_2)$ and $h'_1 = h(R'_1) = w(R'_1)$ and, by construction, $h(Z_2) = h(Z_3) = h(Z'_3) = h$, $h(Z'_2) = h - h'_1$, $w(Z_3) = w - w_1$ and $w(Z'_3) = w - h'_1$ (see Figure 1.32). Trivially,

$$p(Z_2) + p(Z_3) = w_1 + h + w - w_1 + h = (\rho + 2)h$$

and

$$\begin{aligned} p(Z'_2) + p(Z'_3) &= w'_1 + h - h'_1 + w - h'_1 + h \\ &= (\rho + 2)h + w'_1 - 2h'_1 \\ &= p(Z_2) + p(Z_3) + w'_1 - 2h'_1. \end{aligned}$$

Thus, $p(Z_2) + p(Z_3) \leq p(Z'_2) + p(Z'_3) \iff w'_1 - 2h'_1 \geq 0$. Yet, by definition, $h'^2_1 = s(R'_1) = \epsilon wh$, then $h'_1 = \sqrt{\rho\epsilon} \times h$. Moreover, $w'_1(h - h'_1) = s(Z'_2) = (\alpha - \epsilon)s(R)$. Thus, $w'_1 = \frac{\rho(\alpha-\epsilon)}{1-\sqrt{\rho\epsilon}}h$ and finally

$$w'_1 - 2h'_1 = \left(\frac{\rho(\alpha - \epsilon)}{1 - \sqrt{\rho\epsilon}} - 2\sqrt{\rho\epsilon}\right) h.$$

Hence,

$$\begin{aligned} p(Z_2)+p(Z_3) \leq p(Z_2')+p(Z_3') &\iff \frac{\rho(\alpha-\epsilon)}{1-\sqrt{\rho\epsilon}} - 2\sqrt{\rho\epsilon} \geq 0 \\ &\iff 2\sqrt{\rho\epsilon} \leq \frac{\rho(\alpha-\epsilon)}{1-\sqrt{\rho\epsilon}} \\ &\iff 2\sqrt{\rho\epsilon} - 2\rho\epsilon \leq (\alpha-\epsilon)\rho \\ &\iff 2\sqrt{\rho\epsilon} \leq (\alpha+\epsilon)\rho \\ &\iff 2\sqrt{\epsilon} \leq (\alpha+\epsilon)\sqrt{\rho} \\ &\iff -\epsilon\sqrt{\rho} + 2\sqrt{\epsilon} - \alpha\sqrt{\rho} \leq 0. \end{aligned}$$

Moreover, $x \mapsto -\sqrt{\rho}x^2 + 2x - \alpha\sqrt{\rho}$ is positive over $[\frac{1-\sqrt{1-\alpha\rho}}{\sqrt{\rho}}, \frac{1+\sqrt{1-\alpha\rho}}{\sqrt{\rho}}]$. Since $\frac{1+\sqrt{1-\alpha\rho}}{\sqrt{\rho}} \geq \frac{1}{\sqrt{\rho}} \geq \sqrt{\frac{2}{5}}$ and $\sqrt{\epsilon} \leq \alpha \leq 2/5$ then, in any case $\sqrt{\epsilon} \leq \frac{1+\sqrt{1-\alpha\rho}}{\sqrt{\rho}}$. So,

$$\begin{aligned} p(Z_2)+p(Z_3) \leq p(Z_2')+p(Z_3') &\iff \sqrt{\epsilon} \leq \frac{1-\sqrt{1-\alpha\rho}}{\sqrt{\rho}} \\ &\iff \epsilon \leq \frac{(1-\sqrt{1-\rho(R)\alpha})^2}{\rho}, \end{aligned}$$

what achieves the proof of Lemma 1.25. □

In the first case (lines 42 to 45),

$$0 \leq \rho\frac{s''}{s} \leq (1-\sqrt{1-\rho(R)\alpha})^2.$$

In this case, Lemma 1.27 applies (with $\alpha = s'/s$ and $\epsilon = s''/s$, where the bounds on α comes from (1.2)) and proves claimed result. Lemma 1.26 is a technical lemma needed for the proof of Lemma 1.27.

Lemma 1.26. *Let* $f(x,y,z) = \frac{x+2}{2(\sqrt{y-z}+\sqrt{x-y})}$. *Then, for* $x \in [1,5/2]$, $y \in [x - \frac{3(x+1)^2}{16}, 2/5]$ *and* $z \in [0, (1-\sqrt{1-y})^2]$,

$$f(x,y,z) \leq \frac{2}{\sqrt{3}}.$$

Proof. First let us observe that f increases with z. Therefore, $\forall x, y$,

$$\begin{aligned} f(x,y,z) &\leq f(x,y,(1-\sqrt{1-y})^2) \\ &\leq \frac{x+2}{2(\sqrt{y-(1-\sqrt{1-y})^2}+\sqrt{x-y})} \\ &\leq \frac{x+2}{2(\sqrt{y-1+2\sqrt{1-y}-(1-y)}+\sqrt{x-y})} \\ &\leq \frac{x+2}{2(\sqrt{2(\sqrt{1-y}-(1-y))}+\sqrt{x-y})}. \end{aligned}$$

In the following, let us denote $z_{max} = (1 - \sqrt{1-y})^2$.

Let us consider $g_x(y) = \sqrt{2(\sqrt{1-y} - (1-y))} + \sqrt{x-y}$. Successive derivations lead to

$$g''_x(y) = -\frac{1}{4(x-y)^{3/2}} - \frac{(1 - \frac{1}{2\sqrt{1-y}})^2}{2\sqrt{2}(\sqrt{1-y} - (1-y))^{3/2}} - \frac{1}{4\sqrt{2}(1-y)^{3/2}\sqrt{\sqrt{1-y} - (1-y)}}.$$

Therefore, on $y \in [x - \frac{3(x+1)^2}{16}, 2/5]$, $g_x(y) \leq 0$ and g_x is concave.

Then for all $y \in [x - \frac{3(x+1)^2}{16}, 2/5]$, $g_x(y) \geq \min(g_x(x - \frac{3(x+1)^2}{16}), g_x(2/5))$.

Let us first suppose that $\min(g_x(x - \frac{3(x+1)^2}{16}), g_x(2/5)) = g_x(2/5)$. Then,

$$\begin{aligned} f(x,y,z) &\leq \frac{x+2}{2(\sqrt{2(\sqrt{1-y} - (1-y))} + \sqrt{x-y})} \\ &\leq \frac{x+2}{2g_x(y)} \\ &\leq \frac{x+2}{2g_x(2/5)} \\ &\leq \frac{x+2}{2(\sqrt{x-2/5} + \sqrt{2}\sqrt{\sqrt{3/5} - 3/5})}. \end{aligned}$$

Let us denote $A = \sqrt{2}\sqrt{\sqrt{3/5} - 3/5}$. Let us prove that $G_1(x) = \frac{x+2}{2(\sqrt{x-2/5}+A)}$ is inferior or equal to $\frac{2}{\sqrt{3}}$ for $x \in [1, 5/2]$. One can prove that

$$G'_1(x) = \frac{x - 14/5 + 2A\sqrt{x-2/5}}{4\sqrt{x-2/5}(\sqrt{x-2/5} + A)^2}.$$

Then $G'_1(x) \geq 0$ is equivalent to $x - 14/5 + 2A\sqrt{x-2/5} \geq 0$. As $\rho \leq 14/5$, it is equivalent to $2A\sqrt{x-2/5} \geq 14/5 - x$ which is equivalent to $4A^2(x-2/5) \geq 196/25 - 28/5x + x^2$. By replacing A^2 by $2(\sqrt{3/5} - 3/5)$ we obtain,

$$G'_1(x) \geq 0 \iff -x^2 + (8\sqrt{\frac{3}{5}} + \frac{4}{5})x - (\frac{16\sqrt{3}}{5\sqrt{5}} + \frac{148}{25}) \geq 0.$$

One can prove that this is equivalent to $x \geq \frac{2(1+2\sqrt{15})}{5} - \frac{4\sqrt{6}}{5} > 1$ and therefore G_1 is decreasing on $[1, \frac{2(1+2\sqrt{15})}{5} - \frac{4\sqrt{6}}{5}]$ and increasing on $[\frac{2(1+2\sqrt{15})}{5} - \frac{4\sqrt{6}}{5}, 5/2]$. Therefore $G_1(x) \leq \max(G_1(1), G_1(5/2))$. As we have

$$G_1(1) = \frac{3\sqrt{5}}{2(\sqrt{3} + \sqrt{2(\sqrt{15} - 3)})} < \frac{2}{\sqrt{3}}$$

$$G_1(5/2) = \frac{9\sqrt{5}}{2\sqrt{2}(\sqrt{21} + 2\sqrt{(\sqrt{15} - 3)})} < \frac{2}{\sqrt{3}}.$$

We can deduce that $G_1(x) \leq \frac{2}{\sqrt{3}}$ and, if $\min(g_x(x - \frac{3(x+1)^2}{16}), g_x(2/5)) = g_x(2/5)$, then $f(x, y, z) \leq \frac{2}{\sqrt{3}}$.

Let us now suppose that $\min(g_x(x - \frac{3(x+1)^2}{16}), g_x(2/5)) = g_x(x - \frac{3(x+1)^2}{16})$. Then,

$$\begin{aligned} f(x, y, z) &\leq \frac{x+2}{2(\sqrt{2(\sqrt{1-y} - (1-y))} + \sqrt{x-y})} \\ &\leq \frac{x+2}{2g_x(y)} \\ &\leq \frac{x+2}{2g_x(x - \frac{3(x+1)^2}{16})} \\ &\leq \frac{x+2}{2(\sqrt{\frac{3(x+1)^2}{16}} + \sqrt{2(\sqrt{1-y} - (1-y))})} \\ &\leq \frac{2}{\sqrt{3}} \times \frac{x+2}{x+1+\frac{4\sqrt{2}}{\sqrt{3}}\sqrt{\sqrt{1-y} - (1-y)}} = G_2(x). \end{aligned}$$

Hence,

$$\begin{aligned} G_2(x) \leq \frac{2}{\sqrt{3}} &\iff x + 2 \leq x + 1 + \frac{4\sqrt{2}}{\sqrt{3}}\sqrt{\sqrt{1-y} - (1-y)} \\ &\iff 1 \leq \frac{4\sqrt{2}}{\sqrt{3}}\sqrt{\sqrt{1-y} - (1-y)} \\ &\iff 1 \leq \frac{32}{3}(\sqrt{1-y} - (1-y)). \end{aligned}$$

If $X = \sqrt{1-y}$, then,

$$\begin{aligned} G_2(x) \leq \frac{2}{\sqrt{3}} &\iff 1 \leq \frac{32}{3}(X - X^2) \\ &\iff -\frac{32}{3}X^2 + \frac{32}{3}X - 1 \geq 0. \end{aligned}$$

Above inequality is equivalent to

$$\frac{1}{2} - \frac{\sqrt{10}}{8} \leq X \leq \frac{1}{2} + \frac{\sqrt{10}}{8}$$

and furthermore, is equivalent to

$$1 - \left(\frac{1}{2} + \frac{\sqrt{10}}{8}\right)^2 \leq y \leq 1 - \left(\frac{1}{2} - \frac{\sqrt{10}}{8}\right)^2.$$

As $1-(\frac{1}{2}+\frac{\sqrt{10}}{8})^2 \leq 13/64 \leq y$ and $1-(\frac{1}{2}-\frac{\sqrt{10}}{8})^2 \geq 2/5 \geq y$, then for all x, $G_2(x) \leq \frac{2}{\sqrt{3}}$ and we have our result.

So, in any case, we prove that $f(x,y,z) \leq \frac{2}{\sqrt{3}}$. □

Lemma 1.27. *Let R be a rectangle, $\rho = \rho(R)$ and α such that $1-\frac{3(\rho+1)^2}{16\rho} < \alpha < \frac{2}{5\rho}$ and let ϵ be such that $0 \leq \rho\epsilon \leq (1-\sqrt{1-\rho\alpha})^2$. Let us denote $R', Z_3 = Guillotine(R,\alpha)$ and $R_1, Z_2 = Square(R', \epsilon/\alpha)$. Then,*

$$\frac{p(Z_2)+p(Z_3)}{2(\sqrt{s(Z_2)}+\sqrt{s(Z_3)})} \leq \frac{2}{\sqrt{3}}.$$

Proof. With the same notations as in the proof of Lemma 1.25, $p(Z_2)+p(Z_3) = (\rho+2)h$. Moreover, $s(Z_2) = (\alpha-\epsilon)s(R) = \rho(\alpha-\epsilon)h^2$ and $s(Z_3) = (1-\alpha)s(R) = \rho(1-\alpha)h^2$. Then,

$$\frac{p(Z_2)+p(Z_3)}{2(\sqrt{s(Z_2)}+\sqrt{s(Z_3)})} = \frac{\rho+2}{2(\sqrt{\rho(\alpha-\epsilon)}+\sqrt{\rho(1-\alpha)})}.$$

We can thus use Lemma 1.26 with $x=\rho$, $y=\alpha\rho$ and $z=\epsilon\rho$, and we obtain claimed result. □

In the other case (lines 47 and 48), since

$$(1-\sqrt{1-\rho(R)\alpha})^2 \leq \rho\frac{s''}{s} < \frac{2\rho s'^2}{5s^2},$$

Lemma 1.29 applies and achieves the proof. Lemma 1.28 is a technical lemma needed for the proof of Lemma 1.29.

Lemma 1.28. *For any fixed x, y such that $1 \leq x \leq \frac{5}{2}$ and $\frac{1}{5} \leq y \leq \frac{2}{5}$, consider $f(z) = \frac{x+2-B(y,z)}{\sqrt{x-y}+\sqrt{y-z}}$, where $B(y,z) = 2\sqrt{z}-\frac{y-z}{1-\sqrt{z}}$, and denote $z_0 = (1-\sqrt{1-y})^2$ such that $B(y,z_0)=0$. Then for all z such that $z_0 \leq z \leq \frac{2y^2}{5}$, $f(z) \leq f(z_0)$.*

Proof. Computing $f(z_0)-f(z)$ yields to

$$f(z_0)-f(z) = \frac{(x+2)(\sqrt{x-y}+\sqrt{y-z})-(x+2-B(z))(\sqrt{x-y}+\sqrt{y-z_0})}{(\sqrt{x-y}+\sqrt{y-z})(\sqrt{x-y}+\sqrt{y-z_0})}.$$

Hence $f(z_0)-f(z)$ has same sign as $g(z) = B(z)(\sqrt{x-y}+\sqrt{y-z_0})-(x+2)(\sqrt{y-z_0}-\sqrt{y-z})$. We note $C(x,y) = \sqrt{x-y}+\sqrt{y-z_0}$, so that we can write the derivative $g'(z) = B'(z)C(x,y) - \frac{x+2}{2\sqrt{y-z}}$.

Let us first note that, given the bounds on x and y, $C(x,y) \geq \sqrt{x-y} \geq \sqrt{\frac{3}{5}}$, and $x+2 \leq \frac{7}{2}$. Since we have $z \leq \frac{2y^2}{5}$, we get $y-z \geq y(1-\frac{2y}{5}) \geq \frac{1}{5}(1-\frac{4}{25}) \geq \frac{1}{6}$, which yields to $\frac{1}{2\sqrt{y-z}} \leq \frac{\sqrt{6}}{2}$.

Similarly, the bounds on y provide the following bounds on $\sqrt{z}$,

- $\sqrt{z} \geq \sqrt{z_0} = 1 - \sqrt{1-y} \geq 1 - \sqrt{\frac{4}{5}}$ because $y \geq \frac{1}{5}$
- $\sqrt{z} \leq y\sqrt{\frac{2}{5}} \leq \zeta = \frac{2}{5}\sqrt{\frac{2}{5}}$ because $y \leq \frac{2}{5}$.

Computing $B'(z)$ gives $B'(z) = \frac{2-y-\sqrt{z}(2-\sqrt{z})}{\sqrt{z}(1-\sqrt{z})^2} = \frac{2-y}{\sqrt{z}(1-\sqrt{z})^2} - \frac{2-\sqrt{z}}{(1-\sqrt{z})^2}$. A simple analysis shows that both $u \mapsto \frac{1}{u(1-u)^2}$ and $u \mapsto -\frac{2-u}{(1-u)^2}$ are decreasing on $I = [1 - \sqrt{\frac{4}{5}}, \zeta]$. This implies that $B'(z) \geq B'(\zeta^2)$. Together with $y \leq \frac{2}{5}$, this yields to $B'(z) \geq \frac{2-\frac{2}{5}+\zeta(2-\zeta)}{\zeta(1-\zeta)^2} \geq 8$.

Putting all together provides $g'(z) \geq 8\sqrt{\frac{3}{5}} - \frac{9}{2}\frac{\sqrt{6}}{2} \geq 0$. Hence g is an increasing function with z, and since $g(z_0) = 0$ by construction, we get $f(z_0) - f(z) \geq 0$ for all $z \geq z_0$. □

Lemma 1.29. *Let R be a rectangle with $\rho = \rho(R)$, and*

- *α such that $1 - \frac{3(\rho+1)^2}{16\rho} < \alpha < \frac{2}{5\rho}$*
- *ϵ such that $(1 - \sqrt{1-\rho\alpha})^2 \leq \rho\epsilon \leq \frac{2\rho\alpha^2}{5}$*
- *$R_1, Z_2, Z_3 = Superposition(R, \alpha, \epsilon)$.*

Then,

$$\frac{p(Z_2) + p(Z_3)}{2(\sqrt{s(Z_2)} + \sqrt{s(Z_3)})} \leq \frac{2}{\sqrt{3}}.$$

Proof. With the same notations as in the proof of Lemma 1.25 (see on the right of Figure 1.32),

$$\begin{aligned} p(Z_2) + p(Z_3) &= (\rho+2)h + w_1' - 2h_1' \\ &= (\rho+2)h + (\frac{\rho(\alpha-\epsilon)}{1-\sqrt{\rho\epsilon}} - 2\sqrt{\rho\epsilon})h \\ &= (\rho + 2 + \frac{\rho(\alpha-\epsilon)}{1-\sqrt{\rho\epsilon}} - 2\sqrt{\rho\epsilon})h. \end{aligned}$$

Moreover, $s(Z_2) = (\alpha-\epsilon)s(R) = \rho(\alpha-\epsilon)h^2$ and $s(Z_3) = (1-\alpha)s(R) = \rho(1-\alpha)h^2$, so that

$$\frac{p(Z_2) + p(Z_3)}{2(\sqrt{s(Z_2)} + \sqrt{s(Z_3)})} = \frac{\rho + 2 + \frac{\rho(\alpha-\epsilon)}{1-\sqrt{\rho\epsilon}} - 2\sqrt{\rho\epsilon}}{2(\sqrt{\rho(\alpha-\epsilon)} + \sqrt{\rho(1-\alpha)})}.$$

If we set $x = \rho$, $y = \alpha\rho$ and $z = \epsilon\rho$ and $B(y,z) = 2\sqrt{z} - \frac{y-z}{1-\sqrt{z}}$, then we can use Lemma 1.28 to prove that $\forall x, y$ the worst case happens when $\epsilon\rho = (1 - \sqrt{1-\alpha\rho})^2$, which implies $\frac{\rho(\alpha-\epsilon)}{1-\sqrt{\rho\epsilon}} = 2\sqrt{\rho\epsilon}$. We can thus use Lemma 1.26 to conclude the proof. □

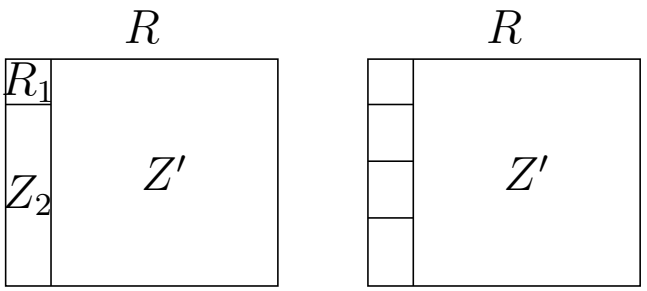


Figure 1.33: **Case (B2-a1)** and **Case (B2-a2′)**

Case (B2-a1) (Lines 24-25) and **Case (B2-a2′) (Lines 29-31)** are presented on Figure 1.33 and **Case (B2-a2″) (Lines 34-36 and 38-40)** is presented on Figure 1.34.

First, let us note that all four cases contain a large zone, denoted Z', and a set of smaller zones on the side. Among these smaller zones, there is at most one simple zone included in a rectangle whose aspect ratio is larger than 5/2. Therefore, the other zones are either composed (and thus not considered here) or their aspect ratio is smaller than 5/2 (which thanks to Lemma 1.3 yields the desired bound on the cost). The idea of the rest of the proof is to group this small zone with the large zone Z' and to show that their combined cost satisfies the bound.

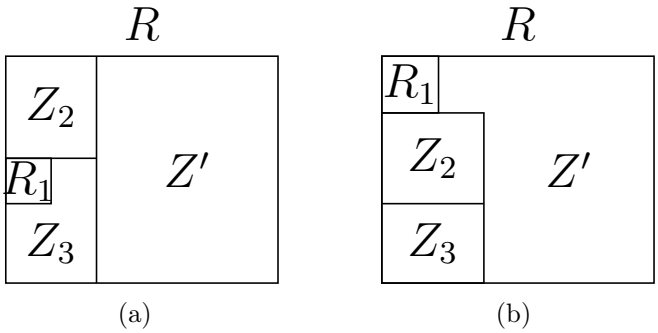


Figure 1.34: **Case (B2-a2″)**

A general proof can be obtained thanks to the remark that since small zones which are correct are not taken into account, all four cases can be reduced to one case, which is described in Figure 1.35, where Z_1 is the rectangle whose aspect ratio is "too large". Then, a general result expressed in Lemma 1.30 shows that all of these cases have a better cost ratio than the case (B2b) shown on Figure 1.32, what concludes the proof that $\frac{p(Z_1)+p(Z')}{2(\sqrt{s(Z_1)}+\sqrt{s(Z')})} \leq \frac{2}{\sqrt{3}}$.

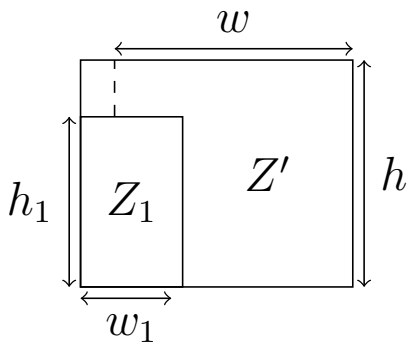


Figure 1.35: Subproblem studied in Lemma 1.30.

It only remains to show that all cases satisfy the assumptions of Lemma 1.30. To prove $\rho(Z_1) \geq \frac{4}{3}$, we can actually prove $\rho(Z_1) > 21/10$. In cases (B2-a1) and (B2-a2′), the question is not raised anyway unless $\rho(Z_1) > 5/2$. In case (B2-a2″), $\rho(Z_1) \leq 21/10$ would imply that $s_{m-2} \leq s_{m-1} \leq \frac{21s'}{s}$ and then $s''' \geq \frac{2s'}{5s}$, what is a contradiction with the assumption $s''' < \frac{2s'}{5s}$. In addition $s(Z') = s - s' \geq s - s(1 - \frac{3(\rho(R)+1)^2}{16\rho(R)})$ and $s = \rho w^2$. Hence we can indeed use Lemma 1.30 to solve these cases (the case in Figure 1.34(a) does not exactly reduce like the others, but Lemma 1.30 and its proof are adaptable to this case).

Lemma 1.30. *Let Z_1 and Z' be as depicted in Figure 1.35. If $\rho(Z_1) = \frac{h_1}{w_1} \geq 4/3$ and $\frac{s(Z')}{h^2} \geq \frac{3(\rho(R)+1)^2}{16}$, then $\frac{p(Z_1)+p(Z')}{2(\sqrt{s(Z_1)}+\sqrt{s(Z_2)})}$ is maximum when $h_1 = h$.*

Proof. Let us denote $x = \rho(Z_1)$. Trivially, $p(Z_1) + p(Z') = w_1(\frac{w+h}{w_1} + 1 + x)$ and $\sqrt{s(Z_1)} + \sqrt{s(Z')} = w_1(\frac{\sqrt{s(Z')}}{w_1} + \sqrt{x})$. Therefore, $\frac{p(Z_1)+p(Z')}{2(\sqrt{s(Z_1)}+\sqrt{s(Z')})} = \frac{\frac{w+h}{w_1}+1+x}{2(\frac{\sqrt{s(Z')}}{w_1}+\sqrt{x})} = f(x)$. $f'(x) = \frac{2\frac{\sqrt{s(Z')}}{w_1}\sqrt{x}+x-\frac{w+h}{w_1}+1}{4\sqrt{x}(\frac{\sqrt{s(Z')}}{w_1}+\sqrt{x})^2}$ and therefore $f'(x) \geq 0$ is equivalent to $2\frac{\sqrt{s(Z')}}{w_1}\sqrt{x} + x - (\frac{w+h}{w_1} + 1) \geq 0$. Since $x \geq 1$, we have to prove that $2\sqrt{s(Z')}\sqrt{x} - (w+h) \geq 0$. Let us denote $\rho = \rho(R)$, so that $w+h = (\rho+1)h$. Let us prove that $2\sqrt{s(Z')}\sqrt{x} - (\rho+1)h \geq 0$.

$$\begin{aligned} 2\sqrt{s(Z')}\sqrt{x} - (\rho+1)h \geq 0 &\iff 2\sqrt{s(Z')}\sqrt{x} \geq (\rho+1)h \\ &\iff 4s(Z')x \geq (\rho+1)h^2 \\ &\iff x\frac{s(Z')}{h^2} \geq \frac{(\rho+1)^2}{4}. \end{aligned}$$

Since we assumed that $x \geq 4/3$ and $\frac{s(Z')}{h^2} \geq \frac{3(\rho+1)^2}{16}$, we have $x\frac{s(Z')}{h^2} \geq \frac{(\rho+1)^2}{4}$ and we obtain the result claimed above.

Therefore, f is increasing with x, and, when w_1 is fixed, the cost ratio is maximum when $h_1 = h$. □

Case (B2-c) (Lines 50-51) can been seen as the extremal case of Case (B2-a2″-2) ($s(R_1) \to 0$) and is thus covered by its study (worst case scenario is when $s(R_1)$ increase).

This completes the proof of all possible cases of NRRP and therefore the proof of Theorem 1.20. □

Note that the bound is tight, the worst case scenario is more or less the same than the one of SNRRP, see Figure 1.36.

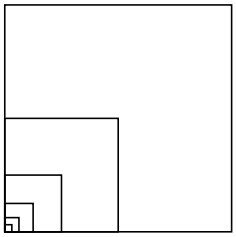


Figure 1.36: (Reminder) Worst case scenario for SNRRP and NRRP.

Complexity

Let us first consider the cost of NRRP without the `Packing` routine. Each routine (`Guillotine`, `Tripartition`, `Square`, `Superposition`) induces $O(1)$ operations. Therefore the only costs during a call of NRRP are the recursive calls and the search for the smallest k such that $\sum_{i=1}^{k} s_i \geq \frac{2s}{5\rho}$. With a $O(m)$ pre-computation of the partial sums $\sum_{i=1}^{j} s_i$ (with $j \in [1, m]$) before the first call of NRRP each continuous partial sum can be done with $O(1)$ operations and therefore the search for the k can be done in $O(\log m)$ with binary search. Trivially, there are at most m calls of NRRP (the size of the instance strictly decreases at each recursive call) and thus the complexity of NRRP without the `Packing` routine is $O(m \log m)$. Note that the same analysis applies for RRP and SNRRP.

The evaluation of the cost of `Packing` routine is a bit more complex. Let us assume that `Packing` is used at each call of NRRP and let $C(m)$ denote its cost for an instance of size m. During a call of `Packing`, the instance is split into $k \in [2, m]$ instances, each of size x_i such that $\sum_{i=1}^{m} x_i = m$. For the same pre-computation of the partial sums as before we can do splitting in $k-1$ binary searches each of cost $\lambda \log m$ where λ is a given constant. Thus

$$C(m) \leq \lambda(k-1)\log m + \sum_{i=1}^{k} C(x_i),$$

where $C(1) = 1$.

Lemma 1.31. $C(m) \leq \lambda m \log(m) + m$.

Proof. First let us prove the following result: for all integers (m, j) such that $1 \leq j < m$, $(m-j)\log(m-j)+j\log(j) \leq (m-1)\log(m-1)$. For this purpose let us consider $f(x) = (m-x)\log(m-x) + x\log(x)$ defined on $[1, m-1]$. $f'(x) = -\log(m-x) - 1 + \log(x) + 1 = \log\left(\frac{x}{m-x}\right)$. Thus,

$$f'(x) \geq 0 \Leftrightarrow \frac{x}{m-x} \geq 1 \Leftrightarrow x \geq \frac{m}{2}.$$

Hence f is decreasing on $[1, \frac{m}{2}]$ and increasing on $[\frac{m}{2}, m-1]$ and therefore $f(x) \leq \max(f(1), f(m-1)) = (m-1)\log(m-1)$.

Let us now prove Lemma 1.31 by induction on m.

For $m = 1$, $C(1) = 1 \leq \lambda \times 1 \times \log(1) + 1$ and that proves the case.

Otherwise let us suppose that the result holds true for every $m' \in [1, m-1]$. In this case,

$$\begin{aligned} C(m) &= \lambda(k-1)\log(m) + \sum_{i=1}^{k} C(x_i) \\ &\leq \lambda(k-1)\log(m) + \sum_{i=1}^{k} (\lambda x_i \log(x_i) + x_i) \\ &\leq \lambda(k-1)\log(m) + \lambda \sum_{i=1}^{k} (x_i \log(x_i)) + m. \end{aligned}$$

To end the proof, let us prove that $\sum_{i=1}^{k} x_i \log(x_i) \leq (m-k+1)\log(m-k+1) \leq (m-k+1)\log(m)$. The result can be proved by induction on k. The result holds trivially true for $k = 1$. Otherwise, assume the result being true for $k-1$, then

$$\begin{aligned} \sum_{i=1}^{k} x_i \log(x_i) &= \sum_{i=1}^{k-1} (x_i \log(x_i)) + x_k \log(x_k) \\ &\leq \left(\sum_{i=1}^{k-1} x_i - k + 2\right) \log\left(\sum_{i=1}^{k-1} x_i - k + 2\right) + x_k \log(x_k) \\ &\leq \left(\sum_{i=1}^{k} x_i - k + 1\right) \log\left(\sum_{i=1}^{k} x_i - k + 1\right) \\ &\leq (m-k+1)\log(m-k+1). \end{aligned}$$

And we can finally conclude that

$$\begin{aligned} C(m) &\leq \lambda(k-1)\log(m) + \lambda\sum_{i=1}^{k}(x_i\log(x_i)) + m \\ &\leq \lambda(k-1)\log(m) + \lambda(m-k+1)\log(m-k+1) + m \\ &\leq \lambda(k-1)\log(m) + \lambda(m-k+1)\log(m) + m \\ C(m) &\leq \lambda m\log(m) + m. \end{aligned}$$

□

Thus, with Lemma 1.31, the total cost of the `Packing` routine calls is proven to be $O(m\log m)$, thus the claimed complexity of NRRP.

Theorem 1.32. *$NRRP(R, \{s_1, \ldots, s_m\})$ runs in at most $O(m\log m)$ operations.*

1.5 SFCP (Space-Filling Curve Partitioning)

In this section, we present SFCP, another algorithm to solve the PERI-SUM problem. Even if its approximation ratio ($\frac{3\sqrt{3}}{\sqrt{11}} \simeq 1.57$) is worse than the ones of NRRP or RRP, this study is interesting for two reasons:

- Space-filling curves (main ingredient of SFCP) are a very common tool to preserve data-locality, PERI-SUM aims at the same goal. Hence it is of interest to provide analysis of an algorithm using space-filling curve to solve PERI-SUM.
- SFCP is designed to solve the discrete variant of PERI-SUM without additional rounding or adaptation like RRP-Accurate (it can also be used to solve PERI-SUM in its original version). Therefore, this is the only known guaranteed approximation ratio for this problem.

1.5.1 Presentation of Space-Filling Curves

We considered a space-filling curve, *i.e.* a bijective function from $[1, N]^2$ to $[1, N^2]$ and more precisely the Hilbert space-filling curve Hilbert [1891], see Figure 1.37. Among the space-filling curves, Hilbert has the property to be continuous and to have a fractal approach (the filling is made locally at every scale). Then it is a classical technique to preserve data locality while enforcing a good load balancing (Deveci *et al.* [2016]; Heinecke and Bader [2008]; Manne and Sørevik [1996]; Pilkington *et al.* [1994]).

In what follows, we consider an $N \times N$ square and suppose that N is a power of 2. Note that we can return to PERI-SUM by simply rescaling the

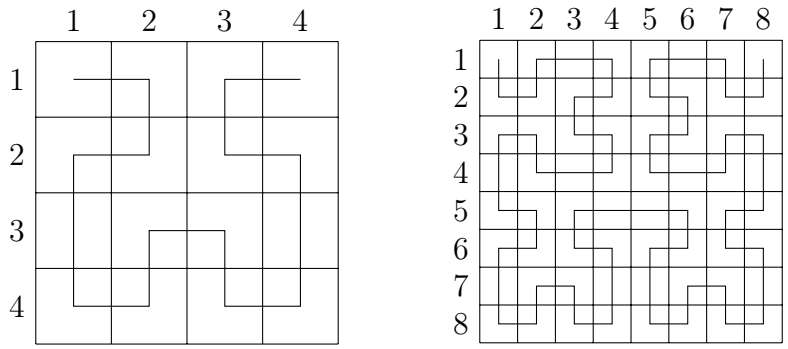


Figure 1.37: Hilbert's curve for $N = 4$ and $N = 8$.

square (in particular if we divide the perimeters of the zone built by SFCP on an $N \times N$ square by N, we have the perimeters for the $[0, 1] \times [0, 1]$ square). Note also that we now consider a discrete problem and no longer a continuous one (there are N^2 elements on the square we allocate to processors). Note that the notations from PERI-SUM can be adapted easily. If Z is a subset of $[1, N] \times [1, N]$, now $\Pi_1(Z) = \{i,\ (i, j) \in Z\}$ and $\Pi_2(Z) = \{j,\ (i, j) \in Z\}$ are the projections of Z in this case (and $\pi_i(Z) = |\Pi_i(Z)|$ for $i \in \{1, 2\}$).

Let us first note, as described in Lemma 1.33, that the projection of the union of two subsets of $[1, N]\times[1, N]$ is smaller than the sum of the projections, even for disjoint subsets.

Lemma 1.33. *Let X and Y be two subsets of $[1, N]^2$. Then, $\forall i \in [1, 2]$,*

$$\pi_i(X \cup Y) \leq \pi_i(X) + \pi_i(Y).$$

Proof. We prove here the property for π_1 and we can prove it similarly for π_2. We have

$$\begin{aligned}\Pi_1(X \cup Y) &= \{i, (i, j) \in X \cup Y\} \\ &= \{i, (i, j) \in X\} \cup \{i, (i, j) \in Y\} \\ &= \Pi_1(X) \cup \Pi_1(Y).\end{aligned}$$

Therefore,

$$\pi_1(X\cup Y) = |\Pi_1(X\cup Y)| = |\Pi_1(X)\cup\Pi_1(Y)| \leq |\Pi_1(X)|+|\Pi_1(Y)| = \pi_1(X)+\pi_1(Y).$$

□

Let us now define a key element of this section: the q-squares. They are important to define the key properties we are looking for in the space-filling curves and for the proof of the approximation ratio.

Definition 1.1. Let us denote as q-squares all the subsquares of the square $[1, N] \times [1, N]$ which are of the form $[1 + n_1 \times 2^q, (n_1 + 1) \times 2^q] \times [1 + n_2 \times 2^q, (n_2 + 1) \times 2^q]$ where $n_1, n_2 \in [0, N/2^q - 1]^2$.

Intuitively q-squares are partitioning a square into a grid of $N/2^q \times N/2^q$ (see Figure 1.38 for a better visualization).

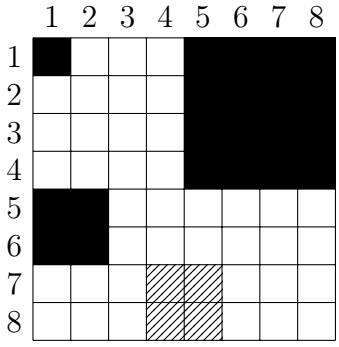


Figure 1.38: Here $N = 8$ and the filled-in-black areas are (from the smallest one to the biggest one) 0-square, 1-square and 2-square. However the dashed area is not a q-square.

The study we propose does not only concern the Hilbert's curve, even if we use this curve in particular for illustration and implementation. It may apply to every space-filling curve that can be seen as a function $H : [1, N^2] \to [1, N]^2$ that fulfils the following two conditions.

Property 1.1. If S_q is a q-square, then there exists an integer n such that $H([n, n + 4^q - 1]) = S_q$.

Property 1.2. For all $n \in [1, N^2-1]$, if $(i_1, j_1) = H(n)$ and $(i_2, j_2) = H(n+1)$, then $i_1 = i_2$ or $j_1 = j_2$.

The first one means that the path through the square $[1, N]^2$ is built, at every level, one q-square after an other. The second means that there are no diagonal moves. From these two properties we can conclude than the second one is transferred at any level to the q-squares, which is described in the following corollary.

Corollary 1.34. *Let H be a function from $[1, N^2]$ to $[1, N]^2$ which satisfies Property 1.1 and Property 1.2. Let q be in $\mathbb{N}$ and S_1, S_2 be two q-squares with $S_1 = I_1 \times J_1$ and $S_2 = I_2 \times J_2$. If there is a k such that $H([k, k + 4^q - 1]) = S_1$ and $H([k + 4^q, k + 2 \times 4^q - 1]) = S_2$, then either $I_1 = I_2$ or $J_1 = J_2$.*

Proof. First if $S_1 = I_1 \times J_1$ and $S_2 = I_2 \times J_2$ are two q-squares, then $I_1 \cap I_2 \neq \emptyset$ (respectively $J_1 \cap J_2 \neq \emptyset$) implies $I_1 = I_2$ (respectively $J_1 = J_2$). This can be

proved simply since $I_1 = [1 + n_1 \times 2^q, (n_1 + 1) \times 2^q]$ and $I'_1 = [1 + n'_1 \times 2^q, (n'_1 + 1) \times 2^q]$ with $n_1, n'_1 \in [0, N/2^q - 1]^2$ and thus if $n_1 \neq n'_1$ there could be no intersection.

Let us now prove Corollary 1.34 by induction on q.

The case $q = 0$ corresponds exactly to Property 1.2 and therefore holds true.

Let us suppose that the result holds true for $q - 1$. Let $S_1 = I_1 \times J_1$ and $S_2 = I_2 \times J_2$ be two q-squares and k such that $H([k, k + 4^q - 1]) = S_1$ and $H([k + 4^q, k + 2 \times 4^q - 1]) = S_2$. Thanks to Property 1.1 if $S_{1,1}, S_{1,2}, S_{1,3}$ and $S_{1,4}$ are the $(q-1)$-squares that compose S_1, then there is k_1, k_2, k_3 and k_4 such that for all $i \in \{1, 2, 3, 4\}$, $H([k_i, k_i + 4^{q-1} - 1]) = S_{1,i}$. If we suppose that $k_1 < k_2 < k_3 < k_4$, since $\bigcup [k_i, k_i + 4^{q-1} - 1] = [k, k + 4^q - 1]$, then,

- $k_1 = k$
- $k_i = k_{i-1} + 4^{q-1}$ for $1 < i \leq 4$.

Similarly, let us define $S_{2,1}, S_{2,2}, S_{2,3}$ and $S_{2,4}$, $(q-1)$-squares included in S_2 and k'_1, k'_2, k'_3, k'_4 equivalent to k_1, k_2, k_3, k_4 for $S_{2,1}, S_{2,2}, S_{2,3}$ and $S_{2,4}$. Similarly

- $k'_1 = k + 4^q$
- $k'_i = k'_{i-1} + 4^{q-1}$ for $1 < i \leq 4$.

In addition

$$k + 4^q = k_1 + 4 \times 4^{q-1} = k_2 + 3 \times 4^{q-1} = k_3 + 2 \times 4^{q-1} = k_4 + 4^{q-1}$$

and then $H([k_4, k_4 + 4^{q-1} - 1]) = S_{1,4}$ and $H([k_4 + 4^{q-1}, k_4 + 2 \times 4^{q-1} - 1]) = S_{2,1}$. Let us denote $S_{1,4} = I'_1 \times J'_1$ and $S_{2,1} = I'_2 \times J'_2$ then, by assumption $I'_1 = I'_2$ or $J'_1 = J'_2$. Thus $I_1 \cap I_2 \neq \emptyset$ or $J_1 \cap J_2 \neq \emptyset$ and therefore $I_1 = I_2$ or $J_1 = J_2$ what achieves the proof for q. □

Relying on H with such properties, SFCP simply makes the partitioning going through the squares until we reach the desired area, as described in Algorithm 1.10.

1.5.2 Approximation Ratio

We claim that SFCP is a $\frac{3\sqrt{3}}{\sqrt{11}}$-approximation ($\frac{3\sqrt{3}}{\sqrt{11}} \simeq 1.57$) algorithm for PERI-SUM (Theorem 1.35), and prove this ratio in this section.

Theorem 1.35. *SFCP is a $\frac{3\sqrt{3}}{\sqrt{11}}$-approximation algorithm for PERI-SUM.*

Algorithm 1.10 : SFCP

Input : a set of integers $\{s_1, \ldots, s_p\}$ such that $\sum s_i = N^2$
Output : For each $1 \leq i \leq p$, a zone Z_i such that $s(Z_i) = s_i$ and $\bigcup Z_i = [1, N]^2$

$s_{tot} = 0$
foreach $k \in [1, p]$ **do**
  $Z_k = H([s_{tot}, s_{tot} + s_k])$
  $s_{tot} = s_{tot} + s_k + 1$

Proof. Let us prove that the ratio holds for every zone Z_k and therefore holds for their sum (*i.e.* for all k, $\frac{p(Z_k)}{2\sqrt{s_k}} \leq \rho$ implies $\frac{\sum p(Z_k)}{\sum 2\sqrt{s_k}} \leq \rho$, see Lemma 1.21).

Let us focus on the study of general subsets of $[1, N] \times [1, N]$ that are the images by H of an interval of $[1, N^2]$ (*i.e.* all the possible Z_k for every possible distribution of the $\{s_1, \ldots, s_m\}$). In order to refine the analysis we distinguish different kinds of q-squares.

Definition 1.2. Let Z be a subset of $[1, N]^2$. A q-square S_q is said to be *completed* by Z if $S_q \subseteq Z$. A q-square S_q is said to be *partially completed* by Z if $S_q \cap Z \neq \varnothing$ and $S_q \nsubseteq Z$.

Definition 1.3. Let Z be a subset of $[1, N]^2$. Let us denote q_Z as the largest q such that there exists a q-square completed by Z.

For a better view of these two definitions see Figure 1.39, that also illustrates the statement of the next lemma, *i.e.* the fact that for every q there are at most two partially-completed q-squares.

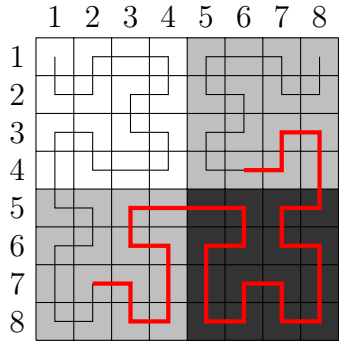


Figure 1.39: If the zone Z is the one defined by the red path, then the black zone is a completed 2-square and the gray zones are partially completed 2-squares. At the same time, in this case $q_Z = 2$.

Lemma 1.36. *Let Z be a subset of $[1, N]^2$ such that there exists an interval I of $[1, N^2]$ such that $H(I) = Z$. Then, for all q, at most two q-squares are partially completed by Z.*

Proof. Let $S_1, S_2, \ldots, S_k$ be all the q-squares such that for $i \in [1,k]$, $S_i \cap Z \neq \varnothing$. By assumption on H (Property 1.1), for all $i \in [1,k]$, there exists n_i such that $S_i = H([n_i, n_i + 4^q - 1])$. Let us assume that $n_1 < n_2 < \ldots < n_k$ and let us denote $I_i = [n_i, n_i + 4^q - 1]$. Since H is bijective, we also know that for $i \neq j$, $n_j \notin I_i$ and, therefore, for all $x, y, z \in I_i \times I_j \times I_l$, $i < j < l$ implies $x < y < z$. At the same time, we know that $Z = H(I)$ where I is an interval of $[1, N^2]$. By assumption, we know that there exists x_1 and x_k such that $x_1 \in I_1 \cap I$ and $x_k \in I_k \cap I$. We know that $x_1 < x_k$ and $x_1, x_k \in I$, then $[x_1, x_k] \subseteq I$. In addition, as for all $y \in I_j$, with j such that $1 < j < k$, $x_1 < y < x_k$, and then $y \in [x_1, x_k]$ which implies $y \in I$. Thus for all $j \in]1, k[$, $S_j \subseteq H(I) = Z$ and therefore S_j is completed by Z. So we have proved that only S_1 and S_k can be partially completed by Z and so there are at most two partially completed q-squares. □

A direct consequence of Lemma 1.36 is that we can bound the number of completed q_Z-squares by 6 (see Figure 1.40 for a visualization of a case with 6 such squares).

Lemma 1.37. *Let Z be a subset of $[1, N]^2$ such that there exists an interval I of $[1, N^2]$ such that $H(I) = Z$. Then there are at most 6 completed q_Z-squares and at most 2 partially completed q_Z-squares and they fit in at most two $(q_Z + 1)$-squares.*

Proof. The second statement is a direct corollary of Lemma 1.36. Indeed, if Z fits in at least three $(q_Z + 1)$-squares, then one of them, at least, must be completed, which is a contradiction with the definition of q_Z.

In order to prove the first statement, we can simply notice that a $(q+1)$-square consists of exactly 4 q-squares. Suppose now that Z has at least 7 completed q_Z-squares. Then the only possible partition of these squares into two $(q_Z + 1)$-squares is one of these two $(q_Z + 1)$-squares containing three completed q_Z-squares and the other one containing four completed q_Z-squares. Therefore this last one is also completed, which is in contradiction with the definition of q_Z. □

The key point in this proof is the partially completed q_Z-squares. Thanks to Lemma 1.37 we know that there are at most two such squares. Let $S_{1,Z}$ and $S_{2,Z}$ be the two q_Z-squares partially completed by Z. Let n_Z be the number of completed q_Z-squares and let $s_{i,Z} = \frac{|S_{i,Z} \cap Z|}{4^{q_Z}}$, $i \in \{1, 2\}$. $s_{i,Z}$ represents the fractions of $S_{1,Z}$ and $S_{2,Z}$ that are intersected by Z (both are in $[0, 1[$). We have $s_Z = 4^q(n_Z + s_{1,Z} + s_{2,Z})$.

The values $\pi_1(Z)$ and $\pi_2(Z)$ are more difficult to estimate. Let us first define $L_Z = \frac{p(Z \setminus (S_{1,Z} \cup S_{2,Z}))}{2^{q_Z}}$, the scaled size of the projections of the zones defined by the q_Z-squares completed by Z. Let us now define

$$x_{1,Z} = \frac{p(Z \setminus S_{2,Z}) - p(Z \setminus (S_{1,Z} \cup S_{2,Z}))}{2^{q_Z}}$$

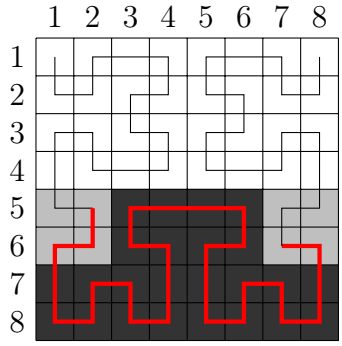


Figure 1.40: In this case $q_Z = 1$ and there are six completed 1-squares and two partially completed 1-squares.

$$x_{2,Z} = \frac{p(Z \setminus S_{1,Z}) - p(Z \setminus (S_{1,Z} \cup S_{2,Z}))}{2^{q_Z}}.$$

These two values are the parts of $p(Z)$ that come exclusively from the partially completed q_Z-squares. A better view of these values can be found in Figure 1.41. However, as shown in Figure 1.41(c), there are some cases where the $x_{i,Z}$ refers to the same data, therefore we have only an inequality (what is enough to establish the bound). Hence $p(Z) \leq 2^q(L_Z + x_{1,Z} + x_{2,Z})$.

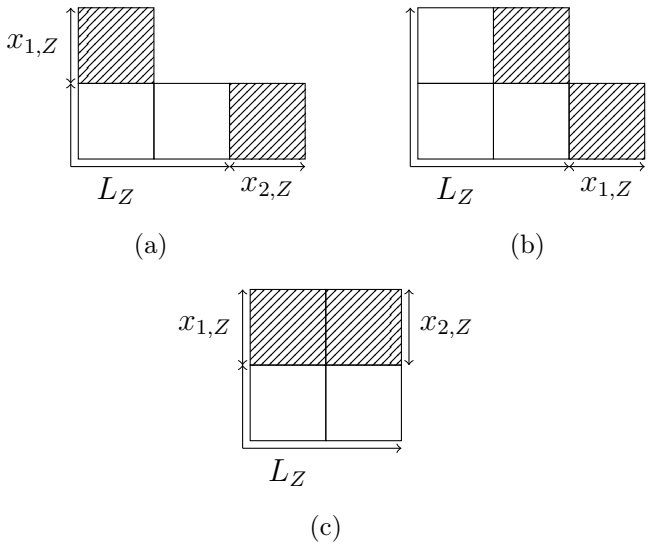


Figure 1.41: Illustrations of the previous definitions.

Therefore,

$$\rho_Z = \frac{p(Z)}{2\sqrt{s_Z}} \leq \frac{2^{q_Z}(L_Z + x_{1,Z} + x_{2,Z})}{2\sqrt{4^{q_Z}(n_Z + s_{1,Z} + s_{2,Z})}} \leq \frac{L_Z + x_{1,Z} + x_{2,Z}}{2\sqrt{n_Z + s_{1,Z} + s_{2,Z}}}.$$

Let us prove a result on the sizes of n_Z and L_Z.

Lemma 1.38. $L_Z \leq n_Z + 1$.

Proof. Let us prove this result by induction on n_Z. If $n_Z = 1$ then $L_Z = 2$. Indeed if $n_Z = 1$ then $Z \setminus (S_{1,Z} \cup S_{2,Z})$ is a single q_Z-square. Therefore $L_Z = \frac{2*2^{q_Z}}{2^{q_Z}} = 2$.

Elsewhere assume that the result holds true for $n_Z = k$ and consider $n_Z = k + 1$. Let Z' be Z minus the last q_Z-square of Z (in the order where the Hilbert's curve goes through them). $n_{Z'} = n_Z - 1 = k$. Therefore $L_{Z'} \leq n_{Z'} + 1$. At the same time Corollary 1.34 states that the removed q_Z-square shares at least one projection with Z'. Therefore $L_Z - L_{Z'} \leq 1$, what leads to $L_Z \leq 2 + (n_Z - 1) = n_Z + 1$. □

Let us now establish a relation between $x_{i,Z}$ and $s_{i,Z}$. Let us define f_q as follows

$$f_q(x) = \begin{cases} 0 & \text{if } q = 0 \text{ and } x = 0 \\ \frac{1}{3} & \text{if } q = 0 \text{ and } x \neq 0 \\ \frac{1}{4} f_{q-1}(2x) & \text{if } q > 0 \text{ and } x \leq \frac{1}{2} \\ \frac{1}{4} + \frac{1}{4} f_{q-1}(2(x - \frac{1}{2})) & \text{if } q > 0 \text{ and } x > \frac{1}{2} \end{cases}$$

Let us first prove that f_q is an increasing function that can be bounded, as stated in Lemma 1.39.

Lemma 1.39. *For all q and for all x, f_q is a increasing function on $[0, 1]$ and for all $x \in [0, 1]$, $\frac{1}{3}x^2 \leq f_q(x) \leq \frac{1}{3}$.*

Proof. We first prove the inequality $\forall x \in [0, 1]$, $\frac{1}{3}x^2 \leq f_q(x) \leq \frac{1}{3}$, by induction on q. If $q = 0$ we have two cases: if $x = 0$, then $\frac{1}{3} \times x^2 = \frac{1}{3} \times 0 = 0 = f_0(0) \leq \frac{1}{3}$, else $\frac{1}{3} \times x^2 \leq \frac{1}{3} = f_0(x) \leq \frac{1}{3}$. Therefore, the property holds true for $q = 0$.

Let us suppose this holds true for $q - 1$. We also have two cases, $x \leq \frac{1}{2}$ and $x > \frac{1}{2}$. In the first case

$$f_q(x) = \frac{1}{4} f_{q-1}(2x) \leq \frac{1}{4} \times \frac{1}{3} \leq \frac{1}{3}$$

and

$$f_q(x) = \frac{1}{4} f_{q-1}(2x) \geq \frac{1}{4} \times \frac{1}{3}(2x)^2 \geq \frac{1}{3}x^2$$

In the second case

$$f_q(x) = \frac{1}{4} + \frac{1}{4} f_{q-1}(2(x - \frac{1}{2})) \leq \frac{1}{4} + \frac{1}{4} \times \frac{1}{3} \leq \frac{1}{4} + \frac{1}{12} = \frac{1}{3}$$

and

$$\begin{array}{ll} f_q(x) & = \frac{1}{4} + \frac{1}{4} f_{q-1}(2(x - \frac{1}{2})) \\ f_q(x) & \geq \frac{1}{4} + \frac{1}{4} \times \frac{1}{3}(2x-1)^2 \\ f_q(x) & \geq \frac{1}{4} + \frac{1}{4} \times \frac{1}{3}(4x^2 - 4x + 1) \\ f_q(x) & \geq \frac{1}{4} + \frac{1}{3}x^2 - \frac{x}{3} + \frac{1}{12} \\ f_q(x) & \geq \frac{1}{3}x^2 + \frac{1-x}{3} \\ f_q(x) & \geq \frac{1}{3}x^2. \end{array}$$

For the increasing part we also use an induction on q. The fact is obvious for $q = 0$. Let us suppose that for all $x, y \in [0,1]^2$, $x \leq y$ implies $f_{q-1}(x) \leq f_{q-1}(y)$. If $x \leq y \leq \frac{1}{2}$, then

$$f_q(x) = \frac{1}{4} f_{q-1}(2x) \leq \frac{1}{4} f_{q-1}(2y) = f_q(y).$$

If $x \leq \frac{1}{2} < y$, then

$$f_q(x) = \frac{1}{4} f_{q-1}(2x) \leq \frac{1}{4} \leq \frac{1}{4} + \frac{1}{4} f_{q-1}(2(y - \frac{1}{2})) = f_q(y).$$

If $\frac{1}{2} \leq x \leq y$, then

$$f_q(x) = \frac{1}{4} + \frac{1}{4} f_{q-1}(2(x - \frac{1}{2})) \leq \frac{1}{4} + \frac{1}{4} f_{q-1}(2(y - \frac{1}{2})) = f_q(y),$$

what achieves the induction proof. □

Let us now prove, as stated in Lemma 1.40, that f_q is a function such that $s_{i,Z} \geq f_q(x_{i,Z})$, where q depends on the size of the q-square we are considering (in practice $q = q_Z$).

Lemma 1.40. *Let S_q be a q-square. Let Z be a connected subset of $[1,N] \times [1,N]$ such that $Z \cap S_q \neq \varnothing$, $Z = H(I)$ and $Z \not\subseteq S_q$. Let $x = \frac{\pi_1(Z \cap S_q)}{2^q}$, $y = \frac{\pi_2(Z \cap S_q)}{2^q}$ and $s = \frac{|Z \cap S_q|}{4^q}$. Then $s \geq f_q(x)$ and $s \geq f_q(y)$.*

Proof. Let us prove this lemma by induction on q. If $q = 0$, we immediately get $x = 1$, $y = 1$ and $s = 1$ (0-squares have an area of exactly one, so either they are completed or they are not intersected at all). As $f_0(1) = \frac{1}{3}$, then $s \geq f_0(x)$ and $s \geq f_0(y)$.

Let us suppose that the property holds true for $q - 1$. Let us consider the q-square as a subdivision of four $(q-1)$-squares. By hypothesis, and because of Lemma 1.36 we know that at most one of these four squares can be partially completed, the hypothesis $Z \not\subseteq S_q$ ensuring that if there are two q-squares partially completed by Z, the second one is outside of S_q. Therefore our original q-square is composed of 0, 1, 2, 3 or 4 completed $(q-1)$-squares plus one or zero partially completed $(q-1)$-square. Let us consider these subcases. In Figure 1.43 one can see the different cases that will be considered. The other cases can be solved by symmetry considerations or cannot occur, such as the ones in Figure 1.42, which lead to a contradiction with Corollary 1.34.

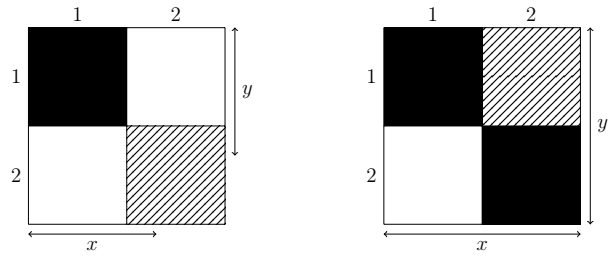


Figure 1.42: Impossible cases, completed squares are in black, partially completed one is dashed

Let us consider the case depicted in Figure 1.43(a). In this case, there is one partially completed $(q-1)$-square. Therefore, it can be observed that $x, y \leq \frac{1}{2}$. Therefore by rescaling x and y we can use the recurrence hypothesis. Indeed, let S_{q-1} by the dashed $(q-1)$-square. One can observe that $\pi_1(Z \cap S_q) = \pi_1(Z \cap S_{q-1})$, $\pi_2(Z \cap S_q) = \pi_2(Z \cap S_{q-1})$ and $|Z \cap S_q| = |Z \cap S_{q-1}|$. Therefore if $x' = 2x$, $y' = 2y$ and $s' = 4s$, then $x' = \frac{\pi_1(Z \cap S_{q-1})}{2^{q-1}}$, $y' = \frac{\pi_2(Z \cap S_{q-1})}{2^{q-1}}$ and $s' = \frac{|Z \cap S_{q-1}|}{4^{q-1}}$ and we can apply the recurrence statement with x', y' and s'. Hence $s' \geq f_{q-1}(x')$ and $s' \geq f_{q-1}(y')$. By definition $f_q(x) = \frac{1}{4} f_{q-1}(x')$ and $f_q(y) = \frac{1}{4} f_{q-1}(y')$ and then $s \geq f_{q-1}(x)$ and $s' \geq f_{q-1}(y)$, thus the claimed result holds true in this case.

Let us now consider the case depicted in Figure 1.43(b). We only represent the case where there is a partially completed square (the other cases are solved similarly by considering the last complete square as a partially completed one). When there is a partially completed square, either x or y is strictly larger than $\frac{1}{2}$. We assume here that $x > \frac{1}{2}$ and $y = \frac{1}{2}$. Let S_{q-1} be the partial $(q-1)$-square and S^c_{q-1} be the completed one. Let $x' = \frac{\pi_1(Z \cap S_{q-1})}{2^{q-1}}$ and $s' = \frac{|Z \cap S_{q-1}|}{2^{q-1}}$. Thanks to Lemma 1.33,

$$x = \frac{\pi_1(Z \cap S_q)}{2^q}$$
$$x \leq \frac{\pi_1(Z \cap S_{q-1}) + \pi_1(Z \cap S^c_{q-1})}{2^q}.$$

Because S^c_{q-1} is completed we have

$$x \leq \frac{\pi_1(Z \cap S_{q-1}) + \pi_1(S^c_{q-1})}{2^q},$$
$$x \leq \frac{\pi_1(Z \cap S_{q-1})}{2^q} + \frac{2^{q-1}}{2^q},$$
$$x \leq \frac{1}{2}x' + \frac{1}{2}.$$

Similarly $s = \frac{1}{4} + \frac{1}{4}s'$. In addition, by using the induction hypothesis on x' and s', $f_{q-1}(x') \leq s'$. Hence $f_{q-1}(2(x - \frac{1}{2})) \leq f_{q-1}(x') \leq 4(s - \frac{1}{4})$ what leads to $\frac{1}{4} + \frac{1}{4}f_{q-1}(2(x - \frac{1}{2})) \leq s$ and then $f_q(x) \leq s$. As $x \geq y$, $f_q(y) \leq s$ thanks to Lemma 1.39, thus the claimed result holds true.

The cases depicted in Figure 1.43(c), Figure 1.43(d) and Figure 1.43(e) can be solved easily. In all these cases one can observe that $s \geq \frac{1}{2} \geq \frac{1}{3} \geq f_q(x)$ for all $x \in [0, 1]$. The last inequality is given by Lemma 1.39. Therefore in these cases, the result holds true.

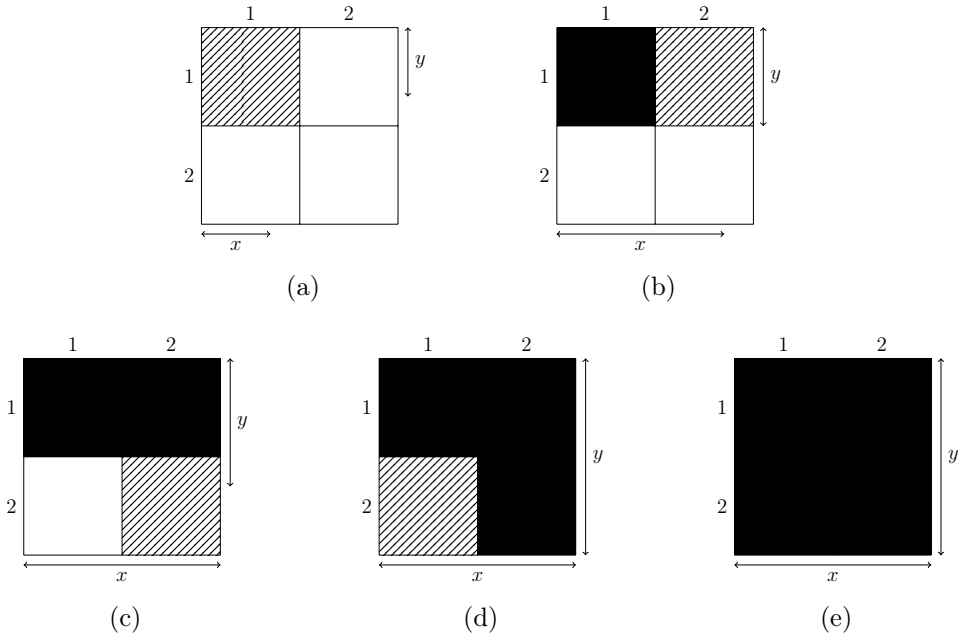


Figure 1.43: Possible cases, completed squares are in black, partially completed one is dashed.

□

An illustration of the intuition behind the recursive aspect of f_q can be seen on Figure 1.44. On the left figure, we are in the first case, *i.e.* $x \leq \frac{1}{2}$. One can observe that the zones to consider are the two dashed ones. We can go even further and prove that we can consider only one of both because we are looking for a lower bound. Therefore, the zone we have to consider is only contained in one of the four sub-square and, after re-scaling x by multiplying it by two, we can have this bound by considering f_{q-1}.

The case where $x > \frac{1}{2}$, is illustrated on the right. For the same reasons, we can consider the lower half of the square only. As $x > \frac{1}{2}$, we know that there must be at least two squares that intersect Z. Using Property 1.1 and the hypothesis stating that another square next to this one intersects Z, we know

that one of them is completed. This is represented on the figure (the black zone is completed) so $s \geq \frac{1}{4}$, and a bound on the area of the zone present in the dashed region can be given by f_{q-1} (after a re-scaling of $x - \frac{1}{2}$).

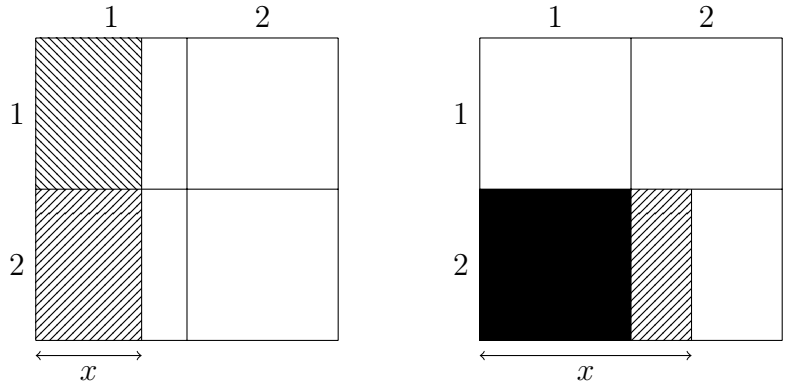


Figure 1.44: A schema to understand the definition of f_q.

The intuition behind $f_0(1) = \frac{1}{3}$ is that $\sum_{k=1}^{+\infty} \frac{1}{4^k} = \frac{1}{3}$. Indeed, if we search for the smallest s such that x can be equal to 1, we notice that this occurs when A is made of a sequence of squares whose area is the quarter of the precedent one. This case is depicted in Figure 1.45. In this case, for a given q, one can observe that $s = \frac{1}{4^q} + \sum_{k=1}^{q} \frac{1}{4^k}$ whose limit, when q increase to infinity, is $\sum_{k=1}^{+\infty} \frac{1}{4^k} = \frac{1}{3}$. In addition, this is the smallest value which ensures that Lemma 1.39 holds true.

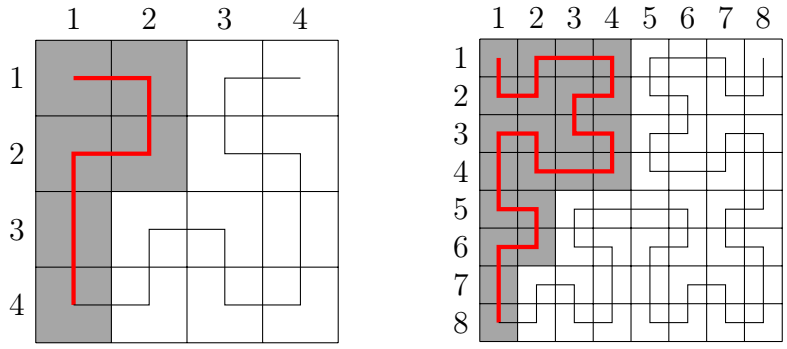


Figure 1.45: The case where s is minimized while x is set to 1.

Thanks to Lemmas 1.39 and 1.40, we know that $s_{i,Z} \geq f_{q_Z}(x_{i,Z}) \geq \frac{1}{3}x_{i,Z}^2$. Therefore

$$\rho_Z \leq \frac{L_Z + x_{1,Z} + x_{2,Z}}{2\sqrt{n_Z + s_{1,Z} + s_{2,Z}}} \leq \frac{L_Z + x_{1,Z} + x_{2,Z}}{2\sqrt{n_Z + \frac{1}{3}(x_{1,Z}^2 + x_{2,Z}^2)}}.$$

Let us now consider the function $g(x,y) = \frac{L_Z+x+y}{2\sqrt{n_Z+\frac{1}{3}(x^2+y^2)}}$. The following lemma states that this function reaches its maximum at $(1,1)$.

Lemma 1.41. *For all $x, y \in [0,1]^2$, $g(x,y) \leq g(1,1)$.*

Proof. Let us define h_y, $y \in [0,1]$ as $h_y(x) = \frac{n_Z+\frac{1}{3}y^2-\frac{1}{3}(L_Z+y)x}{2(n_Z+\frac{1}{3}(x^2+y^2))^{3/2}}$ for $x \in [0,1]$. One can prove that $\frac{\partial g}{\partial x}(x,y) = h_y(x)$ and $\frac{\partial g}{\partial y}(x,y) = h_x(y)$. Easily, we can observe that $h_y(x) \geq 0$ is equivalent to $n_Z + \frac{1}{3}y^2 - (L_Z+y)x \geq 0$. As $x \leq 1$, $n_Z + \frac{1}{3}y^2 - \frac{1}{3}(L_Z+y)x \geq n_Z + \frac{1}{3}y^2 - \frac{1}{3}(L_Z+y)$. Therefore if $\forall y \in [0,1]$, $n_Z + \frac{1}{3}y^2 - \frac{1}{3}(L_Z+y) > 0$, then $\forall y \in [0,1]$, $h_y(x) > 0$. This can be done by considering the polynomial $P(y) = \frac{1}{3}y^2 - \frac{1}{3}y + (n_Z - \frac{L_Z}{3})$, whose discriminant is $\frac{1}{9} - \frac{4}{3}(n_Z - \frac{L_Z}{3})$. Thanks to Lemma 1.38 $L_Z \leq n_Z + 1$. By definition, $n_Z \geq 1$. Therefore $3n_Z \geq L_Z + 1$. This leads to $n_Z - \frac{L_Z}{3} \geq \frac{1}{3}$, what implies $\frac{1}{9} - \frac{4}{3}(n_Z - \frac{L_Z}{3}) \leq -\frac{1}{3}$ and hence $P(y) > 0$. We deduce that for all $x, y \in [0,1]^2$, $h_y(x) > 0$, and so are $\frac{\partial g}{\partial x}(x,y)$ and $\frac{\partial g}{\partial y}(x,y)$. Therefore, for all $y \in [0,1]$, $g(1,y) \geq g(x,y)$ and for all $x \in [0,1]$, $g(x,1) \geq g(x,y)$. So for all $x, y \in [0,1]^2$, $g(x,y) \leq g(1,1)$. □

Thanks to Lemma 1.41,

$$\rho_Z \leq \frac{L_Z + 2}{2\sqrt{n_Z + \frac{2}{3}}}.$$

Let us consider two cases: $n_Z \leq 3$ and $n_Z \geq 4$. In the first case, by using Lemma 1.38, we have $\rho_Z \leq \frac{n_Z+3}{2\sqrt{n_Z+\frac{2}{3}}}$, and, for $n_Z \leq 3$, $\frac{n_Z+3}{2\sqrt{n_Z+\frac{2}{3}}} \leq \frac{3\sqrt{3}}{\sqrt{11}}$.

For the second case, we use a less tight upper bound. We know that $L_Z + x_{1,Z} + x_{2,Z} \leq 6$ and $2\sqrt{n_Z + \frac{1}{3}(x^2+y^2)} \geq 2\sqrt{n_Z}$. Therefore $\rho_Z \leq \frac{3}{\sqrt{n_Z}}$ and for $n_Z \geq 4$, $\frac{3}{\sqrt{n_Z}} \leq \frac{3}{2} \leq \frac{3\sqrt{3}}{\sqrt{11}}$.

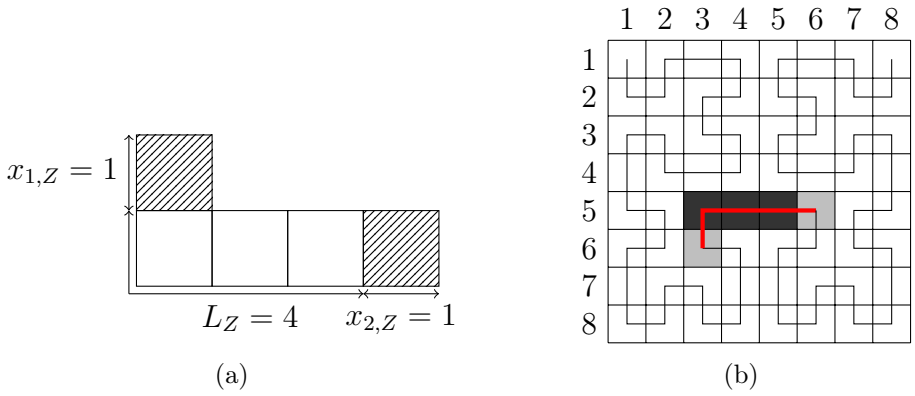
So, in any case, if $Z_1, \ldots, Z_p$ is the partition of $[1,N]^2$ given by SFCP, then for all $k \in [1,p]$, $\rho_{Z_k} \leq \frac{3\sqrt{3}}{\sqrt{11}}$ what achieves the proof of Theorem 1.35.

□

Note that for an individual zone, the worst case is $n_Z = 3$ and $L_Z = 4$ (Figure 1.46(a)) and it can occur, notably in the case of Hilbert' function, as it is shown on Figure 1.46(b).

1.5.3 Complexity

The computation of Hilbert's curve has been a problem studied for a long time, see Butz [1971]; Breinholt and Schierz [1998] for example. We use the implementation from Skilling *et al.* [2004] that provides for given i, j $H^{-1}(i,j)$ in $O(d \log N)$ where d is the dimension ($d = 2$ here).



(a) (b)

Figure 1.46: Worst case for an individual zone during the execution of SFCP. Dashed squares are partially completed.

Therefore with a single pre-computation in $O(m)$ to precompute for each processor the portion of the image of H that is attributed to it (to compare with the $O(m^2 \log m)$ of ColumnBased and the $O(m \log m)$ of NRRP and associated), we can allocate the whole square with a $O(N^2 \log N \log m)$ complexity (versus $O(N^2)$ for the others).

1.6 Conclusion and Perspectives

In this chapter we presented, in Section 1.1, PERI-SUM, the square partitioning problem we studied all along, and how it can be used to provide communication avoiding matrix multiplication. Then, in Section 1.2, we explored the pre-existing studies on this particular problem: lower bound, NP-Completeness, approximation algorithms (ColumnBased and RRP), optimal solution for particular cases and dynamic strategy (MinCost). In Section 1.3 we justified the search of new approximation algorithm by showing that if static strategies are not always reliable in practise they can be used as bases for efficient and cost-effective hybrid strategies. Next, in Section 1.4, we presented our main result: the algorithm NRRP, an approximation algorithm with the best-known ratio for this problem ($\frac{2}{\sqrt{3}} \simeq 1.15$). We also proposed a weaker variant, SNRRP, and discussed possible improvements for divide-and-conquer algorithms to solve PERI-SUM. Finally, in Section 1.5, we presented SFCP, a space-filling-curve-based algorithms that is the only known algorithm designed for the discrete variant of PERI-SUM with guaranteed approximation ratio.

If the current work on PERI-SUM and its close variants appears to be consistent, there are still many possible additional problems to work on.

For example, mostly with a theoretical point of view, NRRP can probably be improved. Let us recall that initially RRP (and SNRRP) aim at splitting rectangles into rectangles with aspect ratios below 3. To design NRRP we

change the aimed aspect ratio to a smaller value $\frac{5}{2}$. However this value, denoted μ can be lowered again (down to 2) but it requires additional subcases (see NRRP in comparison to RRP or SNRRP). By pushing this splitting logic, we think we can design a $(\alpha + \epsilon)$-approximation for every strictly positive ϵ. The aimed α could be $\frac{3}{2\sqrt{2}} \simeq 1.06$, the ratio between half-perimeter of a rectangle with aspect ratio 2 and the lower bound of its half-perimeter. Note that this is just an intuition, the new designed subcases might imply a larger constant ratio.

Another possible improvement of PERI-SUM can be a task and communication heterogeneous version. More precisely, in this chapter (and also in the whole thesis), each task $T_{i,j,k}$ has the same computation time and each data ($A_{i,k}$, $B_{k,j}$ or $C_{i,j}$) has the same size. We can also consider a problem without such assumption (we focus on the bi-dimensional version, omitting the k).

Problem 1.3 (PERI-SUM-HETEROGENEOUS)**.** Let $T = \{t_{i,j},\ (i,j) \in [1,N]^2\}$ be a set of tasks, w be a weight function from T to $\mathbb{R}$ and w_x, w_y be weight functions from $[1,N]$ to $\mathbb{R}$. Given a repartition $\{s_1, \ldots, s_m\}$ such that $\sum s_i = w(T)$, find for each s_i a zone $Z_k \subset T$ such that $w(Z_k) = s_k$ and $\bigcup Z_k = T$ and minimizing

$$\sum_k \left(\sum_{i,\exists j, t_{i,j} \in Z_k} w_x(i) + \sum_{j,\exists i, t_{i,j} \in Z_k} w_y(j) \right).$$

Note that, in its more general form, the building of a repartition that achieves minimal makespan, without communication consideration, is itself a NP-complete problem (equivalent to Bin-Packing problem).

The first motivation to study PERI-SUM-HETEROGENEOUS is the sparse matrix-multiplication. If we set the weights to 0 or 1 (1 if this is a non-zero element or data), we have a model of sparse matrix multiplication (or the outer product of sparse-vectors, in the purely 2D-case). The problem of communication avoiding sparse-matrix multiplication has already been considered but, to the best of our knowledge, not with this model. For example Demmel *et al.* [2008] propose a solution based on kernel reorganisation of the matrices.

By using non binary weights, we can also extend the model to low rank-matrices. More precisely, low-rank matrix approximation is a method to represent a matrix M (with a small approximation and possible loss of information) as the product of two matrices, the first M_r with few columns, the second M_c with few rows. In addition to the possible gain in storage cost, low-rank matrix approximation also improves matrix multiplication. Instead of directly multiplying two matrices A and B, the product $A_c \times B_r$ is computed first, creating a intermediate matrix with few row and columns that is then multiplied with A_r and B_c. If we now suppose that we multiply two block-matrices where each block uses low-rank representation, then we can use PERI-SUM-HETEROGENEOUS as a model for communication-avoiding parallel matrix

multiplication by linking w, w_x and w_y to the sizes of the different $A_{i,j}$ and $B_{k,j}$ (each depending of their rank).

Another motivation to study PERI-SUM-HETEROGENEOUS is FMM (Fast Multipole Method), an algorithm introduced in Greengard and Rokhlin [1987] and considered as one of the most important algorithms of the last century, Dongarra and Sullivan [2000]. FMM is an algorithm to solve N-body interactions. These N particles are in a space, with possibly heterogeneous placement (possible locally empty or dense spaces). In order to efficently compute the interaction force applied on each particle, the space is divided into zones. Inside a zone, the interactions are fully computed (each particle interacts with all the others). Then each zone computes the interaction force of its particle with the ones of each neighbour zone. After these two phases, zones are approximated to one particle each (whose charge depends on the ones in this zone), the space is split again into zones and the procedure continues recursively as before until there is only one zone.

The parallel computation of FMM is an important challenge, see Agullo *et al.* [2016a] for an example. However the link with PERI-SUM-HETEROGENEOUS is not direct. If we represent the space by a square and particles by their coordinates in this square, we have something close to an input of PERI-SUM-HETEROGENEOUS with weight being 0 if there is no particle at this coordinate, 1 otherwise. An important part of the communication between processors during parallel FMM comes from the computations of interactions between particles of a zone and the ones of its neighbours. These interactions, for a given zone, are correlated to its number of particles (and the one of its neighbours) and its number of neighbours. In order to avoid having too many neighbours, having compact zones with low perimeter is important. Therefore, if PERI-SUM-HETEROGENEOUS is currently not a perfect model of parallel FMM, a solution to this problem might be a solution to avoid communication.

To the best of our knowledge, PERI-SUM-HETEROGENEOUS has not really been studied. The problem being strictly harder than PERI-SUM we can already state it is NP-complete. One possible algorithm is SFCP, presented in Section 1.5, whose adaptation is easy and already in use, Deveci *et al.* [2016]. However there is no real study on the efficiency of SFCP for this particular problem. In addition, SFCP has disappointing results for solving PERI-SUM in comparison to ColumnBased or NRRP and there are chance that there exist better algorithms than SFCP to solve PERI-SUM-HETEROGENEOUS. However they are still to be designed.

In Chapter 2, we propose to go back to the study of communication-avoiding matrix multiplication. For this purpose, we propose to study MSCubeP, the direct 3D-transposition of PERI-SUM, the partitioning of a cube into polyhedra.

Chapter 2

Cube Partitioning for Communication-Avoiding Parallel Matrix Multiplication

2.1 Introduction

As stated in Chapter 1, the standard matrix multiplication $C = A \times B$ can be seen as a set of N^3 tasks (for square matrices of size N)

$$T_{i,j,k} : C_{i,j} \leftarrow C_{i,j} + A_{i,k}B_{k,j}$$

for $\{i, j, k\} \in [1, N]^3$. As stated before, our goal here is to avoid as much as possible the replication of the $A_{i,k}$'s, $B_{k,j}$'s and $C_{i,j}$'s, under the assumption that we aim at achieving optimal makespan.

In Chapter 1, we schedule this set of tasks by bags, one value k after another. In this chapter we propose to schedule the whole set at once, adding a dimension to the previous problem. To simplify the model, we consider equally the $A_{i,k}$'s, the $B_{k,j}$'s and the $C_{i,j}$'s, even if these last ones are, in practice, read-write data and not read-only data. In this case we end up with a 3-dimension problem, instead of a 2-dimension one, with a cube partitioning problem MSCubeP. We define the following notations. Let Z be a zone included into a cube Cu. We define by $v(Z)$ its volume. Let $\Pi_1(Z) = \{(x, z),\ \exists y,\ (x, y, z) \in Z\}$, $\Pi_2(Z) = \{(y, z),\ \exists x,\ (x, y, z) \in Z\}$ and $\Pi_3(Z) = \{(x, y),\ \exists z,\ (x, y, z) \in Z\}$ be the projections of Z on the three faces and $\pi_1(Z) = |\Pi_1(Z)|$, $\pi_2(Z) = |\Pi_2(Z)|$ and $\pi_3(Z) = |\Pi_3(Z)|$ be their sizes. Then we define the half-covering surface of Z as $Hs(Z) = \pi_1(Z) + \pi_2(Z) + \pi_3(Z)$.

Problem 2.1 (Minimizing-Surface-Cube-Partition (MSCubeP))**.** Given a set of m rational numbers $\{v_1, \ldots, v_m\}$ such that $\sum v_k = 1$, and the cube $Cu = [0, 1] \times [0, 1] \times [0, 1]$, find a set of zones Z_k of Cu such that $v(Z_k) = v_k$ and $\bigcup Z_k = Cu$, so as to minimize $\sum Hs(Z_k)$.

For the particular case of continuous zones, *i.e.* polyhedrons (that is addressed in Section 2.4), we also define the notion of covering cuboid and denote it by $Cu(P)$, with P being a polyhedron. $Cu(P)$ is the smallest cuboid such that $P \subseteq Cu(P)$. If $Cu(P) = [x_1, x_2] \times [y_1, y_2] \times [z_1, z_2]$, then let us define the width of P by $w(P) = x_2 - x_1$, the height of P by $h(P) = y_2 - y_1$, and the depth of P by $d(P) = z_2 - z_1$. Note that if P is connected, then $\pi_1(P) = w(P) \times d(P)$, $\pi_2(P) = h(P) \times d(P)$ and $\pi_3(P) = w(P) \times h(P)$. In this case we also define $\rho(P) = \frac{\max(w(P),h(P),d(P))}{\min(w(P),h(P),d(P))}$ its aspect ratio and $\rho'(P) = \frac{\max(w(P),h(P),d(P))}{\text{med}(w(P),h(P),d(P))}$ its secondary aspect ratio (where $\text{med}(x, y, z)$ is the median value between x, y and z).

Later, particularly in Section 2.3, we consider a variant of MSCubeP: MSCuboidP. In MSCuboidP, instead of a cube, a cuboid is to be partitioned.

Problem 2.2 (Minimizing-Surface-Cuboid-Partition (MSCuboidP))**.** Given a cuboid $Cu = [0, x] \times [0, y] \times [0, z]$ and a set of m rational numbers $\{v_1, \ldots, v_m\}$ such that $\sum v_k = xyz$, find a set of zones Z_k of Cu such that $v(Z_k) = v_k$ and $\bigcup Z_k = Cu$, so as to minimize $\sum Hs(Z_k)$.

Finally, note that, for the same reason as for PERI-SUM (see Ballard *et al.* [2011b]), the best theoretical solution would be that all zones are cubes. In this case, for all k, $Hs(Z_k) \geq 3v_k^{\frac{2}{3}}$ and thus

$$C_{opt} \geq 3\sum_k v_k^{\frac{2}{3}}, \tag{2.1}$$

where C_{opt} denotes the optimal solution for a given instance of MSCubeP.

2.2 Related Work

For the general problem of parallel matrix multiplication we rely on the related work already presented in Section 1.2. For MSCubeP we found almost no study. The only pre-existing algorithm we found, SCR, is presented below. However, solutions for PERI-SUM can be adapted to solve MSCubeP, this will be developed in Section 2.6. In addition, we present in Section 2.2.2 a sketch of the proof that PERI-SUM is NP-complete which is close to the one we propose in Section 2.3 to prove the NP-completeness of MSCuboidP, a variant of MSCubeP.

2.2.1 SCR (Slice-Column-Row)

SCR (for Slice-Column-Row) is a dynamic programming algorithm introduced by Zarei Zefreh *et al.* [2016] and that is the direct 3-dimension transposition of ColumnBased (see Section 1.2.2) used to solve PERI-SUM. Like ColumnBased,

SCR makes an assumption on the partitioning. SCR assumes that there is a first splitting of the cube into slices and then use ColumnBased to compute the internal splitting of each slice. More precisely let $\{v_1, \ldots, v_m\}$ be an instance of MSCubeP. SCR computes the function $f : [1, m] \times [1, m] \to \mathbb{R}$ such that $f(S, q)$ is the cost of the best partition into slices-columns of the instance $\{v_1, \ldots, v_q\}$ with S slices. In practice

$$f(S,q) = \begin{cases} 1 + \left(\sum_{1 \leq k \leq q} v_k\right) ColumnBased(\{v_1^N, \ldots, v_q^N\}) \text{ if } S = 1 \\ \min_{1 \leq r \leq q-S+1} 1 + f(S-1, q-r) + \left(\sum_{q-r \leq k \leq q} v_k\right) ColumnBased(\{v_{q-r}^N, \ldots, v_q^N\}) \text{ otherwise} \end{cases}$$

where v_k^N is the normalized value of v_k (see Algorithm 2.1 for a formal description of SCR). Like ColumnBased, SCR uses a function f_{cut} in order to compute the number of cuboids per slices (here $f_S^{cut}(q)$ is the number of cuboids in the $S-1$ first slices in a partition with S slices and the q first v_k). The complete partitioning of the unitary cube can then be obtained by applying ColumnBased on each of the S slices (S being the number of slice computed by SCR) with knowledge on the number of cuboids in each slide (the k_is computed by SCR).

Note that ColumnBased, that also relies on dynamic programming, has a computation time of $O(p^2 \log p)$. Thus the naive computation time of SCR, if we directly translate above equation, would be $O(m^5 \log m)$. However, by pre-computing all the $ColumnBased((\{v_k^N, \ldots, v_{k'}^N\})$ for $k, k' \in [1, m]^2$ we can reduce this complexity to $O(m^3 \log m)$. Note also that, unlike for ColumnBased, there is currently no proof that sorting the v_k in increasing order improves the efficiency of SCR and thus no guarantee that SCR outputs the best slice-column partitioning.

2.2.2 NP-Completeness Proof of PERI-SUM

In Section 2.3 we prove the NP-completeness of the decision problem associated to MSCuboidP. This proof is inspired from the one existing for PERI-SUM (Beaumont *et al.* [2002]), and this is why we propose here a survey of this proof to help the reader understand the proof in Section 2.3. First note that PERI-SUM is an optimisation problem and the problem we are considering is therefore PERI-SUM-DEC.

Problem 2.3 (PERI-SUM-DEC)**.** Given a set of m strictly positive rational numbers $\{s_1, \ldots, s_m\}$ such that $\sum s_k = 1$, the square $S = [0, 1] \times [0, 1]$ and a rational K, is there a set of zones $\{Z_1, \ldots, Z_m\}$ such that for each k the area of Z_k is s_k, $\bigcup Z_k = S$, and $\sum p(Z_k) \leq K$?

Algorithm 2.1 : SCR $(\{v_1, \ldots, v_m\})$

Input : A set of positive values $\{v_1, \ldots, v_m\}$ such that $\sum v_i = 1$ and $v_1 \leq v_2 \leq \ldots \leq v_m$

Output : A number of slices S and $\{k_1, \ldots, k_S\}$ the number of cuboid in each slice to partition the unary cube.

for $q = 1$ **to** m **do**

$f_1(q) = 1 + \left(\sum_{1 \leq k \leq q} v_k \right) ColumnBased(\{v_1^N, \ldots, v_q^N\})$;

$f_1^{cut}(q) = 0$;

for $S = 2$ **to** m **do**

for $q = S$ **to** m **do**

$f_S(q) = \min_{1 \leq r \leq q-S+1} 1 + f(S-1, q-r) +$

$\left(\sum_{q-r \leq k \leq q} v_k \right) ColumnBased(\{v_{q-r}^N, \ldots, v_q^N\})$;

$f_S^{cut}(q) = q - r_{min}$;

$q = m$;

$S_{min} = S$ such that $f_S(m) = \min_{1 \leq S \leq m} f_S(m)$;

for $S = S_{min}$ **downto** 2 **do**

$k_S = q - f_S^{cut}(q)$;

$q = f_S^{cut}(q)$;

$k_1 = q$;

return $S_{min}, \{k_1, \ldots, k_{S_{min}}\}$;

First let us consider 2-PART-EQUAL, a variant of 2-PART, the classical partition problem.

Problem 2.4 (2-PART)**.** Given a set of n strictly positive rational numbers $\{a_1, \ldots, a_n\}$, is there $I \subseteq [1, n]$ such that

$$\sum_{i \in I} a_i = \sum_{i \notin I} a_i ?$$

Problem 2.5 (2-PART-EQUAL)**.** Given a set of $2n$ strictly positive rational numbers $\{a_1, \ldots, a_{2n}\}$, is there $I \subseteq [1, 2n]$ such that $|I| = n$ and

$$\sum_{i \in I} a_i = \sum_{i \notin I} a_i ?$$

Lemma 2.1. *2-PART-EQUAL is NP-complete.*

Proof. Trivially 2-PART-EQUAL is in NP (checking that the two sums are equal can be done in polynomial time). For the NP-completeness proof we

propose a reduction from 2-PART, that is known to be NP-complete, Garey and Johnson [2002]. Let $\{a_1, \ldots, b_n\}$ be an instance of 2-PART. Let C be an integer ($C > 0$) and let $\{b_1, \ldots, b_{2n}\}$ be defined as follows,

$$b_i = \begin{cases} a_i + C & \text{if } i \leq n \\ C & \text{otherwise.} \end{cases}$$

We now prove that there exists a subset $I \subseteq [1, n]$ such that $\sum_{i \in I} a_i = \sum_{i \notin I} a_i$ if and only there is a subset $I' \subseteq [1, 2n]$ such that $|I'| = n$ and $\sum_{i \in I'} b_i = \sum_{i \notin I'} b_i$.

Let us assume there is $I \subseteq [1, n]$ such that $\sum_{i \in I} a_i = \sum_{i \notin I} a_i$. Let $I' = I \cup \{b_{n+1}, \ldots, b_{n+n-|I|}\}$. Then,

$$\begin{aligned} \sum_{i \in I'} b_i &= \sum_{i \in I} b_i + \sum_{i=n+1}^{2n-|I|} b_i \\ &= \sum_{i \in I} (a_i + C) + \sum_{i=n+1}^{2n-|I|} C \\ &= \sum_{i \in I} a_i + |I|C + (n - |I|)C \\ &= \sum_{i \in I} a_i + nC \end{aligned}$$

and

$$\begin{aligned} \sum_{i \notin I'} b_i &= \sum_{i \notin I} b_i + \sum_{i=2n-|I|+1}^{2n} b_i \\ &= \sum_{i \notin I} (a_i + C) + \sum_{i=2n-|I|+1}^{2n} C \\ &= \sum_{i \notin I} a_i + (n - |I|)C + |I|C \\ &= \sum_{i \in I} a_i + nC \\ &= \sum_{i \in I'} b_i, \end{aligned}$$

what achieves the proof in this case.

Let us now assume that there is a subset $I' \subseteq [1, 2n]$ such that $|I'| = n$ and $\sum_{i \in I'} b_i = \sum_{i \notin I'} b_i$. Let $I = I' \cap [1, n]$. Then,

$$\begin{aligned}
\sum_{i \in I} a_i &= \sum_{i \in I} (b_i - C) \\
&= \sum_{i \in I} b_i - |I|C \\
&= \sum_{i \in I'} b_i - \sum_{i \in I' \setminus I} b_i - |I|C \\
&= \sum_{i \in I'} b_i - |I' \setminus I|C - |I|C \\
&= \sum_{i \in I'} b_i - |I'|C \\
&= \sum_{i \in I'} b_i - nC
\end{aligned}$$

and

$$\begin{aligned}
\sum_{i \notin I} a_i &= \sum_{i \in [1,n] \setminus I} (b_i - C) \\
&= \sum_{i \in [1,n] \setminus I} b_i - (n - |I|)C \\
&= \sum_{i \notin I'} b_i - \sum_{i \notin [n+1,2n] \cap I'} b_i - (n - |I|)C \\
&= \sum_{i \notin I'} b_i - (n - |I' \setminus I|)C - (n - |I|)C \\
&= \sum_{i \in I'} b_i - (n - |I'| + |I|)C - (n - |I|)C \\
&= \sum_{i \in I'} b_i - nC \\
&= \sum_{i \in I} a_i
\end{aligned}$$

what achieves the proof. □

Both NP-Completeness proofs, the ones for PERI-SUM and MSCuboidP, rely on a reduction from 2-PART-EQUAL. More precisely, the proof of NP-Completeness is a reduction from 2-PART-EQUAL to a variant of PERI-SUM, PARTITION-ALL-SQUARES, where all zones are forced to be squares (the ideal partition).

Problem 2.6 (PARTITION-ALL-SQUARES)**.** Given a set of m strictly positive rational numbers $\{l_1, \ldots, l_m\}$ such that $\sum l_k^2 = 1$, and the square $S = [0,1] \times [0,1]$, is there a set of squares $\{S_1, \ldots, S_m\}$ such that for each k the surface of S_k is l_k^2 and $\bigcup S_k = S$?

Lemma 2.2 (Beaumont *et al.* [2002])**.** *PARTITION-ALL-SQUARES is NP-complete.*

Proof. Let us now consider an instance $\{a_1, \ldots, a_n\}$ of 2-PART-EQUAL. The first step of this proof is to transform this instance into a second one, $\{b_1, \ldots, b_n\}$, such that $\frac{M}{2} < b_i \leq \frac{3M}{4}$ where $M = \frac{4}{3} \max_i b_i$. The utility of this inequality will be developed later.

We define $\{b_1, \ldots, b_n\}$ as: $\forall i,\ b_i = 2(a_i + 2n \max_j a_j)$. Let $N = \max_i b_i = 2(2n+1) \max_i a_i = \frac{3M}{4}$. Thus

$$\forall\ i,\ b_i > 4n \max_j a_j = \frac{4n}{4n+2} N > \frac{2}{3} N = \frac{M}{2}.$$

Therefore $\frac{M}{2} < b_i \leq \frac{3M}{4}$. Let us now prove that there is a solution of 2-PART-EQUAL with the input $\{a_1, \ldots, a_n\}$ if and only if there is a solution with the input $\{b_1, \ldots, b_n\}$.

Let us first suppose that there exists I such that $|I| = \frac{n}{2}$ and $\sum_{i \in I} a_i = \sum_{i \notin I} a_i$. In this case,

$$\begin{aligned}
\sum_{i \in I} b_i &= \sum_{i \in I} 2(a_i + 2n \max_j a_j) \\
&= 2 \sum_{i \in I} a_i + |I| 4n \max_j a_j \\
&= 2 \sum_{i \in I} a_i + 2n^2 \max_j a_j
\end{aligned}$$

and

$$\begin{aligned}
\sum_{i \notin I} b_i &= \sum_{i \notin I} 2(a_i + 2n \max_j a_j) \\
&= 2 \sum_{i \notin I} a_i + |[1,n] \setminus I| 4n \max_j a_j \\
&= 2 \sum_{i \in I} a_i + (n - |I|) 4n \max_j a_j \\
&= 2 \sum_{i \in I} a_i + 2n^2 \max_j a_j \\
&= \sum_{i \in I} b_i.
\end{aligned}$$

Otherwise let us assume there is I such that $|I| = \frac{n}{2}$ and $\sum_{i\in I} b_i = \sum_{i\notin I} b_i$. In this case

$$\begin{aligned}\sum_{i\in I} a_i &= \sum_{i\in I}(\frac{b_i}{2} - 2n \max_j a_j)\\ &= \frac{1}{2}\sum_{i\in I} b_i - |I|2n \max_j a_j\\ &= \frac{1}{2}\sum_{i\in I} b_i - n^2 \max_j a_j\end{aligned}$$

and

$$\begin{aligned}\sum_{i\notin I} a_i &= \sum_{i\notin I}(\frac{b_i}{2} - 2n \max_j a_j)\\ &= \frac{1}{2}\sum_{i\notin I} b_i - (n-|I|)2n \max_j a_j\\ &= \frac{1}{2}\sum_{i\in I} b_i - n^2 \max_j a_j\\ &= \sum_{i\in I} a_i.\end{aligned}$$

Thus there exists a solution of 2-PART-EQUAL with input $\{a_1, \ldots, a_n\}$ if and only if there exists a solution with input $\{b_1, \ldots, b_n\}$.

To achieve the proof, we propose a reduction from the input $\{b_1, \ldots, b_n\}$ to an instance of PARTITION-ALL-SQUARES. For this we need an additional tool: the Kenyon' squares. For each i, we want to tile a rectangle R_i of size $b_i \times (M - b_i)$ with a polynomial number of squares. Kenyon [1996] proves that this can be done with a logarithmic number of squares if the rectangle is not too elongated (what is ensured by the inequality $\frac{M}{2} < b_i \leq \frac{3M}{4}$). We denote $KS(i)$ the number of Kenyon' squares to tile R_i and $w(b_i, j)$, for $1 \leq j \leq KS(i)$ the width of the squares used for this tiling. Let $l = 20S + 17M$, with $S = \frac{1}{2}\sum b_i$. We are now ready to describe the instance of PARTITION-ALL-SQUARES: it consists of $14 + n + \sum_{i=1}^{n} KS(i)$ squares of width

$\frac{13S+11M}{l}$ $(\times 1)$, $\frac{7S+6M}{l}$ $(\times 3)$,
$\frac{3S+2M}{l}$ $(\times 2)$, $\frac{2S+2M}{l}$ $(\times 4)$,
$\frac{4S+3M}{l}$ $(\times 2)$, $\frac{3S+3M}{l}$ $(\times 2)$,
$\frac{b_i}{l}$, $\forall i$, $\frac{w(b_i,j)}{l}$, $\forall i$ and $\forall 1 \leq j \leq KS(i)$,

which should be used to tile the unit square.

In the following we denote by $A_{x,y}$ a square of width $\frac{xS+yM}{l}$, by A_{b_i} a square of width b_i and by $A_{b_i,j}$ a square of width $w(b_i, j)$.

In any case, the tiling presented on Figure 2.1 is valid. The following step is to tile the remaining two rectangles with the remaining squares, *i.e.* the A_{b_i}s and the $A_{b_i,j}$s.

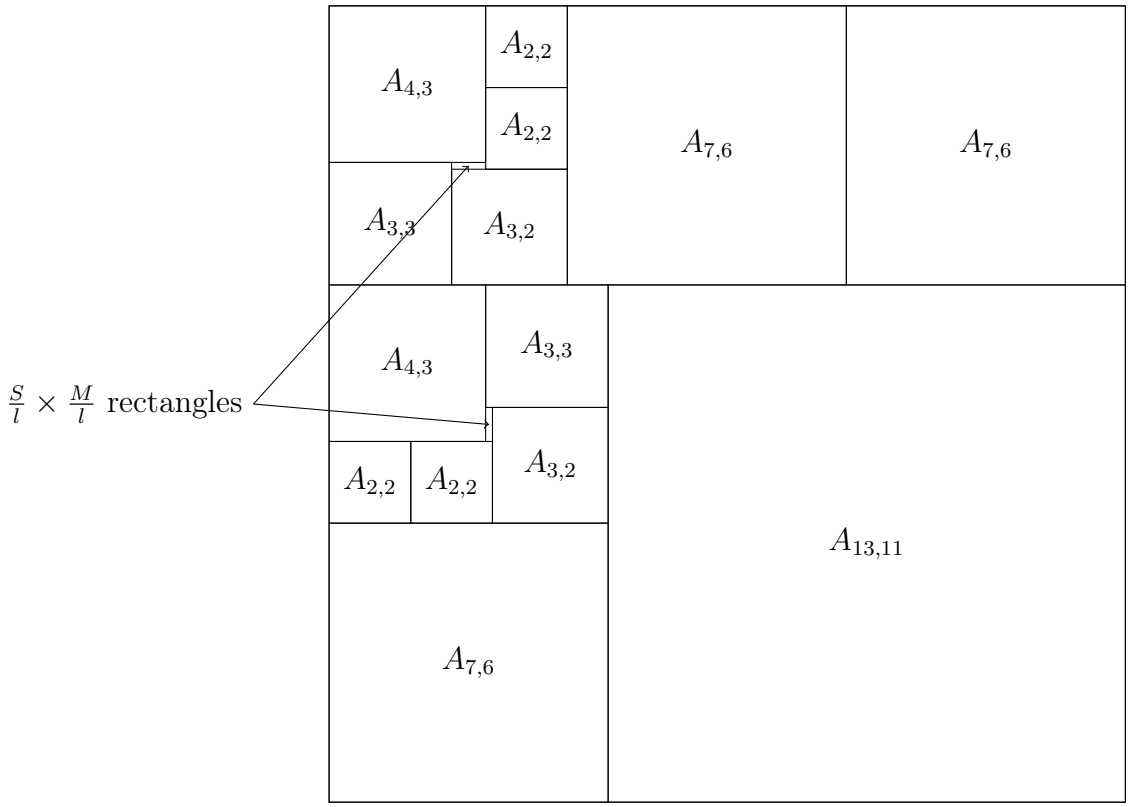


Figure 2.1: Tilling of the unitary square rectangle with the $A_{x,y}$s, the A_{b_i}s and the $A_{b_i,j}$s.

When there is a solution for instance $\{b_1, \ldots, b_n\}$ of 2-PART-EQUAL, we propose the tiling depicted in Figure 2.2 for the two $\frac{S}{l} \times \frac{M}{l}$ rectangles. First, for a given i, the $A_{b_i,j}$s can tile a $\frac{b_i}{l} \times \frac{M-b_i}{l}$ rectangle. Thus, for a given i, the $A_{b_i,j}$s put together with A_{b_i} can tile a $\frac{b_i}{l} \times \frac{M}{l}$ rectangle. Thus, with half of the A_{b_i}s put side by side and by tiling the remaining area by the associated $A_{b_i,j}$s we can tile one of the $\frac{S}{l} \times \frac{M}{l}$ rectangle. This tiling is possible because we know that there exists I such that $\sum_{i \in I} b_i = \sum_{i \notin I} b_i = \frac{S}{2}$. Thus, if there exists a solution for the instance $\{b_1, \ldots, b_n\}$ of 2-PART-EQUAL, then there exists a tiling of the unitary square with our instance of PARTITION-ALL-SQUARES.

The counter-part is a bit more complex to prove. We skip this part of the proof here (see Beaumont *et al.* [2002]). Globally, the idea is to prove that if there is a tiling, the tiling is as in Figure 2.1 (under the assumption than S is significantly greater than M, assumption that is fulfilled for n large enough, for example $n \geq 400$). Similarly, the tiling of the $\frac{S}{l} \times \frac{M}{l}$ can only be made the

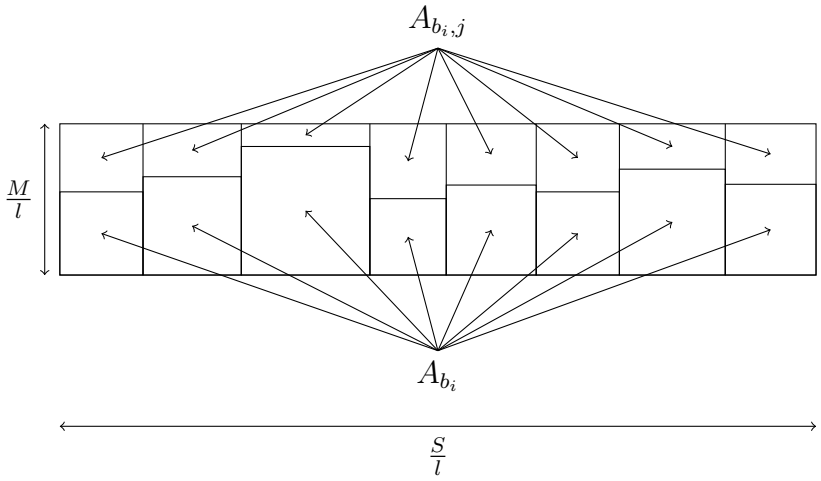


Figure 2.2: Tilling of a $\frac{M}{l} \times \frac{S}{l}$ rectangle with some A_{b_i}s and $A_{b_i,j}$s.

way described on Figure 2.2, notably because $\forall i$, $b_i > \frac{M}{2}$ and two A_{b_i} cannot lie one on top of the other. □

With Lemma 2.2, we achieve the proof of NP-completeness of PERI-SUM-DEC.

Theorem 2.3 (Beaumont *et al.* [2002])**.** *PERI-SUM-DEC is NP-complete.*

Proof. Let $\{l_1, \ldots, l_m\}$ be an instance of PARTITION-ALL-SQUARES. Let $\{s_1, \ldots, s_m\}$ be defined as $s_i = l_i^2$ for all i and $K = \sum_{i=1}^{m} 2l_i$. $(\{s_1, \ldots, s_m\}, K)$ is a valid instance of PERI-SUM-DEC. We know, thanks to equation (2.1), that any valid tiling of the unitary square with zones of surface $s_1, \ldots, s_m$ has a half-perimeter larger than $\sum_{i=1}^{m} 2\sqrt{s_i} = \sum_{i=1}^{m} 2l_i = K$ and this case only happens where all zones are square. Therefore there exists a solution of PERI-SUM with instance $(\{s_1, \ldots, s_m\}, K)$ if and only if there exists a solution for PARTITION-ALL-SQUARES with instance $\{l_1, \ldots, l_m\}$. Thus we have a reduction from PARTITION-ALL-SQUARES to PERI-SUM-DEC what achieves the proof. □

2.3 NP-Completeness of MSCuboidP

In this section we are interested in the NP-Completeness of MSCuboidP. Note that the NP-completeness of MSCuboidP does not imply the NP-Completeness of MSCubeP (the NP-completeness of MSCubeP would imply the NP-completeness of MSCuboidP), that is still an open problem.

MSCuboidP is not a decision problem, so, like for PERI-SUM, we are considering the decision problem associated to MSCuboidP, MSCuboidP-DEC.

Problem 2.7 (MSCuboidP-DEC)**.** Given a cuboid $Cu = [0, x] \times [0, y] \times [0, z]$ and a set of m rational numbers $\{v_1, \ldots, v_m\}$ such that $\sum v_k = xyz$ and a number K, is there a set of m zones Z_k of Cu such that $V(Z_k) = v_k$, $\bigcup Z_k = Cu$ and $\sum Hs(Z_k) \leq K$?

We start by reducing this problem to a more constrained variant named ACCuboidP, in which the goal is to partition the cuboid in cubes of specified side lengths. ACCuboidP is of course the 3D-counterpart of PARTITION-ALL-SQUARES.

Problem 2.8 (All-Cube-Cuboid-Partition (ACCuboidP))**.** Given a set of m given rational numbers $\{l_1, \ldots, l_m\}$ such that $\sum l_k^3 = xyz$, and a cuboid $Cu = [0, x] \times [0, y] \times [0, z]$, is there a set of k cubes $C_k \in Cu$ such that $V(C_k) = l_k^3$ and $\bigcup C_k = Cu$?

Lemma 2.4. *ACCuboidP is NP-Complete.*

Proof. It is easy to check that ACCuboidP belongs to NP. We prove NP-hardiness of ACCuboidP with a method inspired from the hardiness proof of the equivalent 2D problem (see Section 2.2.2), by using a reduction from 2-PART-EQUAL (the partitioning of a set of values into two sets of equal size and equal sum), which is NP-complete according to Lemma 2.1. Our proof consists of two steps: from an instance of 2-PART-EQUAL, we first derive another set of numbers b_i and prove that they can be partitioned into two equal sets if and only if the 2-PART-EQUAL instance has a solution. Then, we use the b_i numbers to build an instance of ACCuboidP for which the existence of a packing is equivalent to partitioning the b_i into two equal size sets.

First Reduction

Let us now consider an instance of 2-PART-EQUAL, $\{a_1, \ldots, a_{2n}\}$ and let us denote $2A = \sum_{1 \leq i \leq 2n} a_i$ and $M = 6n \times \max_i a_i$. Let us suppose, without loss of generality, that n is a multiple of 120 larger than 240 and let us define a new set $\{b_1, \ldots, b_{2n}\}$ as

$$\forall i,\ b_i = a_i + 3n \times \max_i a_i + D \qquad \text{where } D = \frac{60M - (A \bmod 60M)}{n}$$
$$= a_i + \frac{M}{2} + D.$$

In addition, let us set $k = \frac{n}{120} + \frac{A+nD}{60M}$ and $S = \frac{1}{2} \sum_{1 \leq i \leq 2n} b_i$. One can prove that k is an integer (since n is a multiple of 120) and that $S = 60k \times M$. In addition, let us notice that for all i, $\frac{M}{2} < b_i$ (since $D \geq 0$ and $a_i > 0$) and

$b_i \leq M$. Indeed, $D \leq \frac{60M}{n} \leq \frac{M}{4}$ and $a_i + \frac{M}{2} \leq M(\frac{1}{2} + \frac{1}{6n}) \leq \frac{4M}{6}$. Therefore $b_i \leq \frac{11M}{12} \leq M$.

Let us now prove that there exists a solution to the instance of 2-PART-EQUAL if and only if there exists a set $I \subset [1, n]$ such that $\sum_{i \in I} b_i = \sum_{i \notin I} b_i$. If there exists I such that $|I| = n$ and $\sum_{i \in I} a_i = \sum_{i \notin I} a_i$, then

$$\sum_{i \in I} b_i - \sum_{i \notin I} b_i = \sum_{i \in I} a_i - \sum_{i \notin I} a_i + (|I| - |\overline{I}|)(\frac{M}{2} + D)$$
$$\sum_{i \notin I} a_i - \sum_{i \in I} a_i = (|I| - |\overline{I}|)(\frac{M}{2} + D).$$

Yet, $\sum_{i \notin I} a_i - \sum_{i \in I} a_i \leq 2n \times max\ a_i = \frac{M}{3}$ and $\frac{M}{2} \leq \frac{M}{2} + D$. Therefore,

$$(|I| - |\overline{I}|)\frac{M}{2} \leq \frac{M}{3} \text{ and } (|I| - |\overline{I}|) \leq \frac{2}{3} < 1.$$

By symmetry, we obtain $|I| = |\overline{I}|$ and I is a solution to 2-PART-EQUAL.

Second Reduction

In order to build the ACCuboidP instance that will be used in the reduction, we rely on a result from Walters [2009] stating that it is possible to tile any cuboid with a number of cubes which is poly-logarithmic in the side lengths of the cuboid. We denote the cubes in such a tiling Walters' cubes, and we denote by $WS(X, Y, Z)$ the (poly-logarithmic size) set of cubes tiling the cuboid $X \times Y \times Z$.

Let us consider the following instance of ACCuboidP:

- A cuboid of size $11M \times 15M \times S$ (with $S = 60k \times M$).
- $20k$ cubes of length $6M$.
- $24k$ cubes of length $5M$.
- $30k$ cubes of length $4M$.
- $20k$ cubes of length $3M$.
- $\forall i$, a cube of length b_i.
- $\forall i$, $WS(M - b_i, b_i, b_i)$ and $WS(M, M - b_i, b_i)$.

with M, k and the b_i's defined from the a_i's as in the reduction described above. One can see that the reduction is polynomial, since the sizes of the Walters' cubes sets are poly-logarithmic functions of the b_i's.

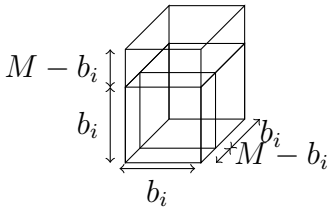


Figure 2.3: Tiling of a $b_i \times M \times M$ cuboid.

In the first part of the proof, we prove that if we can split the b_i items in two equal sets, then the above set of cubes can be packed into the cuboid.

Let us first consider, for each i, the cube of length b_i and the two associated Walters' cubes sets. Figure 2.3 shows how they can be packed into a cuboid of size $M \times M \times b_i$, where the cuboid of size $(M - b_i) \times b_i \times b_i$ and the cuboid of size $(M - b_i) \times b_i \times M$ are tiled with the cubes from $WS(M - b_i, b_i, b_i)$ and $WS(M, M - b_i, b_i)$ respectively. Stacking up such cuboids on top of one another, we can build two $M \times M \times S$ cuboids from the two sets I and $\overline{I}$, see Figure 2.4.

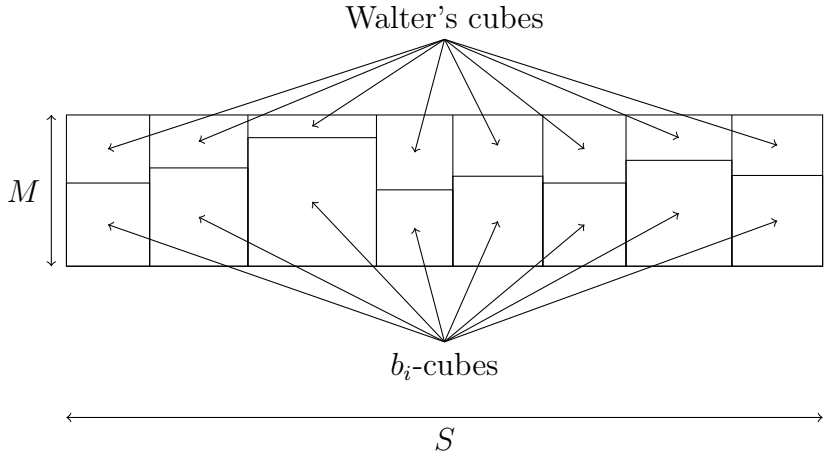


Figure 2.4: Tiling of a $M \times M \times S$ cuboid (on a $M \times S$ face).

Figure 2.5 shows how to tile a $11M \times 15M$ rectangle with the corresponding squares, where both $M \times M$ squares represent a slice of the $M \times M \times S$ cuboids presented above. This arrangement can be repeated for a total length of S, since

$$10k \times 6M = 12k \times 5M = 15k \times 4M = 20k \times 3M = 60k \times M = S.$$

Hence, this provides a tiling of the whole $11M \times 15M \times S$ cuboid.

For the second part of the proof, we need to prove that if the cuboid can be tiled with the set of cubes, then a partition of the b_i values in two equal

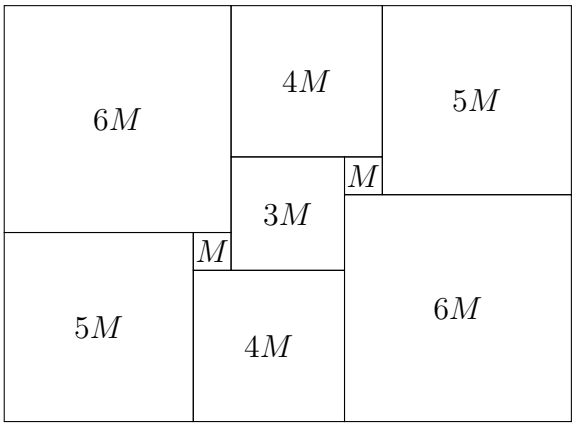


Figure 2.5: Tiling of a $11M \times 15M$ rectangle.

size sets exists. We start by proving that in any valid tiling of the cuboid, the $11M \times 15M$ rectangle can only be tiled as shown on Figure 2.5 (or under the same pattern but up to horizontal symmetry).

Let us note that, except for the b_i-cubes and the Walters' cubes, all cubes have a length that is a multiple of M. Therefore, one can see that the resulting projections of the b_i-cubes and the Walters' cubes in a valid partition on the $11M \times 15M$ rectangle can be seen as several $M \times M$ squares.

Let us consider any valid tiling of the cuboid, and analyse relative arrangements of the $3M$-cubes, b_i-cubes and Walters' cubes. Their total volume is $27M^3 \times 20k + 120M^3 \times k = 660M^3 \times k$. On average over all the S slices of the cuboid, this represents a surface of $\frac{660M^3 \times k}{S} = 11M^2$. We now prove that to tile this $11M \times 15M$ face we need at least a total surface of $11M^2$ coming from $3M$-cubes, b_i-cubes and Walters' cubes.

Let s be the surface coming from these cubes divided by M^2, and assume by contradiction that $s \in [0, 10]$. Let p_6, respectively p_5 and p_4, the number or $6M$-cubes, respectively $5M$-cubes and $4M$-cubes, used to tile the $11M \times 15M$-face. We have computed the possible values such that $(15 \times 11 - s)M^2 = (36p_6 + 25p_5 + 16p_4)M^2$, they can be found on Table 2.1. We now consider each case one by one.

Case (1) $s = 0,\ p_6 = 3,\ p_5 = 1,\ p_4 = 2$**:** 11 and 15 are odd, therefore there must be a least one fraction of an odd square in every line or column (we can see the rectangle as a grid 11×15). The odd cubes are the $5M$-cubes, the $3M$-cubes and the ones we can produce with b_i-cubes and Walters' cubes. Since we have only one $5M$-cube, the total length of odd squares is less than 15, so we cannot have one fraction of an odd square in every column. Therefore the tiling in this case is not possible.

Case (2) $s = 1,\ p_6 = 0,\ p_5 = 4,\ p_4 = 4$**:** As $5 \times 4 > 15$, we cannot have more than three 5-squares in a line and therefore, with four 5-squares, we have to be in one of the arrangements shown in Figure 2.6(a) and Figure 2.6(b). In

Case	s	p_6	p_5	p_4
(1)	0	3	1	2
(2)	1	0	4	4
(3)		1	0	8
(4)	2	2	3	1
(5)	3	0	2	7
(6)	4	1	5	0
(7)		2	1	4

Case	s	p_6	p_5	p_4
	5			
(8)	6	1	3	3
(9)	7	3	2	0
(10)	8	0	5	2
(11)		1	1	6
(12)	9	3	0	3
(13)	10	0	3	5

Table 2.1: Possible values of p_6, p_5 and p_4 as a function of s.

both cases, there exist at least two columns where two 5-squares are superposed (denoted d_i in the figures). However, the only way to complete these columns to a height of 11 is to use a square of size 1, and we have only one M-square since $s = 1$. Therefore the tiling in this case is infeasible.

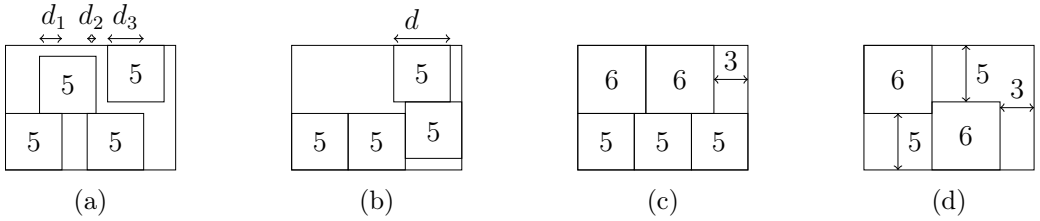


Figure 2.6: Tiling in cases (2), (4) and (9).

Case (3) $s = 1,\ p_6 = 2,\ p_5 = 0,\ p_4 = 8$**:** By a parity argument similar to the one used in Case (1), we can prove the infeasibility of the tiling.

Case (4) $s = 2,\ p_6 = 2,\ p_5 = 3,\ p_4 = 1$**:** It is easy to see that the 6-squares and the 5-squares can only be tiled in a way similar to Figure 2.6(c). Therefore there is no room to place the 4-square and the tiling is infeasible in this case.

Case (5) $s = 3,\ p_6 = 0,\ p_5 = 2,\ p_4 = 7$**:** By a parity argument similar to the one used in Case (1), we can prove the infeasibility of the tiling.

Case (6) $s = 4,\ p_6 = 1,\ p_5 = 5,\ p_4 = 0$**:** By reasoning on the number of 5-squares in a similar fashion than in Case (2), we can prove the infeasibility of the tiling.

Case (7) $s = 4,\ p_6 = 2,\ p_5 = 1,\ p_4 = 4$**:** By a parity argument similar to the one used in Case (1), we can prove the infeasibility of the tiling.

Case (8) $s = 6,\ p_6 = 1,\ p_5 = 3,\ p_4 = 3$**:** Since $5 + 5 + 6 > 15$, we cannot have two 5-squares and one 6-square appearing on the same line. Then we have only two choices to tiles these squares. The first is shown on Figure 2.7(a). In this case there is a rectangle of size $9M \times 6M$ that can be proven infeasible to tile with six M-squares and three $4M$-squares. Therefore we have to be in the

case of Figure 2.7(c), where the dashed zone does not intersect the last $5M$-square (otherwise we are in the case in Figure 2.7(b) that is strictly harder to tile than the case of Figure 2.7(a)). Therefore the dashed zone, a square of size $5M \times 5M$ has to be tiled with only fractions of $4M$-squares and M-squares. In order to fill the gap of size $5M$ the only way to begin is the one shown in Figure 2.7(d). To complete the last column, we have to fill the $7M$-gap, and $7 = 5 + 2 \times 1$ is the only possibility (there are only two M-squares remaining). This yields the situation shown in Figure 2.7(e), and one can see that the tiling can not be finished with the remaining $4M$-squares: the tiling is infeasible in this case.

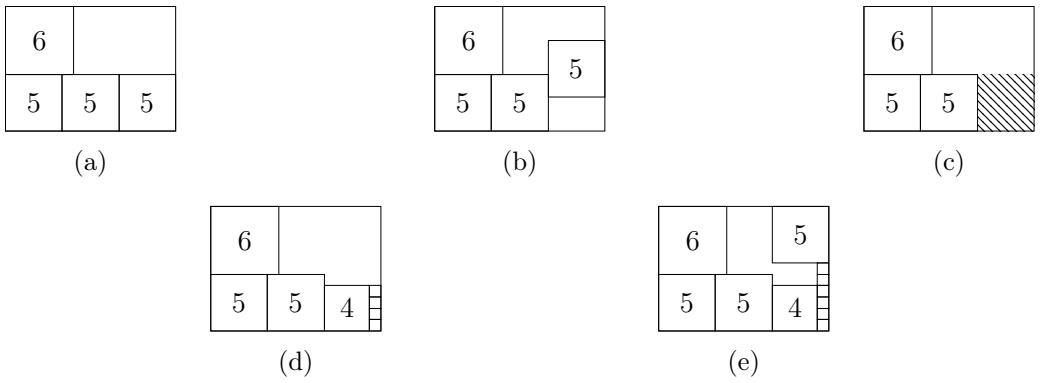


Figure 2.7: Tiling in case (8).

Case (9) $s = 7,\ p_6 = 3,\ p_5 = 2,\ p_4 = 0$**:** It is impossible to have three $6M$-squares. Indeed $2 \times 6 > 11$ and then we cannot have two $6M$-squares appearing on the same column. In the same time, $3 \times 6 > 15$ and then we cannot have three $6M$-squares appearing on the same line (see Figure 2.6(d) to have a better visualization). Hence there is no room to store three $6M$-squares, and the tiling in this case is infeasible.

Case (10) $s = 8,\ p_6 = 0,\ p_5 = 5,\ p_4 = 2$**:** By reasoning on the number of 5-squares in a similar fashion than in Case (2), we can prove the infeasibility of the tiling.

Case (11) $s = 8,\ p_6 = 1,\ p_5 = 1,\ p_4 = 2$**:** By a parity argument similar to the one used in Case (1), we can prove the infeasibility of the tiling.

Case (12) $s = 9,\ p_6 = 3,\ p_5 = 0,\ p_4 = 3$**:** By a parity argument similar to the one used in Case (1), we can prove the infeasibility of the tiling.

Case (13) $s = 10,\ p_6 = 0,\ p_5 = 3,\ p_4 = 5$**:** We consider two sub-cases: either we have one $3M$-square and one M-square, or we have ten M-square. In the first case we notice that with the available squares there are only two ways to achieve a exact length of $11M$: $1M + 5M + 5M$ and $3M + 4M + 4M$. Both options create gaps that cannot be filled with the remaining squares, see Figure 2.8(a). In the second case (ten M-squares and no $3M$-square), there

are more options to achieve a length of $11M$. The first one is to use two $4M$-square and three M-squares. This creates a $3M \times 4M$ gap, and there are too few M-squares to fill it, see Figure 2.8(b), therefore we cannot tile this way. The second option is to use two $5M$-squares and one M-square. In this case we have to use four more M-squares to complete and then we have again a length $11M$ to achieve, that, with the remaining resources, can only be done either with two $4M$-squares and three M-squares, and we have proven above that it is not feasible, or with a $4M$-square, a $5M$-square and two M-squares, but in this case we cannot tile the resulting $2M \times 3M$ rectangle because we have too few M-squares, see Figure 2.8(c). Another option is to use a $4M$-square and seven M-squares, but this results in a gap of length $7M$ to fill and one can see that is not possible, see Figure 2.8(d) (replacing the $5M$-square of the figure by a $4M$-square yields the same problem). Another possibility is to use a $5M$-square and six M-squares, for which the only reasonable continuation is the case shown on Figure 2.8(e) and this creates a gap of length $6M$ which cannot be tiled with the remaining squares. The last option is to use a $5M$-square, a $4M$-square and two M-squares but the tiling is still impossible, as it is shown on Figure 2.8(f). Therefore, in any sub-case, the tiling is infeasible in this case.

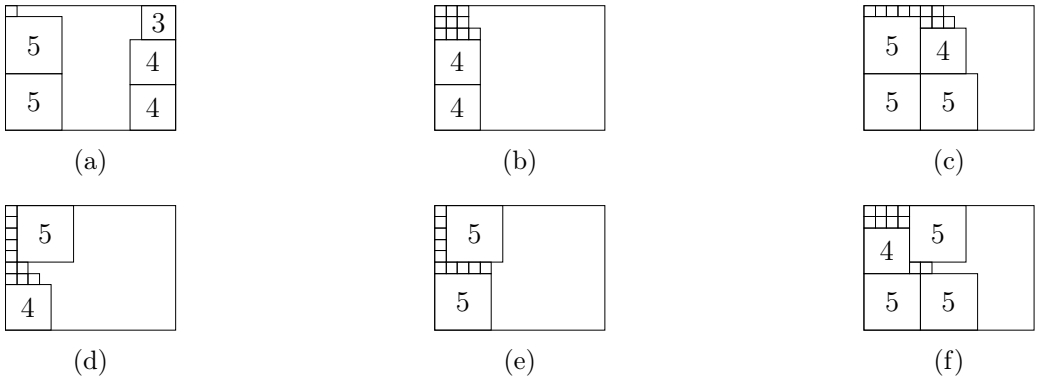


Figure 2.8: Tiling in case (13).

This proves that at every slice of the main cuboid, the surface coming from the $3M$-cubes, the b_i-cubes and the Walters' cubes is at least $11M^2$. Since this is exactly the average value, this implies that this surface is exactly $11M^2$ for each slice. We can now observe that this requires at least two M-squares in any slice and once again this is exactly the average value of M-squares. Therefore, in any slice, there are exactly two M-squares and one $3M$-square. The argument used to create Table 2.1 now proves that the slice necessarily includes two $6M$-squares, two $5M$-squares and two $4M$-squares, and one can prove that the only way to tile the $11M \times 15M$ rectangle with these squares is as shown on Figure 2.5.

We have built a pattern that must appear on each slice of the cuboid, and in which the b_i cubes have to be included into two separate parts of the tiling. Let us denote by I the indexes of the b_i cubes which appear in the leftmost $M \times M \times S$ cuboid. Since by construction, $b_i > \frac{M}{2}$ for all i, these cubes have to be arranged as depicted on Figure 2.4. This proves that $\sum_{i \in I} b_i \leq S$ and $\sum_{i \notin I} b_i \leq S$. Since the total sum is $2S$, this implies that both sums are in fact equal to S, and thus that there exists a solution to the original 2-PART-EQUAL instance.

Note that the pattern in Figure 2.5 can be horizontally reversed. Furthermore, both possible patterns can be present on the final tiling. However, even in this case, considering the b_i-cubes on the left side still yields a set I such that $\sum_{i \in I} b_i = \sum_{i \notin I} b_i = S$.

□

Theorem 2.5. *MSCuboidP-DEC is NP-complete.*

Proof. There is a reduction from ACCuboidP to MSCuboidP-DEC. Indeed, $ACCuboidP(\{l_1, \ldots, l_m\}, [0, x] \times [0, y] \times [0, z])$ is feasible if and only if $MSCuboidP-DEC(\{l_1^3, \ldots, l_m^3\}, [0, x] \times [0, y] \times [0, z], 3 \sum v_k^{2/3})$ is feasible (the bound in (2.1) is tight if and only if there is a partitioning where only cubes are used). Yet, thanks to Lemma 2.4, ACCuboidP is NP-complete. Therefore MSCuboidP-DEC is NP-complete. □

2.4 3D-NRRP (3D Non-Rectangular Recursive Partitioning)

Now that we prove NP-completeness, we look for an approximation algorithm. In this section, we present 3D-NRRP, an approximation algorithm for MSCubeP and MSCuboidP. This algorithm is basically the 3D-counterpart of SNRRP (Section 1.4.1) and we prove it is a $\frac{5}{6^{2/3}}$-approximation ($\frac{5}{6^{2/3}} \simeq 1.51$) for MSCubeP.

2.4.1 Presentation of the Algorithm

3D-NRRP (see Algorithm 2.2) is based on the divide and conquer principle. At each step of the algorithm, the current cuboid is split into two parts, and the same routine is recursively applied to each part. The splitting can be performed according to three modes (the choices between the three modes are made to ensure important invariants for the approximation ratio).

The first mode is the general case, in which the cuboid is partitioned in two disjoint cuboids by cutting along the largest side (Lines 7 to 18 in Algorithm 2.2, and Figure 2.9(a)). If this is not possible, v_m can be proved to be significantly larger than the other v_i's. Therefore, whenever possible (Lines 23

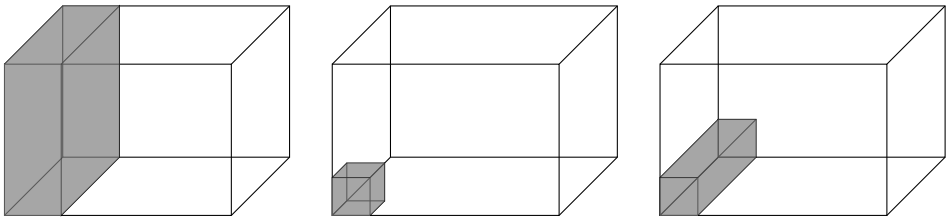
to 24 of Algorithm 2.2, and Figure 2.9(b)), 3D-NRRP shapes the smallest part as a cube included in the covering cuboid of the other part, which is made of one element only, namely v_m. In some cases, the edge length of this cube can be larger than the smallest dimension of the cuboid. In that case, we create a cuboid like in Figure 2.9(c) (Line 26-32 of Algorithm 2.2) by setting one dimension to the smallest dimension of the current cuboid and making the remaining two dimensions equal.

Algorithm 2.2 : 3D-NRRP

Input : A set of values $\{v_1, \ldots, v_m\}$ sorted in non-decreasing order, a cuboid $Cu = [x_1, x_2] \times [y_1, y_2] \times [z_1, z_2]$ such that $v(Cu) = \sum_{i=1}^{m} v_i$

Output : For each $1 \leq i \leq m$ a polyhedron P_i such that $\bigcup P_i = Cu$ and $v(P_i) = v_i$

1 **if** $m = 1$ **then**
2 **return** Cu
3 **else**
4 $v = \sum_{i=1}^{m} v_i$;
5 $w = w(Cu)$; $h = h(Cu)$; $d = d(Cu)$;
6 $\rho = \frac{max(w,h,d)}{min(w,h,d)}$; $\rho' = \frac{max(w,h,d)}{med(w,h,d)}$;
7 **if** *there exists* k *such that* $\sum_{i=1}^{k-1} v_i \geq \frac{v}{3\rho'}$ **then**
8 k = the smallest such index k ;
9 $v' = \sum_{i=1}^{k-1} v_i$;
10 **if** $w = max(w, h, d)$ **then**
11 $Cu_1 = [x_1, x_1 + v'/(h \times d)] \times [y_1, y_2] \times [z_1, z_2]$;
12 $Cu_2 = [x_1 + v'/(h \times d), x_2] \times [y_1, y_2] \times [z_1, z_2]$
13 **else**
14 **if** $h = max(w, h, l)$ **then**
15 $Cu_1 = [x_1, x_2] \times [y_1, y_1 + s'/(w \times d)] \times [z_1, z_2]$;
16 $Cu_2 = [x_1, x_2] \times [y_1 + s'/(w \times d), y_2] \times [z_1, z_2]$
17 **else**
18 $Cu_1 = [x_1, x_2] \times [y_1, y_2] \times [z_1, z_1 + v'/(w \times h)]$;
19 $Cu_2 = [x_1, x_2] \times [y_1, y_2] \times [z_1 + v'/(w \times h), z_2]$
20 **return** 3D-NRRP$(\{v_1, \ldots, v_{k-1}\}, Cu_1)$ + 3D-NRRP$(\{v_k, \ldots, v_m\}, Cu_2)$;
21 **else**
22 $v' = \sum_{i=1}^{m-1} v_i$ and $\alpha = v'/v$;
23 **if** $\alpha\rho^2 \leq \rho'$ **then**
24 $Cu_1 = [x_1, x_1 + \sqrt[3]{v'}] \times [y_1, y_1 + \sqrt[3]{v'}] \times [z_1, z_1 + \sqrt[3]{v'}]$;
25 **else**
26 **if** $w = min(w, h, d)$ **then**
27 $Cu_1 = [x_1, x_2] \times [y_1, y_1 + \sqrt{v'/w}] \times [z_1, z_1 + \sqrt{v'/w}]$;
28 **else**
29 **if** $h = min(w, h, d)$ **then**
30 $Cu_1 = [x_1, x_1 + \sqrt{v'/h}] \times [y_1, y_2] \times [z_1, z_1 + \sqrt{v'/h}]$;
31 **else**
32 $Cu_1 = [x_1, x_1 + \sqrt{v'/d}] \times [y_1, y_1 + \sqrt{v'/d}] \times [z_1, z_2]$;
33 **return** 3D-NRRP$(\{v_1, \ldots, v_{m-1}\}, Cu_1) + (Cu \setminus Cu_1)$



(a) (b) (c)

Figure 2.9: The three splitting modes of 3D-NRRP. In all cases, the gray polyhedra is attributed to $\{v_1, \ldots, v_{k-1}\}$, the white one to $\{v_k, \ldots, v_m\}$.

2.4.2 Correctness

The proof of the approximation ratio relies on an important invariant: the aspect ratio of the cuboids on which 3D-NRRP is called is smaller than 3. Let us now check that this invariant is satisfied after any of the three splitting modes of 3D-NRRP.

In the first mode, in which the cuboid is partitioned into two disjoint cuboids by cutting along the largest side (Figure 2.9(a)), we know there is an index k such that $\sum_{i=1}^{k-1} v_i \geq \frac{v}{3\rho'}$. Hence, we can apply Lemma 1.6 (page 34) to ensure that $\sum_{i=k}^{m} v_i \geq \frac{v}{3\rho'}$. We then use Lemma 2.6 to conclude.

Lemma 2.6. *Let Cu be a cuboid of dimension $w \times h \times d$, with volume $V = hwd$, aspect ratio ρ, and secondary aspect ratio ρ'. Let us assume that Cu_1 and Cu_2 are obtained by cutting Cu along the largest side, with $V(Cu_1)$ and $V(Cu_2)$ not smaller than $\frac{V}{3\rho'}$. Then, $\rho(Cu_1) \leq max(3, \rho)$ and $\rho(Cu_2) \leq max(3, \rho)$.*

Proof. Let us suppose that $w \leq h \leq d$ without loss of generality. In this case, $w = w(Cu_1) = w(Cu_2)$, $h = h(Cu_1) = h(Cu_2)$, $\rho = d/w$ and $\rho' = d/h$. Let us denote $d_1 = d(Cu_1)$ and $d_2 = d(Cu_2)$ and let us consider the cuboid Cu_1. Then, there are 3 cases to consider:

- If $h \leq d_1$, then $\rho(Cu_1) = d_1/w \leq d/w = \rho$,
- If $w \leq d_1 \leq h$, then $\rho(Cu_1) = h/w \leq d/w \leq \rho$,
- If $d_1 \leq w \leq h$, by assumption $w \times h \times d_1 = V(Cu_1) \geq \frac{V}{3\rho'} = \frac{w \times h \times d}{3\rho'}$. Therefore $d_1 \geq d/3\rho'$. Then, $\rho(Cu_1) = h/d_1 \leq \frac{3h\rho'}{d} = 3$.

Thus, in all cases, $\rho(Cu_1) \leq max(3, \rho)$. By symmetry, the same proof applies to Cu_2. □

In the second case, v_m is significantly larger than the other v_is and these last ones are packed into a single cube (see Figure 2.9(b)). Thus, in this case,

the only produced cuboid on which 3D-NRRP is called is a cube whose aspect ratio is then $1 \leq 3$. Therefore, the only thing left here is to prove that we can perform such a partitioning, *i.e.* the side of the cube is smaller than the width, the height and the depth of the initial cuboid. The condition $\frac{v'}{v}\rho^2 \leq \rho'$ and Lemma 2.7 ensures that this is indeed possible (the length of the cube is $\sqrt[3]{v'}$).

Lemma 2.7. *Let Cu be a cuboid of dimension $w \times h \times d$, with volume $V = hwd$, aspect ratio ρ, and second aspect ratio ρ'. For any $\alpha > 0$ such that $\alpha\rho^2 \leq \rho'$, then $\sqrt[3]{\alpha V} \leq \min(w, h, d)$.*

Proof. Without loss of generality, we can assume that $w \leq h \leq d$ and therefore $d = \rho w$ and $d = \rho' h$ and finally $h = \frac{\rho}{\rho'}w$. Thus:

$$\sqrt[3]{\alpha V} = \sqrt[3]{\alpha whd} = \sqrt[3]{\alpha\frac{\rho^2}{\rho'}}w \leq w.$$

□

In the third case, v_m is also significantly larger than the other v_is but these last ones may not be packed into a single cube. In this case the packing is made with a small cuboid (see Figure 2.9(c)). In this case we use Lemma 2.8 that states that this cuboid has an aspect ratio below 3 (more precisely, an aspect ratio smaller than the initial cuboid that is, according to the invariant, below 3) and that the dimensions of this cuboid are small enough to be contained by the initial cuboid.

Lemma 2.8. *Let Cu be a cuboid of dimension $w \times h \times d$, with volume $V = hwd$, aspect ratio ρ, and second aspect ratio ρ'. Let α be such that $\frac{\rho'}{\rho^2} < \alpha < \frac{1}{3\rho'}$. Let us denote $\min_l = \min(w, h, d)$. Then $\sqrt{\alpha\frac{V}{\min_l}} \leq \mathrm{med}(w, h, d)$. In addition, if Cu' is a cuboid of dimensions $\min_l \times \sqrt{\alpha\frac{V}{\min_l}} \times \sqrt{\alpha\frac{V}{\min_l}}$ then $\rho(Cu') \leq \rho$.*

Proof. Without loss of generality we can suppose $\min_l = w \leq h \leq d$ and therefore $d = \rho w$ and $d = \rho' h$ and finally $h = \frac{\rho}{\rho'}w$. Thus:

$$\sqrt{\alpha\frac{V}{w}} = \sqrt{\alpha hd} = (\sqrt{\alpha\rho'})h \leq \frac{h}{\sqrt{3}} \leq h.$$

Moreover $\frac{\rho'}{\rho^2} < \alpha$ implies $\sqrt{\alpha\rho'} \geq \frac{\rho'}{\rho}$, and thus $\sqrt{\alpha\frac{V}{w}} = h\sqrt{\alpha\rho'} \geq \frac{h\rho'}{\rho} = w$. Then, $\rho(Cu') = \frac{\sqrt{\alpha\frac{V}{w}}}{w}$, and the previous result yields $\rho(Cu') \leq \frac{h}{w} = \frac{\rho}{\rho'} \leq \rho$.

□

Thus, we cover the three modes and ensure the invariant stated by Theorem 2.9.

Theorem 2.9. *During the execution of 3D-NRRP, all recursive calls are made on a cuboid with an aspect ratio below* 3.

2.4.3 Approximation Ratio

This part is devoted to the proof of Theorem 2.10, which states that 3D-NRRP achieves a $\frac{5}{6^{2/3}}$ approximation ratio ($\frac{5}{6^{2/3}} \simeq 1.51$).

Theorem 2.10. *3D-NRRP is a $\frac{5}{6^{2/3}}$-approximation for MSCubeP.*

Proof. The sketch of the proof is as follows. First, we prove that if $\{P_1, \ldots, P_m\}$ is an output of 3D-NRRP, then any output polyhedron P_i satisfies $\frac{Hs(P_i)}{3V(P_i)^{\frac{2}{3}}} \leq \frac{5}{6^{\frac{2}{3}}}$. Remind that Equation (2.1) states that $Hs(P_i^*) \geq 3v(P_i^*)^{\frac{2}{3}}$ for any (optimal) solution $\{P_1^*, \ldots, P_m^*\}$. By summing up these inequalities for all i, we get that $\sum_i Hs(P_i) \leq \frac{5}{6^{\frac{2}{3}}} \sum_i Hs(P_i^*)$ and obtain the approximation result claimed in Theorem 2.10.

The rest of the section is devoted to the proof that any polyhedron returned by 3D-NRRP satisfies the property stated above. Let us first observe that there are only two situations in which 3D-NRRP returns a singleton: Line 2 and Line 33. In the first case, according to Theorem 2.9, the returned polyhedron is a cuboid with aspect ratio below 3. In this case Lemma 2.12 provides the desired result, since $\frac{5}{3\sqrt[3]{3}} \leq \frac{5}{6^{\frac{2}{3}}}$. Lemma 2.11 is a technical lemma in order to prove Lemma 2.12.

Lemma 2.11. *Let $f(x,y) = \frac{y+x(1+y)}{3(xy)^{2/3}}$. Then, with $x \in [1,y]$ and $y \in [1,3]$, $f(x,y) \leq \frac{5}{3\sqrt[3]{3}}$.*

Proof. One can show that $\frac{\partial f}{\partial x}(x,y) = \frac{2x-1-y}{9y^{1/3}x^{4/3}}$. Since $x \geq 0$ and $y \geq 0$, $\frac{\partial f}{\partial x}(x,y) \geq 0$ if and only if $x \leq \frac{1+y}{2}$. Hence, $x \mapsto f(x,y)$ is decreasing on $[1, \frac{1+y}{2}]$ and increasing in $[\frac{1+y}{2}, y]$, which implies $f(x,y) \leq max(f(1,y), f(y,y))$.

$f(1,y) = \frac{y+2}{3\sqrt[3]{y}}$ and $f(y,y) = \frac{2y+1}{3y^{2/3}}$. From $y \geq 1$, one can obtain $(y+2)\sqrt[3]{y} \geq 2y+1$, and this implies $\frac{y+2}{3\sqrt[3]{y}} \geq \frac{2y+1}{3y^{2/3}}$. Let us denote $g(y) = \frac{y+2}{3\sqrt[3]{y}}$: the previous statements show that $f(x,y) \leq g(y)$. One can prove that $g'(y) = \frac{2(y-1)}{9y^{4/3}}$. Therefore g is increasing on $[1,\infty]$ and, in the considered case, $g(y) \leq g(3) = \frac{5}{3\sqrt[3]{3}}$, what proves the claimed result. □

Lemma 2.12. *If P is a cuboid with $\rho(P) \leq 3$, then $\frac{Hs(P)}{3v(P)^{2/3}} \leq \frac{5}{3\sqrt[3]{3}}$.*

Proof. Let us denote $\rho(P) = \rho$ and $v(P) = V$. Let us suppose that $w = w(P) \leq h = h(P) \leq d(P) = d$ without loss of generality. Let us denote $x = h/w$. In this case $d = \rho w$ and $V = whd = \rho x w^3$. Therefore

$$\frac{Hs(P)}{3V^{2/3}} = \frac{(x+\rho+x\rho)w^2}{3(\rho x w^3)^{2/3}} = \frac{\rho + x(1+\rho)}{3(\rho x)^{2/3}} = f(x,\rho),$$

where f is as defined in Lemma 2.11, what ends the proof. □

In the other case, 3D-NRRP may return a cuboid minus a cube, as described on Figure 2.9(b), or a cuboid minus a small cuboid, as described on Figure 2.9(c). The bound on the volume of the cube or the small cuboid is such that the conditions of Lemma 2.14 are satisfied. As before, this relies on the technical Lemma 2.13.

Lemma 2.13. *Let* $f(x,y) = \frac{y+x(1+y)}{3(xy-\frac{x^2}{3})^{2/3}}$. *Then, with* $x \in [1,y]$ *and* $y \in [1,3]$, $f(x,y) \leq \frac{5}{6^{2/3}}$.

Proof. One can prove that $\frac{\partial f}{\partial x}(x,y) = \frac{\frac{1+y}{3}x^2+(\frac{7}{3}+y)yx-2y^2}{9(yx-\frac{x^2}{3})^{5/3}}$. Then $\frac{\partial f}{\partial x}$ has the same sign than $P_y(x) = \frac{1+y}{3}x^2+(\frac{7}{3}+y)yx-2y^2$. Since P_y is a second order polynomial with $P_y(0) = -2y^2 < 0$ and $\lim_{x\to+\infty} p_y(x) = +\infty$, P_y is either positive on $[1,y]$, negative on $[1,y]$, or negative and then positive. Therefore $x \mapsto f(x,y)$ on $[1,y]$ is either increasing, decreasing or decreasing and then increasing. In any case, $f(x,y) \leq \max(f(1,y), f(y,y))$.

Let $g(y) = f(1,y) = \frac{1+2y}{3(y-1/3)^{2/3}}$. One can show that $g'(y) = \frac{2(y-2)}{9(y-1/3)^{5/3}}$, hence g is decreasing on $[1,2]$ and increasing on $[2,3]$. Therefore $g(y) \leq \max(g(1), g(3)) = \max((\frac{3}{2})^{2/3}, \frac{7}{4\sqrt[3]{3}}) = (\frac{3}{2})^{2/3}$.

Let $h(y) = f(y,y) = \frac{2+y}{\sqrt[3]{12y}}$. One can show that $h'(y) = \frac{2(y-1)}{\sqrt[3]{12y^2}}$, hence h is increasing on $[1,3]$. Therefore $h(y) \leq h(3) = \frac{5}{6^{2/3}}$.

Putting it all together, we obtain $f(x,y) \leq \max((\frac{3}{2})^{2/3}, \frac{5}{6^{2/3}}) = \frac{5}{6^{2/3}}$. □

Lemma 2.14. *If* $v(P) \geq (1 - \frac{1}{3\rho'(P)})v(Cu(P))$ *and* $\rho(P) \leq 3$, *then* $\frac{Hs(P)}{3v(P)^{2/3}} \leq \frac{5}{6^{2/3}}$.

Proof. Let us denote $\rho = \rho(P)$, $\rho' = \rho'(P)$, $V = v(P)$, and let us denote by w, h, d for the dimensions of $Cu(P)$. Let us assume that $w \leq h \leq d$ without loss of generality. Then, $d = \rho w$ and $d = \rho' h$, and let us also denote $x = h/w = \rho/\rho'$. With such notations, we get $v(Cu(P)) = \rho x w^3$ and $Hs(P) = (\rho + x(1+\rho))w^2$. Thus, the condition on $V(P)$ can be written as

$$v(P) \geq (1 - \frac{1}{3\rho'(P)})v(Cu(P)) = (1 - \frac{x}{3\rho})\rho x w^3 = (\rho x - \frac{x^2}{3})w^3,$$

$$\text{what leads to } \frac{Hs(P)}{3V(P)^{2/3}} \leq \frac{(\rho + x(1+\rho))w^2}{3(\rho x - \frac{x^2}{3})^{2/3}w^2} = \frac{\rho + x(1+\rho)}{3(\rho x - \frac{x^2}{3})^{2/3}} = f(x,\rho),$$

where f is as defined in Lemma 2.13, what achieves the proof. □

Thus in both cases we prove our approximation, which achieves this proof. □

Note that the approximation ratio is also valid for any instance of MSCuboidP where the input cuboid has an aspect ratio below 3.

Also, if only the first mode is used (that may happen for instance with small heterogeneity), then for all k, $\frac{Hs(P_k)}{3V(P_k)^{2/3}} \leq \frac{5}{3\sqrt[3]{3}}$ according to Lemma 2.6 and Lemma 2.12 ($\frac{5}{3\sqrt[3]{3}} \simeq 1,16$). A possible sufficient condition may be for $k > 1$, $\frac{1}{2}v_k \leq v_{k-1}$, the same condition to have RRP being a $\frac{2}{\sqrt{3}}$-approximation according to Fügenschuh *et al.* [2014]. Indeed in this case, if $(\{v_i, \ldots, v_j\}, Cu)$ is the input of a recursive call of 3D-NRRP, then, with $v = \sum_{k=i}^{j} v_k$, $v - v_k = \sum_{k=i}^{j-1} v_k \geq v_{j-1} \geq \frac{1}{2}v_j$. Thus $v \geq \frac{3}{2}v_j$, that is equivalent to $v - v_j \geq \frac{1}{3}v$ which implies $v - v_j \geq \frac{1}{3\rho'(Cu)}v$. Hence there is $k' < j$ such that $\sum_{k=i}^{k'} v_k \geq \frac{1}{3}v$ and therefore Lemma 2.6 applies. Thus we have just proven Theorem 2.15.

Theorem 2.15. *On the set of instance $\{v_1, \ldots, v_m\}$ of MSCubeP such that $\forall 1 < k \leq m$, $\frac{1}{2}v_k \leq v_{k-1}$, 3D-NRRP is a $\frac{5}{3\sqrt[3]{3}}$-approximation of MSCubeP.*

2.4.4 Complexity

As for SNRRP, for the instance $\{v_1, \ldots, v_m\}$, by pre-computing the partial sums, the cost of each call of 3D-NRRP is $O(\log m)$. Thus, as there are at most m calls (one for each v_k), the worst case complexity of 3D-NRRP is $O(m \log m)$.

2.5 3D-SFCP (3D Space-Filling Curve Partitioning)

In this section we present the analysis of 3D-SFCP, the 3D-counterpart of SFCP (Section 1.5), that we prove to be a $\frac{7^{\frac{5}{3}}}{6^{\frac{2}{3}}}$-approximation ($\frac{7^{\frac{5}{3}}}{6^{\frac{2}{3}}} \simeq 1.64$). The basic principle is still the same as for SFCP, 3D-SFCP uses space-filling curves to go through the cube and perform the partitioning. Note that, as for SFCP, 3D-SFCP is an algorithm that can be used directly to solve the discrete version of MSCubeP. Note that the analysis we provide is really close to the one in Section 1.5.

2.5.1 Presentation of 3D-SFCP

In the following we consider an $N \times N \times N$ cube and suppose that N is a power of 2. Note that we can return to MSCubeP by simply rescaling the cube (in particular if we divide the covering half-surfaces by N^2 we have the covering half-surfaces for MSCubeP). Note also that we now consider a discrete problem and no longer a continuous one (there are N^3 elements in the cube to be allocated to processors). Note that the notation from MSCubeP can be adapted easily.

If Z is a subset of $[1,N]\times[1,N]\times[1,N]$, now $\Pi_1(Z)=\{(i,k),\ (i,j,k)\in Z\}$, $\Pi_2(Z)=\{(j,k),\ (i,j,k)\in Z\}$ and $\Pi_3(Z)=\{(i,j),\ (i,j,k)\in Z\}$ are the projections of Z in this case (and $\pi_i(Z)=|\Pi_i(Z)|$ for $i\in\{1,2,3\}$).

Let us first note, as described in Lemma 2.16 (which is a direct adaptation of Lemma 1.33), that the projection of the union of two subsets of $[1,N]\times[1,N]\times[1,N]$ is smaller than the sum of the projections, even for disjoint subsets.

Lemma 2.16. *Let X and Y be two subsets of $[1,N]^3$. Then, $\forall i\in[1,3]$,*

$$\pi_i(X\cup Y)\leq\pi_i(X)+\pi_i(Y).$$

Proof. We prove here that the property holds true for π_1 (proof similar for π_2 and π_3). We have

$$\begin{aligned}\Pi_1(X\cup Y)&=\{(i,k),\ (i,j,k)\in X\cup Y\}\\&=\{(i,k),\ (i,j,k)\in X\}\cup\{(i,k),\ (i,j,k)\in Y\}\\&=\Pi_1(X)\cup\Pi_1(Y).\end{aligned}$$

Therefore $\pi_1(X\cup Y)\leq\pi_1(X)+\pi_1(Y)$. Indeed,

$$\pi_1(X\cup Y)=|\Pi_1(X\cup Y)|=|\Pi_1(X)\cup\Pi_1(Y)|\leq|\Pi_1(X)|+|\Pi_1(Y)|=\pi_1(X)+\pi_1(Y).$$

□

The q-cubes, the 3D-counterpart of q-squares, are now defined as follows.

Definition 2.1. Let us denote as q-cubes all the subcubes of the cube $[1,N]^3$ which are of the form $[1+n_1 2^q,(n_1+1)2^q]\times[1+n_2\times 2^q,(n_2+1)2^q]\times[1+n_3 2^q,(n_3+1)2^q]$ where $n_1,n_2,n_3\in[0,N/2^q-1]^3$.

See Figure 2.5.1 for an illustrated example of q-cubes.

The study we propose is a bit more general than Hilbert's curve only, even if we use this curve in particular for implementation. It applies to every space-filling curve that can be seen as a function $H:[1,N^3]\to[1,N]^3$ that fulfils the following two conditions.

Property 2.1. If C_q is a q-cube, then there exists a positive integer n such that $H([n,n+2^{3q}-1])=C_q$.

Property 2.2. $\forall n\in[1,N^3-1]$, if we denote $(i_1,j_1,k_1)=H(n)$ and $(i_2,j_2,k_2)=H(n+1)$, then two of these are true:

- $i_1=i_2$;

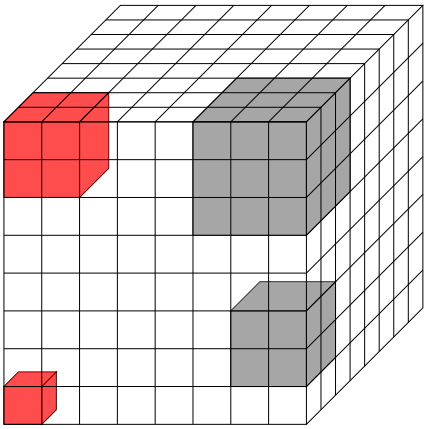


Figure 2.10: The two red cubes are 0-cube and 1-cube, the two gray ones are not q-cubes.

- $j_1 = j_2$;
- $k_1 = k_2$.

The first one means that the path through the cube $[1, N]^3$ is built, at every level, one q-cube after another. The second means that there are no diagonal moves. From these two properties we can deduce than the second one is transferred at any level to the q-cubes, which is described in the following corollary.

Corollary 2.17. *Let H be a function from $[1, N^3]$ to $[1, N]^3$ which satisfies Property 2.1 and Property 2.2. Let q be in $\mathbb{N}$ and C_1, C_2 be two q-cubes with $C_1 = I_1 \times J_1 \times K_1$ and $C_2 = I_2 \times J_2 \times K_2$. If we suppose there exists an index n such that $H([n, n + 2^{3q} - 1]) = C_1$ and $H([n + 2^{3q}, n + 2 \times 2^{3q} - 1]) = C_2$, then two of these hold true:*

- $I_1 = I_2$;
- $J_1 = J_2$;
- $K_1 = K_2$.

Note that this is equivalent to the following: if there exists an index n such that $H([n, n + 2^{3q} - 1]) = C_1$ and $H([n + 2^{3q}, n + 2 \times 2^{3q} - 1]) = C_2$, then one of these hold true:

- $\Pi_1(C_1) = \Pi_1(C_2)$;
- $\Pi_2(C_1) = \Pi_2(C_2)$;
- $\Pi_3(C_1) = \Pi_3(C_2)$.

Proof. First if $C_1 = I_1 \times J_1 \times K_1$ and $C_2 = I_2 \times J_2 \times K_2$ are two q-cubes, then $I_1 \cap I_2 \neq \emptyset$ (respectively $J_1 \cap J_2 \neq \emptyset$ and $K_1 \cap K_2 \neq \emptyset$) implies $I_1 = I_2$ (respectively $J_1 = J_2$ and $K_1 = K_2$). This can be proved simply by recalling that $I_1 = [1 + n_1 \times 2^q, (n_1 + 1) \times 2^q]$ and $I_2 = [1 + n_2 \times 2^q, (n_2 + 1) \times 2^q]$ with $n_1, n_2 \in [0, N/2^q - 1]^2$ and thus if $n_1 \neq n_2$ the intersection is empty.

Let us now prove Corollary 2.17 by induction on q.

The case $q = 0$ corresponds exactly to Property 2.2 and therefore holds true.

Let us now suppose that the result holds true for $q-1$. Let $C_1 = I_1 \times J_1 \times K_1$ and $C_2 = I_2 \times J_2 \times K_2$ be two q-cubes and k such that $H([k, k + 2^{3q} - 1]) = C_1$ and $H([k + 2^{3q}, k + 2 \times 2^{3q} - 1]) = C_2$. Thanks to Property 2.1 we know that if the $C_{1,i}$s, $i \in [1, 8]$ are the $(q-1)$-cubes that compose C_1, then $\forall i \in [1, 8]$ there exists k_i such that, $H([k_i, k_i + 2^{3(q-1)} - 1]) = C_{1,i}$. If we suppose that $k_i < k_{i+1}$ $\forall i \in [1, 7]$, as $\bigcup [k_i, k_i + 2^{3(q-1)} - 1] = [k, k + 2^{3q} - 1]$, then

- $k_1 = k$;
- $k_i = k_{i-1} + 2^{3(q-1)}$ otherwise.

Similarly we can define $C_{2,i}$, $i \in [1, 8]$, as the $(q-1)$-cubes included in C_2 and the k'_is such that $H([k'_i, k'_i + 2^{3(q-1)} - 1]) = C_{2,i}$. Similarly

- $k'_1 = k + 2^{3q}$;
- $k'_i = k'_{i-1} + 2^{3(q-1)}$ otherwise.

In addition

$$k + 2^{3q} = k_1 + 8 * 2^{3(q-1)} = k_2 + 7 * 2^{3(q-1)} = \ldots = k_8 + 2^{3(q-1)}$$

and then $H([k_8, k_8 + 2^{3(q-1)} - 1]) = C_{1,8}$ and $H([k_8 + 2^{3(q-1)}, k_8 + 2 \times 2^{3(q-1)} - 1]) = C_{2,1}$. If we now denote $C_{1,8} = I'_1 \times J'_1 \times K'_1$ and $C_{2,1} = I'_2 \times J'_2 \times K'_2$, by assumption, two of these hold true:

- $I'_1 = I'_2$;
- $J'_1 = J'_2$;
- $K'_1 = K'_2$.

Thus two of these hold true:

- $I_1 \cap I_2 \neq \emptyset$ implying $I_1 = I_2$;
- $J_1 \cap J_2 \neq \emptyset$ implying $J_1 = J_2$;
- $K_1 \cap K_2 \neq \emptyset$ implying $K_1 = K_2$;

what achieves the proof for q. □

Relying on a function H with such properties, 3D-SFCP simply performs the partitioning going through the cube until the wanted volume is reached, as described in Algorithm 2.3.

Algorithm 2.3 : 3D-SFCP

Input : a set of positive integer values $\{v_1, \ldots, v_m\}$ such that $\sum v_i = N^3$

Output : For each $1 \leq i \leq m$, a zone Z_i such that $v(Z_i) = v_i$ and $\bigcup Z_i = [1, N]^3$

$v_{tot} = 0$

foreach $k \in [1, m]$ **do**

  $Z_k = H([v_{tot}, v_{tot} + v_k])$

  $v_{tot} = v_{tot} + v_k + 1$

2.5.2 Approximation Ratio

We claim that 3D-SFCP is a $\frac{7^{\frac{5}{3}}}{62^{\frac{2}{3}}}$-approximation ($\frac{7^{\frac{5}{3}}}{62^{\frac{2}{3}}} \simeq 1.64$) to MSCubeP in Theorem 2.18, and prove this ratio in this section.

Theorem 2.18. *3D-SFCP is a $\frac{7^{\frac{5}{3}}}{62^{\frac{2}{3}}}$-approximation to MSCubeP.*

Proof. In this proof we prove that the ratio holds for every zone Z_k and therefore holds for their sum (*i.e.* for all k, $\frac{Hs(Z_k)}{3v_k^{\frac{2}{3}}} \leq \rho$ implies $\frac{\sum Hs(Z_k)}{\sum 3v_k^{\frac{2}{3}}} \leq \rho$).

We now focus on the study of general subsets of $[1, N]^3$ that are the image by H of an interval of $[1, N^3]$ (*i.e.* all the possible Z_k for every possible distribution of the $\{v_1, \ldots, v_m\}$). In order to refine the analysis we distinguish different kinds of q-cubes.

Definition 2.2. Let Z be a subset of $[1, N]^3$. A q-cube C_q is said to be *completed* by Z when $C_q \subseteq Z$. A q-cube C_q is said to be *partially completed* by Z when $C_q \cap Z \neq \varnothing$ and $C_q \nsubseteq Z$.

Definition 2.3. Let Z be a subset of $[1, N]^3$. We denote as q_Z the largest q such that there exists a q-cube completed by Z.

These two notions are the exact 3D counterpart of completed and partially-completed q-squares introduced in Section 1.5. Similar properties apply, for example for every q there are at most two partially-completed q-cubes.

Lemma 2.19. *Let Z be a subset of $[1, N]^3$ such that there exists an interval I of $[1, N^3]$ such that $H(I) = P$. Then $\forall q$ at most two q-cubes are partially completed by Z.*

Proof. Let $C_1, C_2, \ldots, C_k$ be all the q-cubes such that for $i \in [1, k]$, $C_i \cap Z \neq \varnothing$. By assumption on H, $\forall i \in [1, k]$, there exists n_i such that $C_i = H([n_i, n_i + 8^q - 1])$. Let us assume that $n_1 < n_2 < \ldots < n_k$ and let us denote $I_i = [n_i, n_i + 8^q/N - 1]$. Since H is bijective, we also know that for $i \neq j$, $n_j \notin I_i$ and, therefore, for all $x, y, z \in I_i \times I_j \times I_l$, $i < j < l$ implies $x < y < z$. At

the same time, we know that $Z = H(I)$ where I is an interval of $[1, N^3]$. By assumption we know there exist x_1 and x_k such that $x_1 \in I_1 \cap I$ and $x_k \in I_k \cap I$. We know that $x_1 < x_k$ and $x_1, x_k \in I$, so that $[x_1, x_k] \subseteq I$. In addition, as $\forall y \in I_j$, $1 < j < k$, $x_1 < y < x_k$, and then $y \in [x_1, x_k]$ what implies $y \in I$. We deduce that for all $j \in]1, k[$, $C_j \subseteq H(I) = Z$ and therefore C_j is completed by A. So we have proved that only C_1 and C_k can be partially completed by Z and so there are at most two partially completed q-cubes. □

A direct consequence of Lemma 2.19 is that we can bound the number of completed q_Z-cubes by 14.

Lemma 2.20. *Let Z be a subset of $[1, N]^3$ such that there exists an interval I of $[1, N^3]$ such that $H(I) = Z$. Then there are at most 14 completed q_Z-cubes and at most 2 partially completed q_Z-cubes and they fit into at most two $(q_Z + 1)$-cubes.*

Proof. The second statement is a direct corollary of Lemma 2.19. Indeed, if Z fits in more than two $(q_Z + 1)$-cubes, then one of them, at least, must be completed, which is a contradiction with the definition of q_Z.

In order to prove the first statement, we can simply notice than a $(q + 1)$-cube consists of exactly 8 q-cubes. Let us now assume that Z has at least 15 completed q_Z-cubes. Then the only possible repartition of these cubes in the two (q_Z+1)-cubes which contains them is 7+8 and one of the two (q_Z+1)-cubes contain 8 completed q_Z-cubes, therefore this one is also completed, which is in contradiction with the definition of q_Z. □

The key point in our proof are the partially completed q_Z-cubes. Thanks to Lemma 2.20 we know that they are at most two. Let $C_{1,Z}$ and $C_{2,Z}$ be the two q_Z-cubes partially completed by Z. Let us consider the following values: let n_Z be the number of completed q_Z-cubes and let $v_{i,Z} = \frac{|C_{i,Z} \cap Z|}{8^{q_Z}}$, $i \in \{1, 2\}$. The $v_{i,Z}$ represent the fractions of $C_{1,Z}$ and $C_{2,Z}$ which are intersected by Z (both are in $[0, 1[$). We have $|Z| = 2^{3q_Z}(n_Z + v_{1,Z} + v_{2,Z})$.

The size of $Cu(Z)$, and the values $\pi_1(Z)$, $\pi_2(Z)$ and $\pi_3(Z)$ are more difficult to estimate. Let us first define $S_Z = \frac{Hs(Z \setminus (C_{1,Z} \cup C_{2,Z}))}{4^{q_Z}}$, the scaled half surface of the covering truncated cuboid of the volume defined by the q_Z-cubes completed by Z.

Let us now define

$$s_{1,Z} = \frac{Hs(Z \setminus C_{2,Z}) - Hs(Z \setminus (C_{1,Z} \cup C_{2,Z}))}{4^{q_Z}}$$

and

$$s_{2,Z} = \frac{Hs(Z \setminus C_{1,Z}) - Hs(Z \setminus (C_{1,Z} \cup C_{2,Z}))}{4^{q_Z}}.$$

These two values are the portion of $Hs(Z)$ that come exclusively from partially completed q_Z-cubes. However, there are cases where the $s_{i,Z}$ refer to the same

data, therefore we have only an inequality (that is enough for the bound we are looking for). Hence $Hs(Z) \leq 4_P^q(S_Z + s_{1,Z} + s_{2,Z})$.

Therefore,

$$\rho_Z = \frac{Hs(Z)}{3|Z|^{2/3}} \leq \frac{4^{q_Z}(S_Z + s_{1,Z} + s_{2,Z})}{3(2^{3q_Z}(n_Z + v_{1,Z} + v_{2,Z}))^{2/3}} \leq \frac{S_Z + s_{1,Z} + s_{2,Z}}{3(n_Z + v_{1,Z} + v_{2,Z})^{2/3}}.$$

We can refine this upper bound. Let

$$x_{1,i,Z} = \frac{\pi_i(Z \setminus C_{2,Z}) - \pi_i(Z \setminus (C_{1,Z} \cup C_{2,Z}))}{4^{q_Z}}$$

and

$$x_{2,i,Z} = \frac{\pi_i(Z \setminus C_{1,Z}) - \pi_i(Z \setminus (C_{1,Z} \cup C_{2,Z}))}{4^{q_Z}}.$$

Therefore $s_{i,Z} = x_{i,1,Z} + x_{i,2,Z} + x_{i,3,Z}$. However, we know that C_1 precedes one of the other q_Z-cubes of Z and this q_Z-cube cannot be C_2 by definition of q_Z. Thus, and according to Corollary 2.17, there exists C' in $Z \setminus (C_1 \cup C_2)$ such that $\Pi_1(C_1) = \Pi_1(C')$ or $\Pi_2(C_1) = \Pi_2(C')$ or $\Pi_3(C_1) = \Pi_3(C')$. Hence $\Pi_1(Z \setminus C_{2,Z}) \subset \Pi_1(Z \setminus (C_{1,Z} \cup C_{2,Z}))$ or $\Pi_2(Z \setminus C_{2,Z}) \subset \Pi_2(Z \setminus (C_{1,Z} \cup C_{2,Z}))$ or $\Pi_3(Z \setminus C_{2,Z}) \subset \Pi_3(Z \setminus (C_{1,Z} \cup C_{2,Z}))$. Therefore $x_{1,1,Z} = 0$, $x_{1,2,Z} = 0$ or $x_{1,3,Z} = 0$.

With a similar reasoning for the $x_{2,i,Z}$ and with $x_{1,Z} = \max(x_{1,1,Z}, x_{1,2,Z}, x_{1,3,Z})$ and $x_{2,Z} = \max(x_{2,1,Z}, x_{2,2,Z}, x_{2,3,Z})$, $s_{1,Z} \leq 2x_{1,Z}$ and $s_{2,Z} \leq 2x_{2,Z}$. Hence,

$$\rho_Z \leq \frac{S_Z + 2(x_{1,Z} + x_{2,Z})}{3(n_Z + v_{1,Z} + v_{2,Z})^{2/3}}.$$

Let us first prove a result on the relative values of n_Z and S_Z.

Lemma 2.21. $S_Z \leq 2n_Z + 1$.

Proof. Let us prove this result by induction on n_Z. If $n_Z = 1$ then $S_Z = 3$. Indeed if $n_Z = 1$ then $Z \setminus (C_{1,Z} \cup C_{2,Z})$ is a single q_Z-cube. Therefore $S_Z = \frac{3*4^{q_Z}}{4^{q_Z}} = 3$.

Let us suppose that the result holds true for $n_Z = k$ and consider $n_Z = k+1$. Let Z' be Z minus the last (in the order where the Hilbert's curve go through them) q_Z-cube of Z. $n_{Z'} = n_Z - 1 = k$. Therefore $S_{Z'} \leq 2n_{Z'} + 1$. At the same time Corollary 2.17 states that the q_Z-cube we remove shares at least one face with Z'. Therefore $S_Z - S_{Z'} \leq 2$, what leads to $S_Z \leq 2 + 2(n_Z - 1) = 2n_Z + 1$. □

Let us find a relationship between $x_{i,Z}$ and $v_{i,Z}$. Let us look for a function f_q such that $v_{i,Z} \geq f_q(x_{i,Z})$, where q depends on the size of the q-cubes we are considering (in practice $q = q_Z$). Let us consider the following definition of f_q.

$$f_q(x) = \begin{cases} 0 & \text{if } q = 0 \text{ and } x = 0 \\ \frac{3}{7} & \text{if } q = 0 \text{ and } x \neq 0 \\ \frac{1}{8} f_{q-1}(4x) & \text{if } q > 0 \text{ and } x \leq \frac{1}{4} \\ \frac{1}{8} + \frac{1}{8} f_{q-1}(4(x - \frac{1}{4})) & \text{if } q > 0 \text{ and } \frac{1}{4} < x \leq \frac{1}{2} \\ \frac{1}{4} + \frac{1}{8} f_{q-1}(4(x - \frac{1}{2})) & \text{if } q > 0 \text{ and } \frac{1}{2} < x \leq \frac{3}{4} \\ \frac{3}{8} + \frac{1}{8} f_{q-1}(4(x - \frac{3}{4})) & \text{if } q > 0 \text{ and } \frac{3}{4} < x \end{cases} \tag{2.2}$$

Before any other consideration, let us prove two important lemmas dealing with f_q. First Lemma 2.22 states that f_q is an increasing function that can be bounded.

Lemma 2.22. *$\forall q$, f_q is an increasing function on $[0,1]$ and $\forall x \in [0,1]$, $\frac{3}{7}x^{\frac{3}{2}} \leq f_q(x) \leq \frac{3}{7}$.*

Proof. Let us first prove that $\forall x \in [0,1]$, $\frac{3}{7}x^{\frac{3}{2}} \leq f_q(x) \leq \frac{3}{7}$. We prove this claim by induction on q.

If $q = 0$ then there are two cases. If $x = 0$ then $f_q(x) = 0$ and $\frac{3}{7}0^{\frac{3}{2}} = 0 = f_q(x) \leq \frac{3}{7}$. Else $f_q(x) = \frac{3}{7}$ and then $\frac{3}{7}x^{\frac{3}{2}} \leq \frac{3}{7} = f_q(x) \leq \frac{3}{7}$ what achieves the proof.

Let us now assume that property holds true for $q-1$. Let x be in $]\frac{k}{4}, \frac{k+1}{4}]$ with $k \in [0,3]$. In this case $f_q(x) = \frac{k}{8} + \frac{1}{8} f_{q-1}(4(x - \frac{k}{4}))$. Therefore,

$$f_q(x) = \frac{k}{8} + \frac{1}{8} f_{q-1}(4(x - \frac{k}{4})) \leq \frac{3}{8} + \frac{3}{56} = \frac{3}{7}.$$

Similarly,

$$f_q(x) \geq \frac{k}{8} + \frac{3(4x-k)^{\frac{3}{2}}}{8 \times 7} \geq \frac{k}{8} + \frac{3(x - \frac{k}{4})^{\frac{3}{2}}}{7}.$$

Let us consider the function $g_k(x) = \frac{k}{8} + \frac{3(x-\frac{k}{4})^{\frac{3}{2}}}{7} - \frac{3}{7}x^{\frac{3}{2}}$. $g_k'(x) = \frac{9}{14}(\sqrt{x - \frac{k}{4}} - \sqrt{x})$. As $x \geq \frac{k}{4}$, $(\sqrt{x - \frac{k}{4}} - \sqrt{x}) \leq 0$ and therefore g_k is decreasing on $]\frac{k}{4}, \frac{k+1}{4}]$. Hence $g_k(x) \geq g_k(\frac{k+1}{4}) = \frac{k}{8} + \frac{3(1-(k+1)^{\frac{3}{2}})}{56}$. One can prove that the function $h : y \mapsto \frac{y}{8} + \frac{3(1-(y+1)^{\frac{3}{2}})}{56}$ reaches its lower bound on $[0,3]$ for $y = 0$ and $y = 3$ and $h(0) = h(3) = 0$. Therefore $\forall k \in [0,3]$ and $\forall x \in]\frac{k}{4}, \frac{k+1}{4}]$,

$$f_q(x) - \frac{3}{7}x^{\frac{3}{2}} = g_k(x) \geq h(k) \geq 0.$$

We just proved that $\forall x \in [0,1]$ $\frac{3}{7}x^{\frac{3}{2}} \leq f_q(x) \leq \frac{3}{7}$. By induction this property holds for every q.

Let us now prove that f_q is an increasing function. We again prove this claim by induction on q. For $q = 0$ the result is trivial.

Let us now suppose that f_{q-1} is an increasing function on $[0, 1]$. Let $k \in [0, 3]$. If $x \leq y$ and $x, y \in [\frac{k}{4}, \frac{k+1}{4}[$, then

$$f_q(x) = \frac{k}{8} + \frac{1}{8} f_{q-1}(4(x - \frac{k}{4})) \leq \frac{k}{8} + \frac{1}{8} f_{q-1}(4(y - \frac{k}{4})) = f_q(y).$$

In the other cases, if $x \leq y$ and there are $k, k' \in [0, 3]$ such that $k < k'$ and $x \in [\frac{k}{4}, \frac{k+1}{4}[$ and $y \in [\frac{k'}{4}, \frac{k'+1}{4}[$, then,

$$f_q(x) = \frac{k}{8} + \frac{1}{8} f_{q-1}(4(x - \frac{k}{4})) \leq \frac{k+1}{8} \leq \frac{k'}{8} \leq \frac{k'}{8} + \frac{1}{8} f_{q-1}(4(y - \frac{k'}{4})) = f_q(y).$$

Therefore our property holds true for q and by induction we prove that f_q is an increasing function for every q.

□

Secondly f_q enables to bound $v_{i,Z}$ by a function of $x_{i,Z}$, as stated in the following lemma.

Lemma 2.23. *Let C_q be a q-cube. Let Z be a connected subset of $[1, N]^3$ such that $Z \cap C_q \neq \varnothing$, $Z = H(I)$ and $Z \nsubseteq C_q$. Let $x = \frac{\pi_1(Z \cap C_q)}{4^q}$, $y = \frac{\pi_2(Z \cap C_q)}{4^q}$, $z = \frac{\pi_3(Z \cap C_q)}{4^q}$ and $v = \frac{|Z \cap C_q|}{8^q}$. Then $v \geq f_q(x)$, $v \geq f_q(y)$ and $v \geq f_q(z)$.*

Proof. Let us prove this lemma by induction on q. If $q = 0$,then $x = 1$, $y = 1$, $z = 1$ and $v = 1$ (0-cubes have a volume of exactly one, so either they are completed or they are not intersected at all). As $f_0(1) = \frac{3}{7}$, then $v \geq f_0(x)$, $v \geq f_0(y)$ and $f_0(z)$.

Let us suppose that the property holds true for $q - 1$. Let us consider our q-cube as a subdivision of eight $(q-1)$-cubes. By assumption, and because of Lemma 2.19, we know that at most one of these eight cubes can be partially completed (the assumption $Z \nsubseteq C_q$ ensures that if there are two q-cubes partially completed by Z, the second one is outside of C_q). Therefore our original q-cube is composed of $n \in [0, 8]$ completed $(q-1)$-cubes plus one or zero partially completed $(q-1)$-cube.

- First of all, if $n \geq 4$ then $v \geq \frac{1}{2}$ as more than half of the q-cube is completed. In addition $\frac{1}{2} \geq \frac{3}{7} \geq f_q(x)$ for all x according to Lemma 2.22. Therefore $v \geq f_q(x)$, $v \geq f_q(y)$ and $v \geq f_q(z)$ is true.

- If $n = 0$, then Z is contained in only one partially completed $(q-1)$-cube. We denote it by C_{q-1}. In addition, we have $x \leq \frac{1}{4}$. Let x' (respectively y' and z') be equal to $\frac{\pi_1(Z \cap C_{q-1})}{4^{q-1}}$ (respectively to $\frac{\pi_2(Z \cap C_{q-1})}{4^{q-1}}$ and $\frac{\pi_3(Z \cap C_{q-1})}{4^{q-1}}$) and $v' = \frac{|Z \cap C_{q-1}|}{8^{q-1}}$. Then,

$$v' = \frac{|Z \cap C_{q-1}|}{8^{q-1}} = \frac{|Z \cap C_q|}{8^{q-1}} = 8v$$

and

$$x' = \frac{\pi_1(Z \cap C_{q-1})}{4^{q-1}} = \frac{\pi_1(Z \cap C_q)}{4^{q-1}} = 4x.$$

Therefore, by assumption,

$$v = \frac{1}{8}v' \geq \frac{1}{8}f_{q-1}(x') = \frac{1}{8}f_{q-1}(4x) = f_q(x)$$

and similarly we prove that $v \geq f_q(y)$ and $v \geq f_q(z)$.

- If $n = 1,\ 2$ or 3, let C_{q-1} be the additional partially completed $(q-1)$-cubes and let Z' be the fraction of Z in C_{q-1} (we suppose $Z' \neq \emptyset$, the reverse can be interpreted as the extremal case with $n-1$ completed $(q-1)$-cubes). Then $v = n \times 8^q + V(Z')$. If we denote $v' = \frac{|Z'|}{8^{q-1}} = \frac{|Z \cap C_{q-1}|}{8^{q-1}}$ then $v = \frac{n}{8} + \frac{v'}{8}$. Similarly, if $x' = \frac{\pi_1(Z')}{4^{q-1}} = \frac{\pi_1(Z \cap C_{q-1})}{4^{q-1}}$ (similar definition for y' and z') and $x'' = \frac{\pi_1(Z \setminus Z')}{4^q}$, then Lemma 2.16 leads to $x \leq \frac{x'}{4} + x''$. $Z \setminus Z'$ consists exactly in n completed $(q-1)$-cubes, therefore, according to Lemma 2.16, $\pi_1(Z' \setminus Z) \leq n \times 4^{q-1}$ and $x'' \leq \frac{n}{4}$. As $0 < x' \leq 1$, $\frac{x'}{4} + \frac{n}{4} \in]\frac{n}{4}, \frac{n+1}{4}]$, then,

$$f_q(\frac{x'}{4} + \frac{n}{4}) = \frac{n}{8} + \frac{1}{8}f_{q-1}(4(\frac{x'}{4} + \frac{n}{4} - \frac{n}{4})) = \frac{n}{8} + \frac{1}{8}f_{q-1}(x').$$

Since f_{q-1} is increasing (Lemma 2.22), then $f_q(x) \leq f_q(\frac{x'}{4} + \frac{n}{4}) = \frac{n}{8} + \frac{1}{8}f_{q-1}(x')$.

By application of the induction assumption $f_{q-1}(x') \leq v'$ we can conclude

$$f_q(x) \leq \frac{n}{8} + \frac{1}{8}f_{q-1}(x') \leq \frac{n}{8} + \frac{v'}{8} = v.$$

Similarly we prove the property for y and z and by induction for every q.

□

An illustration of the intuition behind the definition of f_q can be seen on Figure 2.11. On Figure 2.11(a), we are in the first case, $x \leq \frac{1}{4}$. One can see that the volume to consider is the dashed one. We can even go further and prove that only the intersection with one of the two cubes is to be considered since we are searching for a lower bound. Therefore, the area we have to consider is only contained in one of the eight sub-cube and, after re-scaling x by multiplying it by four, we obtain this bound by using f_{q-1}.

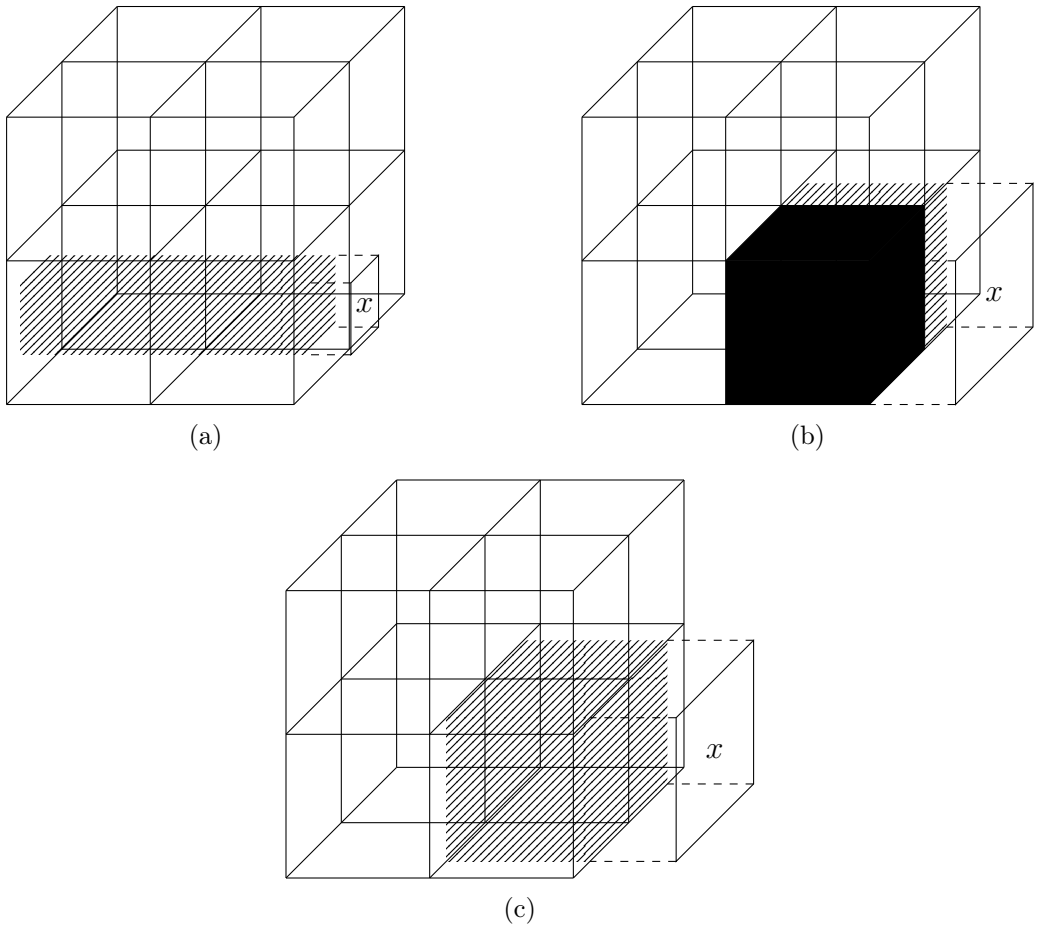


Figure 2.11: A schema to understand the definition of f_q.

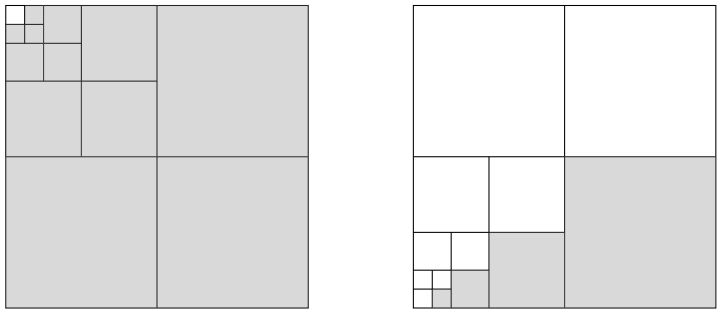
The case where $x > \frac{1}{4}$, is illustrated on Figure 2.11(b) (in the particular case of $x \leq \frac{1}{2}$). For the same reasons than before, we can only consider the right hand side of the cube. As $\frac{1}{4} < x$, we know that there must be at least two cubes intersecting Z (else $x \leq \frac{1}{4}$).

We know that one of them is completed and that the case presented on Figure 2.11(c) (two partial $(q_Z - 1)$-cubes) cannot happen. Indeed thanks to Property 2.2 we know that the path of the curve goes to the $(q_Z - 1)$-cubes one after another. In addition we know that there exists a completed q_Z-cube

just before and furthermore a completed $(q_Z - 1)$-cube. Therefore after this completed $(q_Z - 1) - cube$, if the intersection between the curve and Z goes through two $(q_Z - 1)$-cubes the first of the two must be completed.

If we go back to Figure 2.11(b), we now know that one of the $(q_Z - 1)$-cube is completed (represented by the black volume), so $v \geq \frac{1}{8}$, and we want a bound on the area present in the dashed region, that can be given by f_{q-1} (after a re-scaling of $x - \frac{1}{4}$).

The intuition behind $f_0(1) = \frac{3}{7}$ is that $\sum_{k=1}^{+\infty} \frac{3}{8^k} = \frac{3}{7}$. Indeed, to obtain the smallest v such that x can be equal to 1, we notice that this asymptotically occurs when Z is made of a sequence of cubes whose projections are as in Figure 2.12(a) on the face of area x (three competed q-cubes, then three completed $(q-1)$-cubes and so goes on) and which projection on the other faces is similar to Figure 2.12(b). In this case, for a given q, one can observe that $v = \sum_{k=1}^{q} \frac{3}{8^k}$ whose limit, when q increases to infinity, is $\sum_{k=1}^{+\infty} \frac{3}{8^k} = \frac{3}{7}$.



(a) (b)

Figure 2.12: An illustration of the intuition behind the definition of f_q.

Thanks to Lemmas 2.22 and 2.23, we prove that $v_{i,Z} \geq f_{q_Z}(x_{i,Z}) \geq \frac{3}{7} x_{i,Z}^{\frac{3}{2}}$. Therefore

$$\rho_Z \leq \frac{S_Z + 2(x_{1,Z} + x_{2,Z})}{3(n_Z + v_{1,Z} + v_{2,Z})^{2/3}} \leq \frac{S_Z + 2(x_{1,Z} + x_{2,Z})}{3(n_Z + \frac{3}{7}(x_{1,Z}^{\frac{3}{2}} + x_{2,Z}^{\frac{3}{2}}))^{2/3}}.$$

Let us now consider the function $g(x,y) = \frac{S_Z + 2(x+y)}{3(n_Z + \frac{3}{7}(x^{\frac{3}{2}} + y^{\frac{3}{2}}))^{2/3}}$. As stated below, $g(x,y)$ reaches its maximum on $(1,1)$ for $x, y \in [0,1]^2$.

Lemma 2.24. $\forall x, y \in [0,1]^2$, $g(x,y) \leq g(1,1)$.

Proof. Let us first recall Lemma 2.21 that states that $S_P \leq 2n_P + 1$.

One can show that $\frac{\partial f}{\partial x}(x,y) = \frac{n_Z + \frac{3}{7}(y^{\frac{3}{2}} - x^{\frac{1}{2}}(S_Z + y))}{3(n_Z + \frac{3}{7}(x^{\frac{3}{2}} y^{\frac{3}{2}}))}$. Therefore the sign of $\frac{\partial f}{\partial x}(x,y)$ is the same as $h_y(x) = n_Z + \frac{3}{7}(y^{\frac{3}{2}} - x^{\frac{1}{2}}(S_Z + y))$.

Let us suppose that $n_Z \geq 4$. Hence,

$$\begin{aligned} h_y(x) &= n_Z + \frac{3}{7}(y^{\frac{3}{2}} - x^{\frac{1}{2}}(S_Z + y)) \\ &\geq n_Z + \frac{3}{7}(y^{\frac{3}{2}} - (S_Z + y)) && (x \leq 1) \\ &\geq n_Z - \frac{3}{7}S_Z + \frac{3}{7}(y^{\frac{3}{2}} - y) \\ &\geq \frac{1}{7}n_Z - \frac{3}{7} + \frac{3}{7}(y^{\frac{3}{2}} - y) && (S_Z \leq 2n_Z + 1) \\ &\geq \frac{1}{7} + \frac{3}{7}(y^{\frac{3}{2}} - y). \end{aligned}$$

One can prove that the function $y \mapsto \frac{1}{7} + \frac{3}{7}(y^{\frac{3}{2}} - y)$ is larger than $\frac{5}{63} \geq 0$. Thus, in this case $h_y(x) \geq 0$ and then $\frac{\partial f}{\partial x}(x, y)$ is an increasing function with x. Therefore $g(x, y) \leq g(1, y)\ \forall y$ and similarly we prove $g(x, y) \leq g(x, 1)$ for all x and conclude that $g(x, y) \leq g(1, 1)$.

If $n_Z \leq 3$, let us remark that $h_y(x)$ is a decreasing function, and then g is a convex function in x. Hence $g(x, y) \leq \max(g(0, y), g(1, y))\ \forall y$. Similarly we prove $g(x, y) \leq \max(g(x, 0), g(x, 1))$ for all x and conclude that $g(x, y) \leq \max(g(0, 0), g(0, 1), g(1, 0), g(1, 1)) = \max\left(\frac{S_Z}{3n_Z^{2/3}}, \frac{S_Z+2}{3(n_Z+\frac{3}{7})^{2/3}}, \frac{S_Z+4}{3(n_Z+\frac{6}{7})^{2/3}}\right)$. By an exhaustive enumeration of cases (we know the value of n_Z and the fact that S_Z is included between 3 and $2n_Z + 1$), we can prove that if $n_Z \leq 3$, then $g(x, y) \leq g(1, 1)$ (see Table 2.2).

	$\frac{S_Z}{3n_Z^{2/3}}$	$\frac{S_Z+2}{3(n_Z+\frac{3}{7})^{2/3}}$	$\frac{S_Z+4}{3(n_Z+\frac{6}{7})^{2/3}}$
$n_Z = 1,\ S_Z = 3$	1	$\frac{5}{3*(1+3/7)^{2/3}} \simeq 1.31$	$\frac{7}{3*(1+6/7)^{2/3}} \simeq 1.54$
$n_Z = 2,\ S_Z = 3$	$\frac{1}{2^{2/3}} \simeq 0.62$	$\frac{5}{3*(2+3/7)^{2/3}} \simeq 0.92$	$\frac{7}{3*(2+6/7)^{2/3}} \simeq 1.15$
$n_Z = 2,\ S_Z = 4$	$\frac{4}{3\times 2^{2/3}} \simeq 0.84$	$\frac{6}{3*(2+3/7)^{2/3}} \simeq 1.11$	$\frac{8}{3*(2+6/7)^{2/3}} \simeq 1.32$
$n_Z = 2,\ S_Z = 5$	$\frac{5}{3\times 2^{2/3}} \simeq 1.05$	$\frac{7}{3*(2+3/7)^{2/3}} \simeq 1.29$	$\frac{8}{3*(2+6/7)^{2/3}} \simeq 1.49$
$n_Z = 3,\ S_Z = 3$	$\frac{1}{3^{2/3}} \simeq 0.48$	$\frac{5}{12}(\frac{7}{3})^{2/3} \simeq 0.73$	$\frac{7^{5/3}}{27} \simeq 0.95$
$n_Z = 3,\ S_Z = 4$	$\frac{4}{3^{5/3}} \simeq 0.64$	$\frac{1}{2}(\frac{7}{3})^{2/3} \simeq 0.88$	$\frac{8\times 7^{2/3}}{27} \simeq 1.08$
$n_Z = 3,\ S_Z = 5$	$\frac{5}{3^{5/3}} \simeq 0.80$	$\frac{7}{12}(\frac{7}{3})^{2/3} \simeq 1.02$	$\frac{7^{2/3}}{3} \simeq 1.22$
$n_Z = 3,\ S_Z = 6$	$\frac{6}{3^{5/3}} \simeq 0.96$	$\frac{2}{3}(\frac{7}{3})^{2/3} \simeq 1.17$	$\frac{10\times 7^{2/3}}{27} \simeq 1.36$
$n_Z = 3,\ S_Z = 7$	$\frac{7}{3^{5/3}} \simeq 1.12$	$\frac{3}{4}(\frac{7}{3})^{2/3} \simeq 1.32$	$\frac{11\times 7^{2/3}}{27} \simeq 1.49$

Table 2.2: Values of $g(0, 0)$, $g(0, 1)$ $(= g(1, 0))$ and g(1,1) for $n_Z \leq 3$.

□

Thanks to Lemma 2.24, we know that

$$\rho_Z \leq \frac{S_Z + 4}{3(n_Z + \frac{6}{7})^{\frac{2}{3}}}.$$

Let us consider two cases: $n_Z \leq 8$ and $n_Z \geq 9$. In the first case, using Lemma 2.21, we have $\rho_Z \leq \frac{2n_Z+5}{3(n_Z+\frac{6}{7})^{\frac{2}{3}}}$. One can prove that the function $x \mapsto \frac{2x+5}{3(x+\frac{6}{7})^{\frac{2}{3}}}$ reaches its maximum on $[0, 8]$ for $x = 8$ and then

$$\rho_Z \leq \frac{21}{3(8 + \frac{6}{7})^{\frac{2}{3}}} = \frac{7^{\frac{5}{3}}}{62^{\frac{2}{3}}}.$$

For the second case, we use a less sophisticated upper bound. We know that $S_Z + 2(x_{1,Z} + x_{2,Z}) \leq 20$ and $3(n_Z + \frac{3}{7}(x^{\frac{3}{2}} + y^{\frac{3}{2}}))^{2/3} \geq 3(n_Z)^{2/3}$. Therefore $\rho_Z \leq \frac{20}{3(n_Z)^{2/3}}$ and for $n_Z \geq 9$, $\frac{20}{3(n_Z)^{2/3}} \leq \frac{20}{3\sqrt[3]{3}} \leq \frac{7^{\frac{5}{3}}}{62^{\frac{2}{3}}}$.

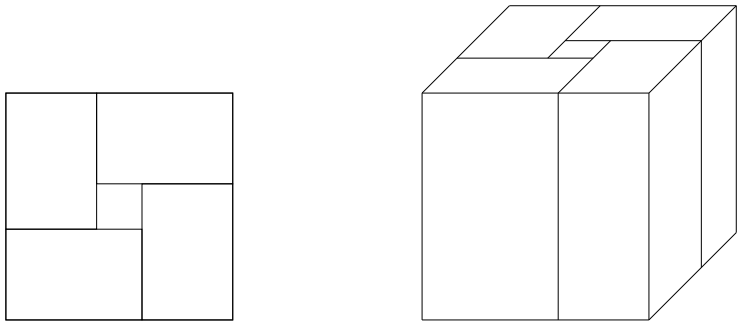
Therefore, in any case, if $Z_1, \ldots, Z_m$ is the partition of $[1, N]^3$ given by 3D-SFCP, then for all $k \in [1, m]$, $\rho_{Z_k} \leq \frac{7^{\frac{5}{3}}}{62^{\frac{5}{3}}}$ what achieves the proof of Theorem 2.18.

□

2.6 Comparison Between Square and Cube Partitioning

In Section 2.1 we observed that solutions of PERI-SUM can be transformed into solutions of MSCubeP. In this section we are interested in theoretical and practical comparisons between both.

First let us recall that the transformation of a solution of PERI-SUM into a solution of MSCubeP is obtained by replicating it alongside the third dimension, see Figure 2.13. More precisely, let $\{Z_1, \ldots, Z_m\}$ be a solution of PERI-SUM for the instance $\{s_1, \ldots, s_m\}$. We now define $\{P_1, \ldots, P_m\}$ such that for all $1 \leq k \leq m$, $P_k = Z_k \times [0, 1]$. For all k, $v(P_k) = s(Z_k) = s_k$, thus $\sum v(P_k) = 1$, and $\bigcup P_k = (\bigcup(Z_k)) \times [0, 1] = [0, 1]^3$. Therefore $\{P_1, \ldots, P_m\}$ is a valid solution of MSCubeP for the instance $\{s_1, \ldots, s_m\}$.



(a) (b)

Figure 2.13: Transformation of a PERI-SUM solution into a MSCubeP solution.

2.6.1 Theoretical Comparison

First let us try to evaluate the covering half-surface of such a partitioning. Let $\{Z_1, \ldots, Z_m\}$ and $\{P_1, \ldots, P_m\}$ be as defined above. For a given k,

$$\begin{aligned} Hs(P_k) &= \pi_1(P_k) + \pi_2(P_k) + \pi_3(P_k) \\ &= |\Pi_1(P_k)| + |\Pi_2(P_k)| + |\Pi_3(P_k)| \\ &= |\Pi_1(Z_k) \times [0,1]| + |\Pi_2(Z_k) \times [0,1]| + |\Pi_1(Z_k) \times \Pi_2(Z_k)| \\ &= \pi_1(Z_k) + \pi_2(Z_k) + s_k \\ &= p(Z_k) + s_k \end{aligned}$$

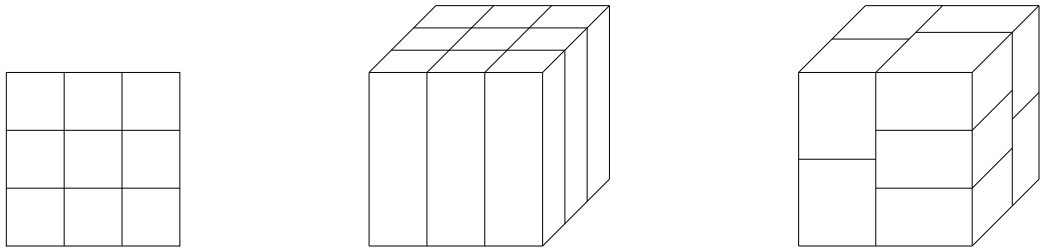
and thus

$$\sum Hs(P_k) = \sum (s_k + p(Z_k)) = 1 + \sum (p(Z_k)).$$

By reducing the number of degrees of freedom from 3 to 2, 2D-allocation algorithm (the one that solves PERI-SUM) misses more efficient allocation, independently of the quality of the 2D communication cost. For example let us consider 9 identical processors. The corresponding instance of PERI-SUM is then $\{s_1, \ldots, s_9\}$ with $s_i = \frac{1}{9}$ for all i. In this case the lower bound from (1.1) is reachable (see Figure 2.14(a)) and then $\sum_k p(Z_k) = 6$ and $\sum_k Hs(P_k) = 7$ where $\{P_1, \ldots, P_9\}$ is the 3D counterpart of this allocation (see Figure 2.14(b)). However there is an allocation $\{P'_1, \ldots, P'_9\}$ (see Figure 2.14(c)) such that $Hs(P'_k) = \frac{19}{3} < 7 = Hs(P_k)$. Thus even the best possible 2D solution they miss better communication avoiding allocations.

For the moment, let us focus on instances $\{s_1, \ldots, s_m\}$ with $s_i = \frac{1}{m}$. We know that, for every $\{Z_1, \ldots, Z_m\}$ that is a solution of PERI-SUM for instance $\{s_1, \ldots, s_m\}$,

$$\sum_{k=1}^{m} p(Z_k) \geq \sum_{k=1}^{m} \sqrt{s_k} = \sum_{k=1}^{m} 2\sqrt{\frac{1}{m}} = 2\sqrt{m}.$$



(a) (b) (c)

Figure 2.14: Solutions of $\{\frac{1}{9}, \frac{1}{9}, \frac{1}{9}, \frac{1}{9}, \frac{1}{9}, \frac{1}{9}, \frac{1}{9}, \frac{1}{9}, \frac{1}{9}\}$

Meanwhile, using 3D-NRRP, there exists a solution $\{P_1, \ldots, P_m\}$ for $\{s_1, \ldots, s_m\}$ of MSCubeP such that

$$\sum_{k=1}^{m} Hs(Z_k) \leq \alpha \sum_{k=1}^{m} 3s_k^{\frac{2}{3}} = \alpha \sum_{k=1}^{m} 3\left(\frac{1}{m}\right)^{\frac{2}{3}} = 3\alpha\sqrt[3]{m}$$

where α denotes the approximation ratio.

Thus the communication cost of the best possible solution of MSCubeP obtained from PERI-SUM is larger than $1 + 2\sqrt{m}$ while there is a solution with a cost less than $3\alpha\sqrt[3]{m}$. Hence

$$\frac{\text{Communication with 2-dimension allocation}}{\text{Communication with 3-dimension allocation}} \geq \frac{1 + 2\sqrt{m}}{3\alpha\sqrt[3]{m}} = O(\sqrt[6]{m})$$

and this ratio goes to infinity with the number of processors. Therefore, if we define Transform(AlgoPERI-SUM,$\{s_1, \ldots, s_m\}$) the algorithm that applies AlgoPERI-SUM to $\{s_1, \ldots, s_m\}$ and then transforms the output into a solution of MSCubeP, then there is no constant approximation ratio for Transform.

Theorem 2.25. *For any algorithm A that solves PERI-SUM, there is no constant α such that Transform(A) is a α-approximation for MSCubeP.*

2.6.2 Simulation Comparison

In order to compare solutions of MSCubeP with solution of PERI-SUM on other distributions, we propose a few simulations.

First we propose two general distributions:

- Uniform distribution: for a given x, Uniform-x returns a real in $[1 - x, 1 + x]$.
- Pareto distribution: for a given $\alpha \geq 1$, Pareto-α returns a real X such that $Pr(X > x) = x^{-\alpha}$. Pareto distribution is a very heterogeneous distribution, with few large values and many small ones. On Figure 2.15 are

given 3 examples of the corresponding repartition functions (the probability for a given x that a probabilistic variable following this distribution is lower than x).

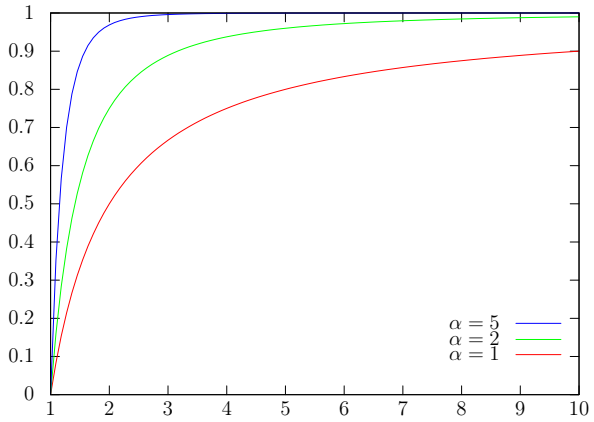
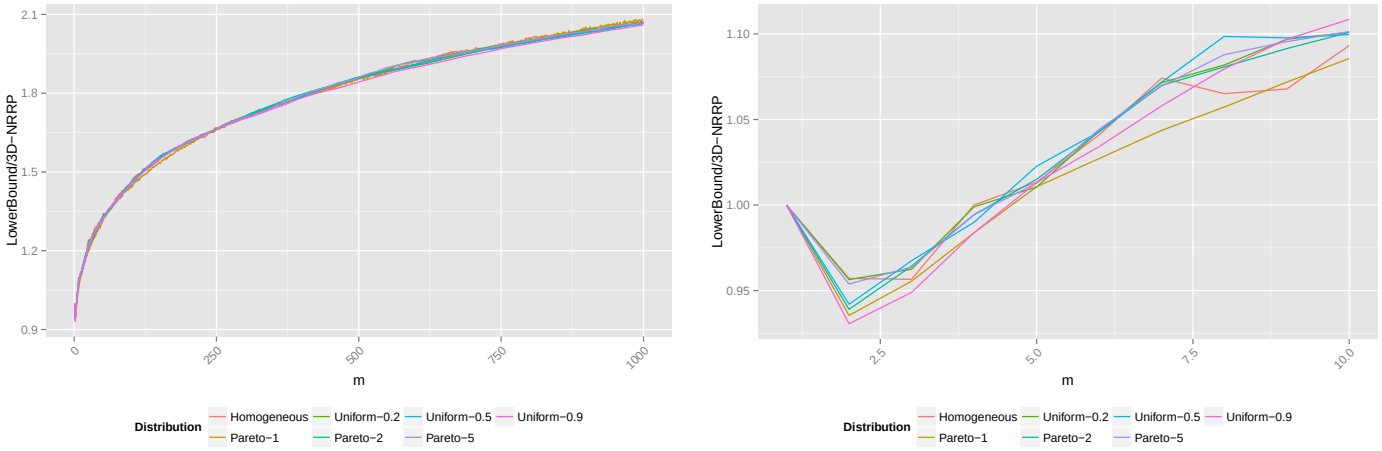


Figure 2.15: Repartition function of the Pareto distribution for different values of the parameter α.

With these two distributions, we emulate two cases, one with similar processors but with more of less small differences (Uniform distribution) and another one with a heterogeneous platform with few faster processors (for example GPUs) and many slower processors (for example CPUs).



(a) $1 \leq m \leq 1000$ (b) $1 \leq m \leq 10$

Figure 2.16: Average ratio between the results of 3D-NRRP and the lower bound of Transform for different values of m and different distributions.

Let us first compare 3D-NRRP (SCR returns close results with larger computation time) with the lower bound of Transform, *i.e.* 1 plus the lower bound

of PERI-SUM. For each distribution and each m (where m denotes the number of processors) 3D-NRRP is run on 100 instances. Results are depicted in Figure 2.16.

In general, the distribution seems to have little effect on the gain from 3D-NRRP in comparison to 2D-based strategies with an asymptotic point of view. In Figure 2.16(a) all curves seem to be close of the one from the Homogeneous distribution which asymptotic behaviour has been proved above, *i.e.* the ratio is a $0(\sqrt[6]{m})$ function (in particular, the increase of the ratio is very strong at the beginning, going to 1 for $m = 1$ to around 1.5 for $m = 125$, and then for $m = 750$ the ratio is still below 2). However, for very small values, Transform using a perfect 2D-allocation (all zones being squares) would be, in average, better than solutions from 3D-NRRP to solve MSCubeP, see Figure 2.16(b).

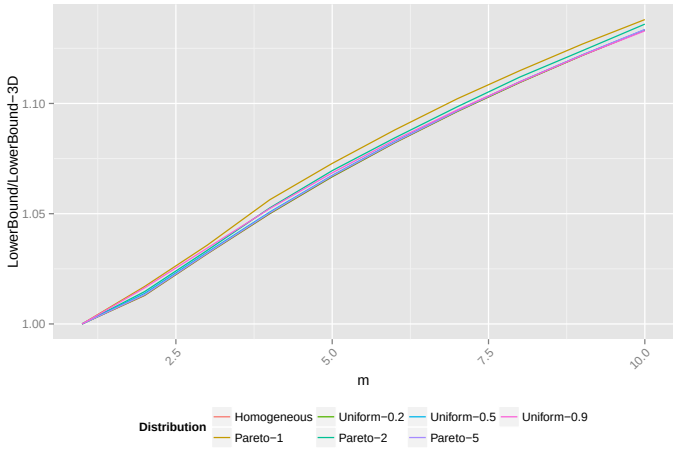
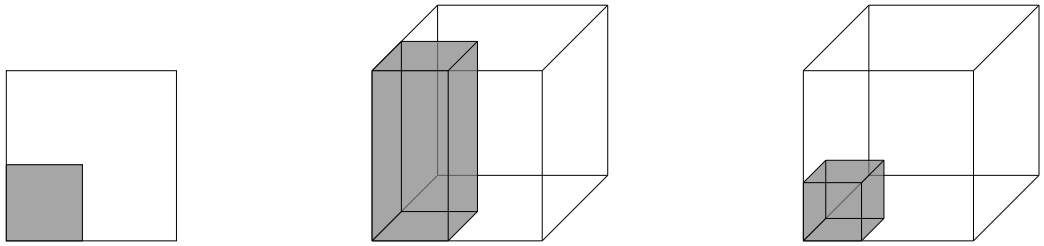


Figure 2.17: Average ratio between the results of 3D-NRRP and the result of Transform(max(ColumnBased,RRP,NRRP)) for small values of m and different distributions.

On Figure 2.17 we propose this time a comparison between 3D-NRRP and Transform(min(ColumnBased,RRP,NRRP)), *i.e.* an algorithm that returns the best solution of PERI-SUM between the three computed by the algorithms ColumnBased, RRP and NRRP. We focus only on the case where $m \leq 10$. If the results are slightly better than the one with the lower bound, there are still solutions for Transform(min(ColumnBased,RRP,NRRP)) that are better than solutions from 3D-NRRP. In particular, let us consider the example of the instance $\{\frac{1}{5}, \frac{4}{5}\}$. The solution given by NRRP is on Figure 2.18(a) (and is optimal according to Theorem 1.4) and its transformation in a solution of MSCubeP depicted in Figure 2.18(b). The half perimeter of the square of area $\frac{1}{5}$ is $\frac{2}{\sqrt{5}}$ and the half-perimeter of the remaining zone is 2. Thus the total is $2 + \frac{2}{\sqrt{5}}$, and hence the solution returned by Transform(NRRP,$\{\frac{1}{5}, \frac{4}{5}\}$) has a covering half-surface of $3 + \frac{2}{\sqrt{5}} \simeq 3.894$. The solution returned by

3D-NRRP is on Figure 2.18(c). The covering half-surface of the cube is $\frac{3}{5^{\frac{2}{3}}}$ and the covering half-surface of the remaining polyhedron is 3, for a total of $3\left(1+\frac{1}{5^{\frac{2}{3}}}\right) \simeq 4.026$, that is larger than the covering half-surface of the solution from Transform(NRRP).



(a) Solution from NRRP

(b) Solution from Transform(NRRP)

(c) Solution from 3D-NRRP

Figure 2.18: Solutions of the instance $\{\frac{1}{5}, \frac{4}{5}\}$.

The conclusion of this study is then that for large platforms the solution of MSCubeP generates, theoretically and experimentally, significantly less communication than the transformed solutions of PERI-SUM. However, for very small platforms ($m \leq 5$) this no longer holds true, and transformed solutions are really competitive.

2.7 Conclusion and Perspectives

In this chapter, we presented in Section 2.1 MSCubeP, the 3D counterpart of PERI-SUM. MSCubeP, as PERI-SUM, can be used to model communication-avoiding parallel matrix multiplication. In Section 2.2 we proposed a short presentation of pre-existing works, including SCR, an adaptation of ColumnBased for MSCubeP, and a sketch of the proof of NP-completeness for PERI-SUM. From this proof we proposed, in Section 2.3, a proof of the NP-completeness of MSCuboidP-DEC, the decision problem associated to MSCuboidP that is itself a (more general) variant of MSCubeP. Then, in Sections 2.4 and 2.5 we proposed two approximation algorithms, 3D-NRRP and 3D-SFCP, that are respectively $\frac{5}{6^{2/3}}$ and $\frac{7^{\frac{5}{3}}}{62^{\frac{2}{3}}}$-approximation algorithms for MSCubeP. Finally, in Section 2.6, we proposed a comparative analysis of the solutions of PERI-SUM with the solutions directly designed for MSCubeP, proving that the ratio between them grows to infinity when the number of processors increases.

The study of MSCubeP and its variants is not yet finished. First, as stated before, the NP-completeness of MSCubeP is still an open problem. The performance of 3D-NRRP on small instances, notably in comparison to transformed solutions of PERI-SUM, can still be improved. In addition, the perspectives

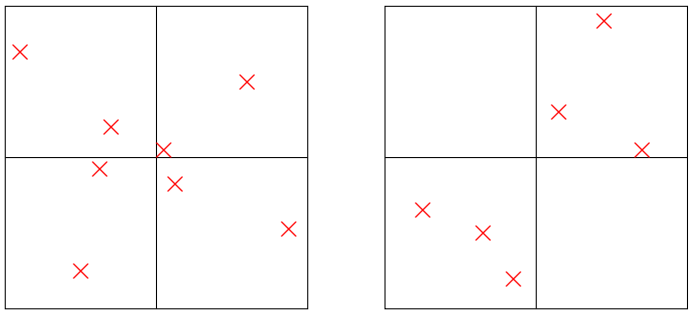


Figure 2.19: A same allocation on two different layers of a same parallel sparse matrices multiplications.

we propose in Section 1.6 can be translated to cube partitioning. In particular a heterogeneous version of MSCubeP, MSCubeP-HETEROGENEOUS, could be even more interesting, in the case of dense or low-rank matrices, than PERI-SUM. Indeed, to solve MSCubeP using a PERI-SUM solution we reproduce this partitioning alongside the third dimension. Such solution is no longer desirable. Indeed, let us assume that $A_{i,k}$ and $B_{k,j}$ are not zero while $A_{i,k'}$ or $B_{k,j'}$ are. In such case, the task $T_{i,j,k}$ has a non-zero weight while the one of $T_{i,j,k'}$ has a zero weight. Therefore, from a layer to another, the allocation might have to differ. For example, on Figure 2.19, the left hand figure presents an allocation with a perfect load-balancing between its 4 zones for one particular layer. However, the same allocation on the right hand figure has no longer this good load-balancing for a different layer. Therefore, the solutions of PERI-SUM-HETEROGENEOUS for different layers have to be different (as must be the inputs) what requires computations for each. Therefore, a direct solution of MSCubeP-HETEROGENEOUS could improve general computation time.

As we did for PERI-SUM, we can also update MSCubeP into a problem of hypercube (or hyperrectangle) partitioning, *i.e.* increase the dimension of the object to split. Recently, tensor contraction computation has become a very considered problem, see Peise *et al.* [2014], Napoli *et al.* [2014], Shi *et al.* [2016], Nelson *et al.* [2015] or Huang *et al.* [2017]. A tensor can be seen as a generalisation of a matrix in an N-dimension space. More precisely, an N-D tensor T can be defined as a multi-dimensional array of scalar elements, each dimension having a length $l_{k,T}$ (k being the index of the dimension). With this definition, an $m \times n$ matrix M would be a 2-D tensor with $l_{1,M} = m$ and $l_{2,M} = n$. In order to facilitate the description of tensor contraction we need to define the index bundles of a tensor T. An index bundle I is simply a d_T-tuple, where d_T is the dimension of the tensor T. If $I = \{t_1, \ldots, t_{d_T}\}$ is an index bundle, T_I is then in $\mathbb{R}$, more precisely the element of T placed in the t_1-th array according to the first dimension, in the t_2-th array according to the

second dimension, ..., and in the t_{T_d}-th array in the last dimension.

Let us suppose that we have 3 tensors A, B, C with $d_A + d_B = 2k + d_C$ and $k \in \mathbb{N}$. Let us also suppose that $\forall i \in [1, k],\ l_{d_A-k+i,A} = l_{i,B}$. Then if I_m, J_n and P_k are partial index bundles of respective sizes m, n, k, with $m = d_A - k$ and $n = d_B - k$, then the general tensor contraction is defined by

$$C_{I_m J_n} = \alpha \sum_{P_k \in [1,l_{1,B}] \times \ldots [1,l_{k,B}]} A_{I_m P_k} B_{P_k J_n} + \beta C_{I_m J_n}$$

with α, β being given constants and $I_m J_n$ (respectively $I_m P_k$ and $P_k J_n$) being considered as an index bundle resulting of the concatenation of I_m and J_n (respectively I_m and P_k and P_k and J_n). If we set $d_A = d_B = d_C = 2$, then $k = 1$ and the tensor contraction is the exact definition of the general matrix multiplication (GEMM) defined as

$$C_{i,j} = \alpha \sum A_{i,l} B_{l,j} + \beta C_{i,j}$$

that is, with $\alpha, \beta = 1$, the definition of the matrix multiplication.

There is a lot of conceptualization and modelling work to do, but a parallel tensor contraction requires thus to split a hyperrectangle (of dimension d_C if we assume that all computations done on $C_{I_m J_n}$ are done on a same processor), while minimizing the replication of the $A_{I_m P_k}$s and the $B_{P_k J_n}$.

In the following chapter we will be concerned with the practical implementation of parallel matrix multiplication with the StarPU framework. Our goal will be to rely on the results of Chapter 1 and Chapter 2 and establish the exact gain in time and communication they can bring to parallel matrix multiplication.

Chapter 3

Implementation of Square and Cuboid Partitioning with the StarPU Software

3.1 Introduction

In Chapter 1 and Chapter 2, we have considered two models of the parallel matrix multiplication, with a focus on communication issues. In both cases, in order to provide simple (but NP-complete) problems which we could efficiently tackle, we introduced several (possibly optimistic) assumptions.

Let us first recall that we are considering the parallel square matrices multiplication $C = AB$ and that we see this product as a set of tasks $T_{i,j,k}$ such that

$$T_{i,j,k} : C_{i,j} \leftarrow C_{i,j} + A_{i,k}B_{k,j}$$

where $A_{i,k}$, $B_{k,j}$ and $C_{i,j}$ are block-submatrices of A, B and C. The goal here is to schedule this set of tasks in order to reach an almost optimal makespan (by never letting any processor idle), while minimizing the number of replicates (or data transfers) of the $A_{i,k}$s, $B_{k,j}$s and $C_{i,j}$s. The transformation of this problem into PERI-SUM or MSCubeP requires several assumptions.

(i) We assume the existence of a reliable evaluation of the computation time of each task (that are here presumably equivalent) on each processor. This assumption, that is unavoidable to work on theoretical static problems, is reasonable. Indeed the basic tasks $T_{i,j,k}$s are GEMM (GEneral Matrix Multiply, an operation of type $C \leftarrow \alpha AB + \beta C$, $\alpha = 1$ and $\beta = 1$ in our case), a well-know routine that is expected to be very stable. In addition, simulations from Section 1.3 tend to show that, even when these estimations are unreliable, static algorithms can be complemented with work stealing strategies to obtain comparable or better performance than purely dynamic ones.

(ii) In our model communications are not associated with any time penalty and can always be overlapped with computations. More precisely we want to avoid them for different reasons (limitation of data transfer to avoid energy consumption, reduction of data storage cost, managing to overlap communication) but in our model the amount of communication is only a metric. Yet, data transfers are not immediate and therefore can impact the makespan. We assume that the overlapping is possible, but is this overlapping perfect ? Moreover we consider the processors as equal for this metric, but their physical arrangement and their proper characteristics make that the different transfer times and energy costs may not be equal.

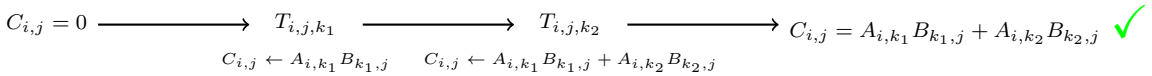
(iii) In our model, particularly for MSCubeP, we assume that the $A_{i,k}$s, $B_{k,j}$s and $C_{i,j}$s are similar. However the $A_{i,k}$s and the $B_{k,j}$s are read-only data and the $C_{i,j}$s are read-write data. This has two consequences.

First, in order to execute $T_{i,j,k}$, $A_{i,k}$ and $B_{k,j}$ only have to be downloaded while $C_{i,j}$ has to be downloaded and then uploaded when $C_{i,j}$ is no longer needed by the current processor, inducing higher communication cost.

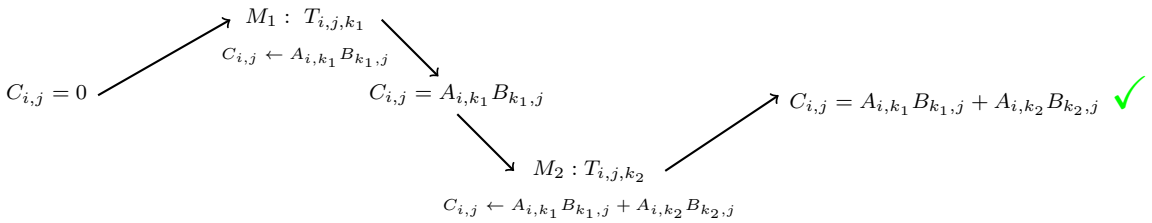
Moreover the $C_{i,j}$s create (weak) dependencies between tasks. Let us consider two tasks T_{i,j,k_1} and T_{i,j,k_2}. If both two tasks are executed sequentially, no problem occurs, $C_{i,j}$ is modified during the first task and the second task is aware of this modification during its own execution, the result at the end is as expected, see Figure 3.1(a). If they are executed on two different processors, two cases may occur. In the first case, the executions of both tasks are not overlapping. In this case, if the modification of $C_{i,j}$ is propagated on the central memory (or directly sent to the processor M_2) then the execution of the second task is aware of it and the final result is as expected, see Figure 3.1(b). Otherwise, both executions are done, at least partially, with overlapping time. In this case, at the end of the execution, each of the two processors has a different version of $C_{i,j}$. If both propagate their modifications without precaution, then the last one will overwrite the version of the first one, resulting in a false result, see Figure 3.1(c).

This conflict can be solved with two approaches. First, executions of tasks with the same pair (i, j) can be forced not to overlap, to ensure the situation of Figure 3.1(b), but the parallelism of the application is lowered. Otherwise, a reduction operation that sums the two partial versions of $C_{i,j}$ may solve the problem, see Figure 3.1(d). However this solution implies additional tasks (the reduction tasks) and thus additional computation time and data transfers. None of these solutions is perfectly satisfying. Therefore, in general, considering that the replication of the $C_{i,j}$s is not more expensive than the replication of the $A_{i,k}$s

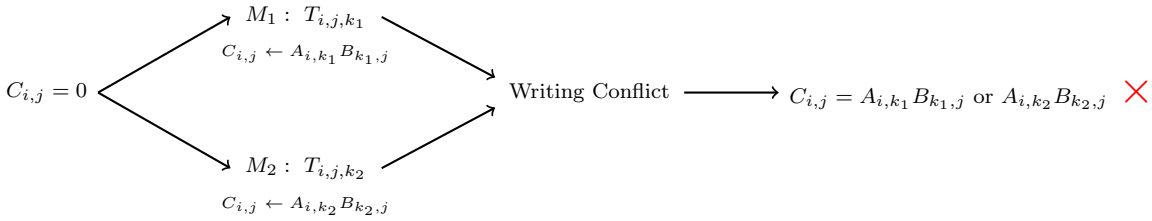
and the $B_{k,j}$s seems too optimistic and most probably false.



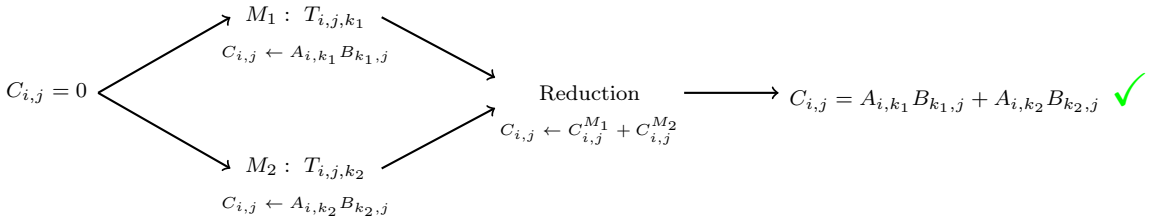
(a) Sequential execution



(b) Parallel non-overlapping execution



(c) Parallel overlapping execution without reduction



(d) Parallel overlapping execution with reduction

Figure 3.1: Execution of two tasks T_{i,j,k_1} and T_{i,j,k_2} with different modes.

Our goal in this chapter is mainly to test our algorithms in a real situation, *i.e.* to extend the work of Section 1.3 by replacing simulations with practical implementation. In particular we are concerned with the resilience of our scheduling techniques if the assumptions presented above happen to be at least partially false (we already show that considering that the replication of the $C_{i,j}$s does not cost more than the replication of the $A_{i,k}$s and the $B_{k,j}$s is not totally reasonable).

Before that, we provide a small introduction to StarPU, Augonnet *et al.* [2011], the software we use to implement our scheduler.

3.2 Presentation of StarPU

StarPU is a task programming library for hybrid architectures developed at Inria Bordeaux Sud-Ouest. For our needs we only use a fraction of the possibilities of StarPU. This section does not claim to provide a good overview of StarPU but focuses on the parts of StarPU we use in order to understand how we implement our scheduler and what difficulties we encountered.

3.2.1 Tasks

The central objects in StarPU are *tasks*. A program using StarPU usually consists in a set of tasks that are submitted to StarPU that handles the scheduling and dispatching of these tasks among available resources. We will discuss it later, but note that StarPU provides some schedulers while also allowing programmers to create their own.

The structure of a task is very simple. The main component is the *codelet* which is basically the function that is executed when the task is processed. Since StarPU targets heterogeneous environments, in particular platforms with CPUs and GPUs, a codelet is in fact composed of several functions, one for each type of processor. For example, in our case, we need to provide a GEMM function that works on CPU and a GEMM function that works on GPU. In addition, codelets can also have a *performance model*. Basically, for each execution of a given codelet on a processing node, StarPU keeps in memory basic informations in order to produce this performance model. With such information, StarPU can evaluate, if needed, the performance of this codelet on a given processor, notably to produce an estimation of the computation time.

In addition to the codelet, a task also has *arguments* and *handles*. The goal or arguments is literal. The handles are data, that are also arguments of the functions from the codelet, but that can be used by several tasks. In particular, registering an argument as a handle allows StarPU to create and manage duplicates. Note that handles can be tagged as read-only, write-only or read-write data. This precision is important for the StarPU scheduler which is based on the STF model (Sequential Task Flow, see Agullo *et al.* [2016b] for example). More precisely, StarPU creates, from the submitted tasks and the tag of each handle and each task, a dependency graph in order to avoid conflicts (StarPU assumes that a task submitted before another one must be the first to be processed if there are dependencies between the two).

3.2.2 Workers

For StarPU, each processor is a *worker*. Each worker has a type (CPU or GPU for example) and is associated to a *memory node*. A memory node is simply a

part of the memory available, that can be the main RAM or the local memory or a processor. In the platform we consider in Section 3.4, all CPUs have the same memory node (that is also the main RAM) and each GPU has its own memory node. As our algorithms (NRRP and the others) aim at reducing the number of data transfers (which are made from memory nodes to others), we chose not to distinguish two workers having the same memory node and to consider them as a single worker during the static allocation.

3.2.3 Scheduler

As stated above, StarPU provides some pre-implemented schedulers, in particular DMDA that, with the help of performance models, evaluates the computation time of the task on all workers at its submission and chooses the worker that is able to finish this task first (an algorithm similar to MCT that is presented in Section 1.3, but that also have communication considerations during the evaluation). If the programmer wants to use this option, the StarPU part of the program is then only definition of the codelets, registration of the handles, submission of the tasks and waiting until the end of task executions.

If the programmer wants to plug its own scheduler on StarPU this is possible but with respect of the general structure of a StarPU scheduler.

A StarPU scheduler consists of several functions, the two main being

- *push_task*: Called when a submitted task is ready (*i.e.* all the tasks that precede it in the dependency graph are done), mostly to put the task in a task list (that can be general or for a specific worker/memory node).
- *pop_task*: Called when a worker is idle and awaken (wake up workers can, for example, be done during push_task). It is used to choose the next task to be processed by this worker.

The task cycle of StarPU is then simple. As soon as a task has been submitted with all its dependencies solved, this task is pushed (with push_task) into the system and ready to be executed. Later, a worker will "pop" this task (with pop_task) and process it.

3.3 Implementation and Strategies

In this section we present the key points of our scheduler implementation and the different strategies we propose.

3.3.1 Static Allocations

Chapter 1 and Chapter 2 are devoted to the study of static strategies. We thus implement each of the algorithms we present before to solve PERI-SUM

and MSCubeP (ColumnBased, RRP, SNRRP, NRRP, SFCP, SCR, 3D-NRRP, 3D-SFCP) in C. Just a note on the outputs:

- In the case of PERI-SUM solvers, except for SFCP, the solution is a set of zones. Zones are defined as the union of rectangles, themselves defined by their coordinates. In practice all zones produced by ColumnBased and RRP are rectangles and the ones produced by SNRRP and NRRP are composed of at most two rectangles.
- In the case of MSCubeP solvers, except for 3D-SFCP, the solution is a set of polyhedra. This time a polyhedron is defined as the union of cuboids (at most three for 3D-NRRP).
- In the case of SFCP, the discrete aspect of the solutions makes that it is more effective to deal directly with a bi-dimensional array containing for each i, j the ID of the memory node on which, for all k, the tasks $T_{i,j,k}$ are processed. Similarly, in the case of 3D-SFCP, it is a tri-dimensional array containing for each i, j, k the ID of the memory node on which the task $T_{i,j,k}$ is computed.

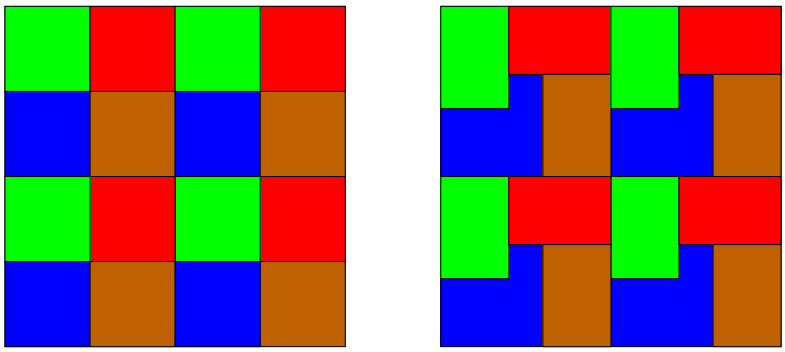
At the exception of SFCP and 3D-SFCP, the outputs of our algorithms have to be transformed to be used on an $N \times N$ matrix multiplication. The problem is here the same than the one presented in Section 1.3.2. We recall the two proposed solutions:

- Rounded routine, in this case the coordinate of each rectangle is rounded to the closest integer. The main inconvenient of this technique is the degradation of the load-balancing.
- Accurate routine, in this case the rectangles are modified using broken lines, see Figure 3.2. In this case the load balancing is saved with small degradation of the communication cost

However, if the Rounded routine is easy to generalize for any algorithm returning rectangles (or cuboids), we have currently no such generalisation for the Accurate routine. In addition the combinatorial complexity of a specific implementation makes us preferring to use another heuristic, slightly less effective for communication purpose but generalisable to any set of rectangles.

In the following, we transform the input $\{s_1, \ldots, s_m\}$ into number of tasks to process for each memory node (we group workers belonging to the same memory node as one), $\{n_1, \ldots, n_m\}$, by rounding the partial sums. More precisely, if Round is defined as the classical rounding operation (if $x - \lfloor x \rfloor < 0.5$ then $x = \lfloor x \rfloor$, else $x = \lceil x \rceil$),

$$n_1 = Round(s_1 \times N^2)$$
$$n_k = Round(N^2 \times \sum_{i=1}^{k} s_i) - \sum_{i=1}^{k-1} n_i$$



(a) Partition from RRP

(b) Partition from RRP-Accurate

Figure 3.2: Illustration of RRP-Accurate

where N is the size of the matrices (in the case of solutions of MSCubeP, replace the N^2 by N^3).

We propose Precise, a new routine that preserves load balancing. The routine goes in two waves (we describe here the 2D version for simplicity):

- First, each rectangle is reduced to its *inner rectangle*, *i.e.* the larger rectangle with integer coordinates fully contained in the initial rectangle. More precisely, if $R = [x_1, x_2] \times [y_1, y_2]$, its inner rectangle is the rectangle $R_{in} = [\lceil x_1 \rceil, \lfloor x_2 \rfloor] \times [\lceil y_1 \rceil, \lfloor y_2 \rfloor]$. Each element of an inner rectangle is assigned to the corresponding memory node and the n_is are updated ($n_i = n_i - s(R_{in,i})$).

- Second, one by one, for each unassigned element t of the matrix (*i.e.* element that are not in an inner rectangle), we check the assignment of the (up to) 8 neighbouring elements (we consider diagonals and there are fewer neighbours on the border). Then we look for a memory node with $n_i > 0$ that has been assigned at least on one of these neighbours. If at least one exists, element t is assigned to the one with smallest n_i. Otherwise the task is attributed to the memory node with the smallest positive n_i.

Precise is generalisable to MSCubeP solvers (inner rectangles become inner cuboids and the neighbourhood includes up to 26 elements instead of 8).

We now have, for each algorithm, two ways to transform the output into tasks assignment for memory nodes, Rounded and Precise. The first one produces less communication, the second preserves load balancing. Both will be tested in Section 3.4.

3.3.2 Dynamic Strategies

We consider here the dynamic strategies that we implemented within StarPU. We can distinguish two kinds: work-stealing strategies and purely dynamic strategies. The first ones complement a static allocation (turning it into a hybrid strategy, like in Section 1.3), the second ones do not rely on an initial allocation. For both, we define the cost of a task for a memory node as the number of handles to move or duplicate to execute this task on a worker that belongs to this memory node (therefore this cost can be between 0 and 3).

Work-Stealing

Work-Stealing strategies aim to fix possible errors performed during the tasks repartition. Basically, when a worker is idle with no remaining attributed tasks on its memory node, work-stealing strategies determine a task to be stolen (implying possible handle transfers) from another memory node.

We propose three strategies:

- RandSteal: Choose a memory node at random and steal the last ready task submitted on it. If there is no remaining task on this memory node, RandSteal tries the next one (in Round-Robin fashion) until a task is found or all memory nodes have been checked.
- ChoiceSteal: Check the last submitted tasks of each memory node and choose the one with the smallest cost.
- EffectiveSteal: Check all unprocessed tasks. During a first pass, EffectiveSteal stops if there is a free task (with cost 0). Otherwise, EffectiveSteal remembers the memory nodes that own a task of cost 1 or 2 and attempt to find these tasks during a second pass (they might have been processed in the meantime). If after these two passes there is still no task to steal, then RandSteal is called.

In addition to these three strategies, Static denotes a strategy without work-stealing where tasks allocation is enforced until the end.

Purely Dynamic Strategies

The strategies presented here do not rely on an initial static allocation. Each strategy is defined by a function that is called by idle workers. We implement three such strategies:

- FirstDyn: Idle workers begin the execution of the first submitted and unprocessed task.
- ChoiceDyn-X: An idle worker looks at the (at most) X first submitted and unprocessed tasks and chooses the one with the smallest cost.

- EffectiveDyn: An idle worker looks at all the submitted and unprocessed tasks and chooses the one with the smallest cost.

In addition, we also use DMDA, an already implemented task-centric dynamic scheduler based on MCT (each task, at the submission, is allocated to the worker that will complete it first). Note that unlike the MCT of Section 1.3, DMDA is communication-aware, and evaluates the data transfer duration to choose among the workers.

3.3.3 Scheduler Implementation

We now focus on some practical details of our StarPU scheduler implementation.

Prefetching

In the default settings of StarPU, the data handles of a task are loaded into a memory node just before the beginning of its execution. With such behaviour, the communications would not be overlapped with computations, what would create an important difference with our assumptions. This is why, in our implementation, we use the StarPU prefetch option that allows to begin the data transfers during the planing of a future task execution. More precisely, when a worker pops a task (to process it), this worker also begins loading the handles for a future task (in our case, when the n-th task begins, the loading of the inputs of the n+2-th task begins too). Practically, we also use work-stealing in advance, a task is not stolen when a worker has no remaining ready task but when it has not enough ones (*i.e.* strictly less than 2 in our implementation). Note that we decide to forbid the stealing of already prefetched tasks.

Reduction

In Section 3.1, we present the issue of overlapping two tasks $T_{i,j,k}$ and $T_{i,j,k'}$ that share a $C_{i,j}$. By default, $C_{i,j}$ being a read-write handle, StarPU assumes that there is a dependency between them. More precisely, if $T_{i,j,k}$ is submitted before $T_{i,j,k'}$, then push_task is called on $T_{i,j,k'}$ only once $T_{i,j,k}$ is terminated. So there cannot be an overlap of $T_{i,j,k}$ and $T_{i,j,k'}$.

In the case of 2D-strategies, as $T_{i,j,k}$ and $T_{i,j,k'}$ are scheduled on the same memory node, this is a problem only if there are many workers on the same memory node (the pool of tasks present on the task list of the memory node can be too small to satisfy all the workers, and they may begin to steal tasks from other memory nodes). In the platform we consider in Section 3.4, all CPUs are on a same memory node and thus the problem occurs. To avoid that, we use the work of Cojean *et al.* [2016] which allows to use several CPUs on a same task. More precisely, the set of CPUs is split into two subsets, each

can be considered as a single worker. This trick reduces the number of workers on this memory node to 2 and avoids the problem of early stealing. Since GEMMs are highly parallel, this has very low overhead.

In the case of 3D-strategies, the non-overlapping of $T_{i,j,k}$ and $T_{i,j,k'}$ is a real problem. Indeed, it is very common that two memory nodes share $C_{i,j}$ blocks on many (or all) their tasks. In such a case, one of the two memory node must wait for the completion of many tasks of the other memory node before beginning its own set, implying idle workers for a long time or early steals. To avoid this problem, we create reduction tasks thanks to auxiliary handles. For each $C_{i,j}$, if at least two memory nodes share this $C_{i,j}$, one $C_{i,j}^{aux}$ is created for each memory node that needs it. The use of different handles means that StarPU does not consider the tasks as having a dependency and both can be pushed simultaneously. In addition, after the submission of all GEMM tasks, we create reduction tasks, each having two handles, $C_{i,j}$ and one $C_{i,j}^{aux}$. The reduction tasks are then executed after the GEMM tasks, ensuring the coherency. However, these additional tasks also implies additional computation time and data transfers whose costs have to be evaluated.

3.4 Experimental Results

In this section we present the first experimental results from our StarPU implementation.

The tests we run are made on a "Sirocco" node of PlaFRIM2. A Sirocco node is composed of 4 GPUs and 24 CPUS (4 of which are dedicated to GPU management). More precisely the platform contains:

- 2 Dodeca-core Haswell Intel® Xeon® E5-2680 v3 @ 2.50 GHz,
- 128 GB of RAM,
- 4 Nvidia GK110BGL [Tesla K40m] (rev a1).

StarPU allows to plug several BLAS library. For our experiments, we use the MKL library for the GEMM operation on CPUs and cuBLAS library for the GEMM operation on CPUs.

As experiment set, we use matrices of double with size 7680, 15360, 23040 and 30720 (the size of the matrix A, B and C) split into square blocks of size 960 (the size of one $A_{i,k}$, $B_{k,j}$ or $C_{i,j}$). Therefore the matrices of blocks, which should be to split and distribute (with a PERI-SUM or MSCubeP solver) among memory node, have size 8, 16, 24 and 32. The size of the block is chosen to achieve efficiency for the GEMM operation on GPUs.

In this section we decide to focus on NRRP and 3D-NRRP among all the static algorithms we present in Chapter 1 and Chapter 2, the others having

similar or worst results (in addition, since the performance model and the number of memory nodes do not change between runs, the input of PERI-SUM, or MSCubeP, is always the same). Each time two versions are proposed, NRRP-Rounded and NRRP-Precise (respectively 3D-NRRP-Rounded and 3D-NRRP-Precise) and each version is tested on every dynamic strategies. In addition, each version of 3D-NRRP is used with and without reduction. Therefore, we have:

- 5 purely dynamic strategies (DMDA, FirstDyn, ChoiceDyn-10, ChoiceDyn-50, EffectiveDyn).
- 4 work-stealing strategies (Static, RandSteal, ChoiceSteal, EffectiveSteal) and 6 allocation strategies (NRRP-Rounded, NRRP-Precise, 3D-NRRP-Rounded, 3D-NRRP-Precise, 3D-NRRP-Rounded-Redux, 3D-NRRP-Precise-Redux), thus 24 hybrid strategies.

We compare these 29 different strategies on the 4 different sizes for matrices (8×8, 16×16, 24×24 and 32×32 blocks). For each configuration we perform 25 runs.

3.4.1 Trace Analysis

Before going to general results, we present some notable behaviour of our strategies. For this purpose, we use the FxT library that can be used by StarPU to produce traces that can be analysed. The goal of this section is not to compare strategies but to check the prevision we made on their behaviour (for instance on the impact of work-stealing or reduction) and to prepare explanations for the results in the two following sections.

In the following figures, the different operations can be distinguished with their colors. Mainly three of them are important:

- Green rectangles represent GEMM tasks.
- Pink rectangles represent data movement (mainly loading of $C_{i,j}$s, especially at the begin of the execution for GPUs).
- Blue rectangles represent Reduction tasks, if any.

In order to have a basis for comparison, we propose, on Figure 3.3, a trace of Static with NRRP-Rounded used as partitioning algorithm. In this trace, and all others ones, the two sets of 10 CPUs are at the top, the four GPUs are at the bottom.

Globally GEMM are scheduled one after another with little idle times. There is no notable communication, except at the beginning of the execution for the GPUs (that cannot start without an initial delay transfer of their

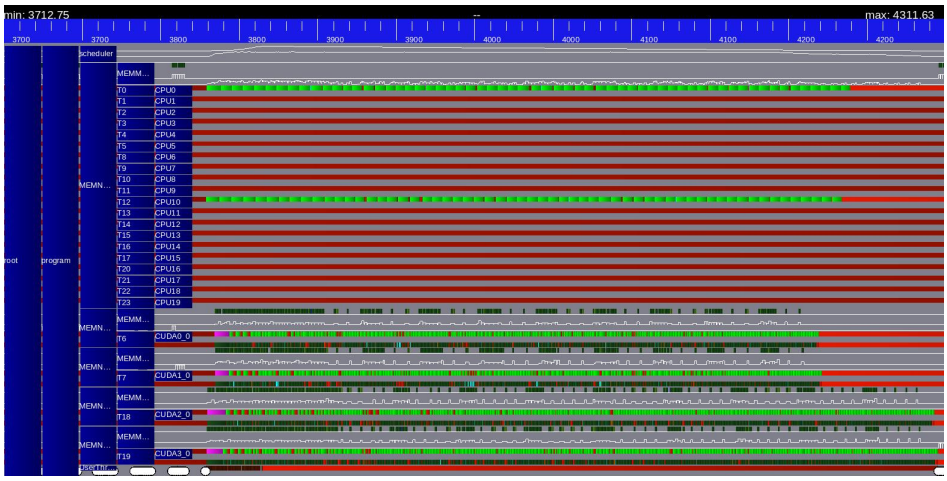


Figure 3.3: Trace of an execution with strategy Static and NRRP-Rounded allocation

$C_{i,j}$s). This means that the overlapping of the communication is done perfectly. However this trace also points the main problem with purely static strategies (especially when the static allocation exhibits bad load-balancing, as Rounded): some workers finish earlier than others. Here two of the GPUs clearly have fewer tasks to process than the two others.

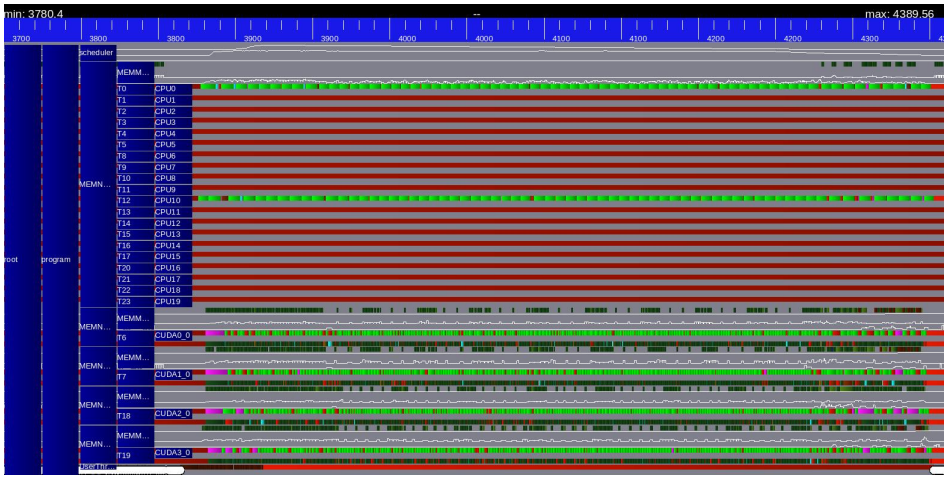


Figure 3.4: Trace of an execution with strategy RandSteal and NRRP-Rounded allocation

In Figure 3.4 we now present a version with RandSteal strategy, keeping NRRP-Rounded as partitioning algorithm. Here, all workers finish at the same time, but communications are now visible, in particular at the end (when work-stealing begins). The communications mainly come from the fact that the $C_{i,j}$s are now shared and $C_{i,j}$ transfers cannot always be overlapped with compu-

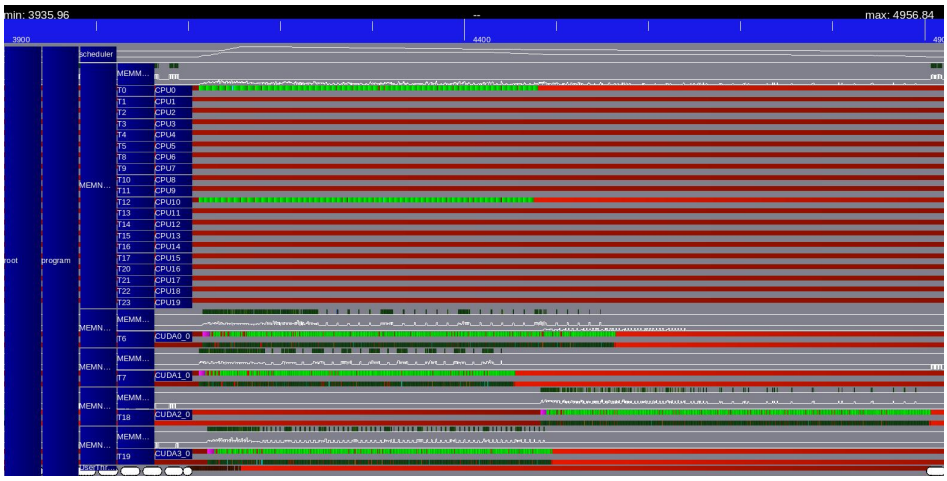


Figure 3.5: Trace of an execution with strategy Static and 3D-NRRP-Rounded allocation

tations. Therefore, RandSteal (and other work-stealing strategies) may pay for their better load-balancing with waiting time in addition to the foreseeable additional data transfers.

If we now use 3D-NRRP-Rounded instead of NRRP-Rounded, see Figure 3.5, we observe the dependency problem mentioned earlier.

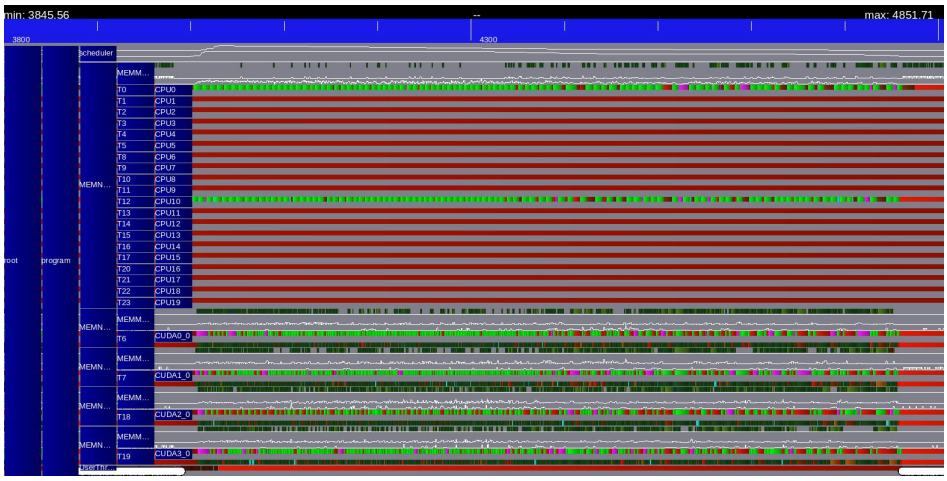


Figure 3.6: Trace of an execution with strategy RandSteal and 3D-NRRP-Rounded allocation

For example the third GPU begins processing its tasks significantly later than the other ones (and even after some have finished their tasks, probably with one of them sharing all its $C_{i,j}$s with this GPU). As stated above, without reduction tasks, StarPU generates many "wrong" dependencies between tasks sharing a $C_{i,j}$ and this implies that some workers may have to wait for the

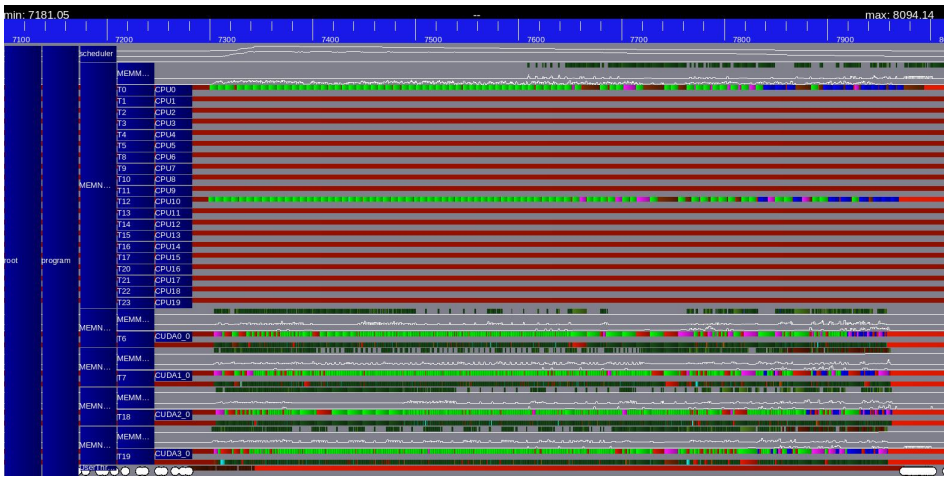


Figure 3.7: Trace of an execution with strategy RandSteal and 3D-NRRP-Rounded-Redux allocation

others to finish all their tasks before starting their own.

Note that the use of work-stealing, see Figure 3.6, improves the behaviour, but with the work-stealing happening too early (the communication operations are visible earlier than in Figure 3.4).

The use of reduction tasks, like in 3D-NRRP-Rounded-Redux can be seen on Figure 3.7, with the reduction tasks being visible after the GEMM tasks. Globally we also see that this option is not fully satisfying with additional communications (the transfers of the $C_{i,j}s$ and the $C^{aux}_{i,j}$s) and additional computation time.

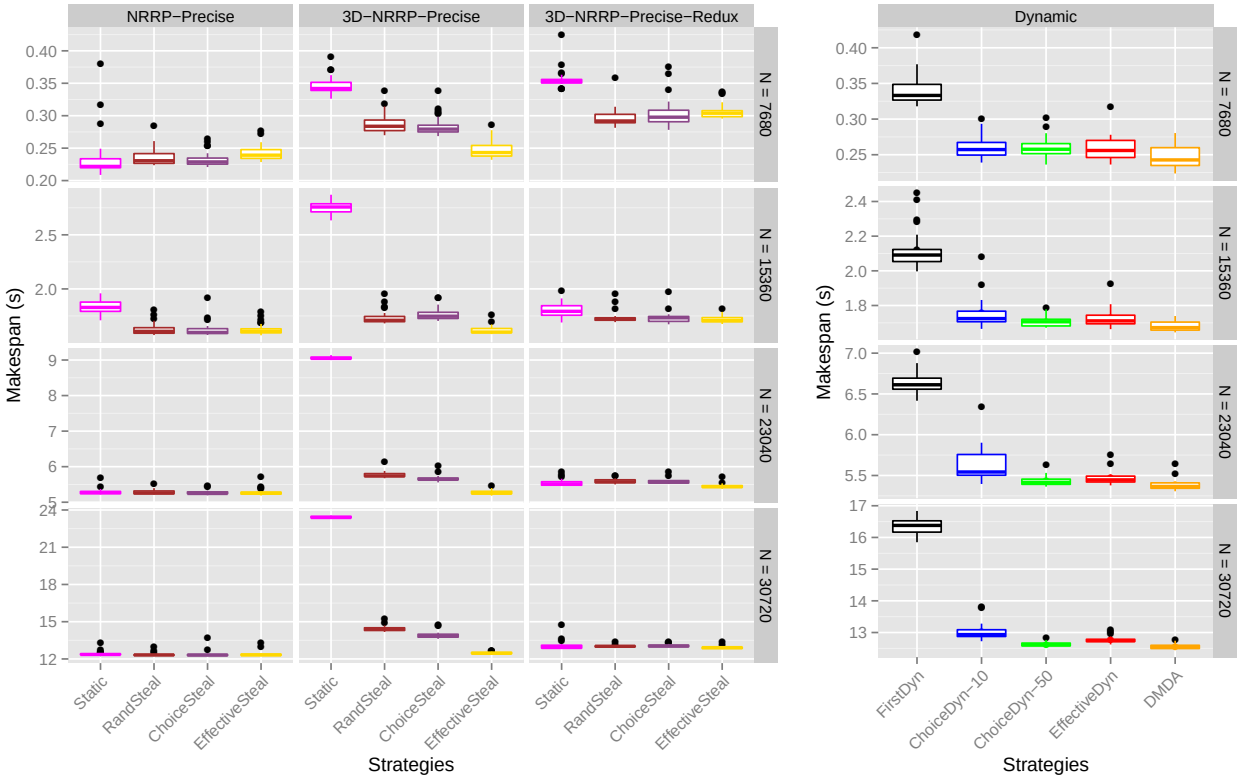
3.4.2 Makespan

Let us first consider the makespan metric, mainly to eliminate strategies that produce unsatisfying makespan. Note that in this section (and in Section 3.4.3) the y scale on every figure does not start at 0.

On Figure 3.8(a) we propose a comparison between all three partitioning algorithms, NRRP-Precise, 3D-NRRP-Precise and 3D-NRRP-Precise-Redux. Let us first note that, as expected, Static strategies with 3D-NRRP as partitioning algorithm and without reduction operation have a catastrophic makespan that gets worse when N (and the number of tasks) increases, reaching up to twice the best makespan achieved by the other allocations. The ratio 2 can be explained by the fact that two memory nodes share all their $C_{i,j}$s and thus work one after the other. For the two other strategies, NRRP-Precise and 3D-NRRP-Precise-Redux, the performance of Static strategy is much closer to the other strategies, a bit worse in some cases ($N = 15360$ for example). Note that for NRRP-Precise, Static is the best strategy for $N = 7680$, probably because of the shorter execution time of its pop_task function. Anyway, the

good results of Static on Precise partitions indicate the good quality of the performance model for the GEMM operations.

If we now look at work-stealing strategies, let us first note that EffectiveSteal is effective enough to make 3D-NRRP-Precise partitioning able to challenge the other partitioning algorithms whereas the two others, ChoiceSteal and RandSteal, improve the makespan but not enough. In other cases, there is a very small difference between all three strategies. EffectiveSteal is a bit less effective in the case $N = 7680$, a bit more in the case $N = 15360$ and there is no significant winner in the other cases. To finish, note that in general NRRP-Precise partitionings appear to be the most effective for all the values of N (around $0.225s$ against $0.25s$ and $0.3s$ for $N = 7680$, around $1.6s$ against $1.6s$ and $1.75s$ for $N = 15360$, around $5.25s$ against $5.25s$ and $5.5s$ for $N = 23040$ and around $12s$ against $12s$ and $13s$ for $N = 30720$). The better parallelism (in comparison to 3D-NRRP-Precise with no EffectiveSteal) and the absence of reduction tasks (in comparison to 3D-NRRP-Precise-Redux) generate a real advantage.



(a) Makespan of strategies using NRRP-Precise, 3D-NRRP-Precise and 3D-NRRP-Precise-Redux

(b) Makespan of purely dynamic strategies

Figure 3.8: Makespan of strategies using NRRP-Precise, 3D-NRRP-Precise and 3D-NRRP-Precise-Redux or purely dynamic strategies.

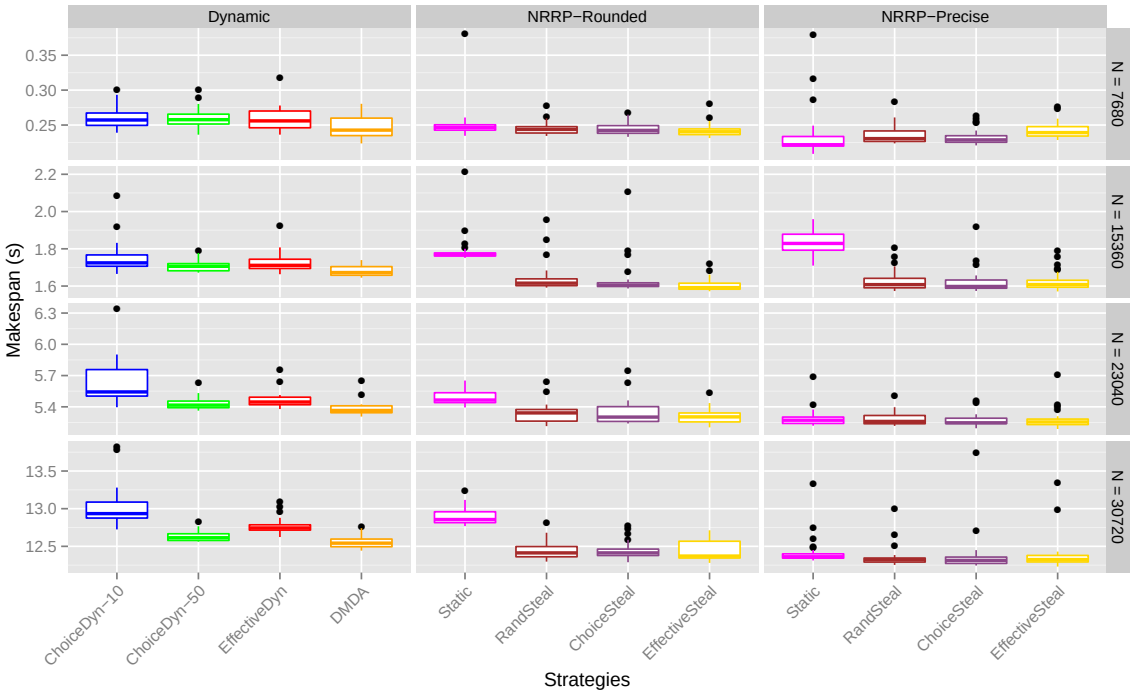


Figure 3.9: Makespan of strategies using NRRP-Precise, NRRP-Rounded and purely dynamic strategies.

In Figure 3.8(b), we present the results for dynamic strategies. This mainly emphasizes the unsatisfying results of FirstDyn that is removed on the following figure, Figure 3.9. In this one, we propose a comparison between strategies using NRRP as partitioning algorithm (the most effective according to Figure 3.8(a)) and compare both versions, NRRP-Rounded and NRRP-Precise, with purely dynamic strategies (without FirstDyn).

If we focus on purely dynamic strategies, we can mainly do three observations. First the criterion X in ChoiceDyn-X has a major influence (ChoiceDyn-10 is always worse than ChoiceDyn-50). Secondly EffectiveDyn is often less effective than ChoiceDyn-50. A possible interpretation is that stolen tasks in ChoiceDyn-50 are more or less of same quality than the ones in EffectiveDyn but are found faster. Last, DMDA appears to be more effective than purely communication-based dynamic strategies (ChoiceDyn-50 is close but slightly less effective).

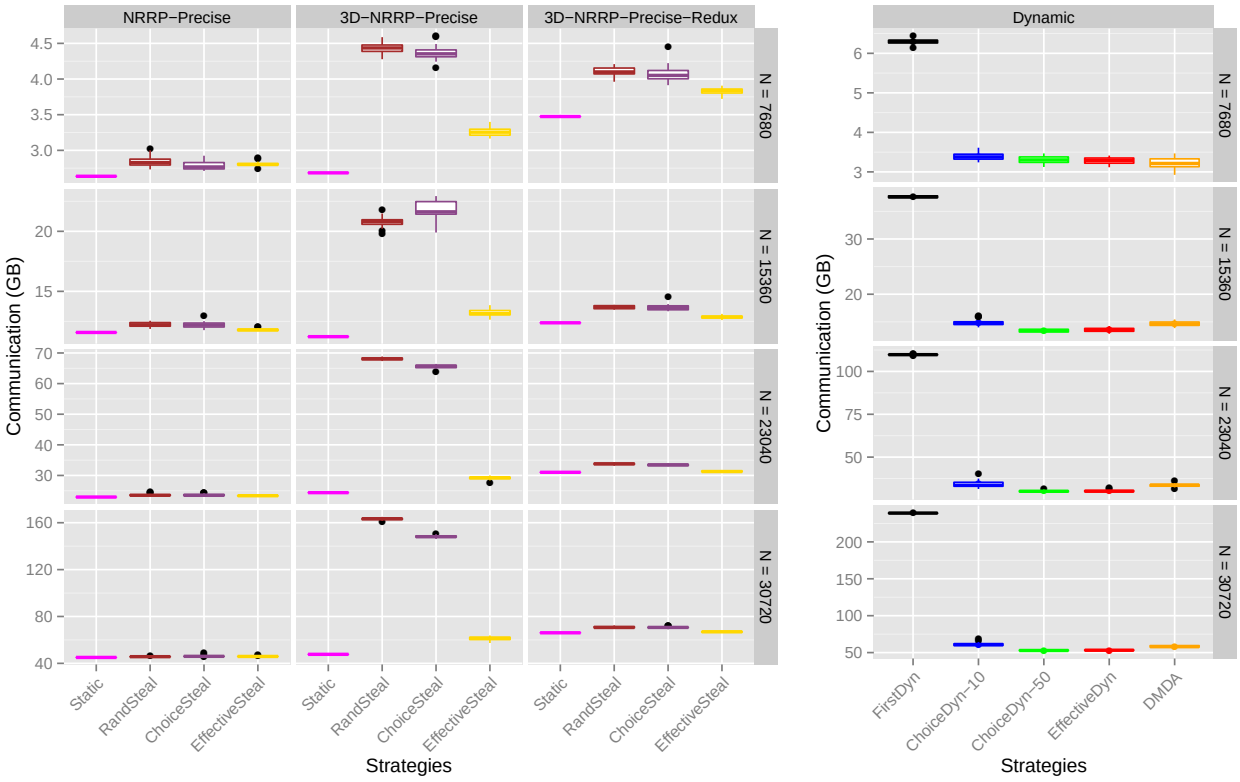
The comparison between Rounded and Precise is at the advantage of the second. In the case of Static strategies, Rounded, having a worse load-balancing, has a larger makespan except for the case $N = 15360$ (for reasons that are still to explain). For the other strategies, results are close, but Precise-based ones is slightly more stable.

To finish, static-based strategies, augmented with work-stealing, are more

makespan-effective than purely dynamic ones, which is a surprise (if we compare to results from the simulations of Section 1.3). One possible explanation is that DMDA, despite its data-aware property, induces too many data transfers, especially of $C_{i,j}$ blocks, and is thus less effective than the combination of NRRP and work-stealing strategies.

3.4.3 Communication

Let us now focus on the amount of communications induced by each strategy. We have two goals: first, determine if there is a correlation between makespan results and communication results; second determine if, with comparable makespan, some strategies are more communication-effective than others.



(a) Communication amount of strategies using NRRP-Precise, 3D-NRRP-Precise and 3D-NRRP-Precise-Redux

(b) Communication amount of purely dynamic strategies

Figure 3.10: Communication amount of strategies using NRRP-Precise, 3D-NRRP-Precise and 3D-NRRP-Precise-Redux or purely dynamic strategies.

First, we consider in Figure 3.10(a) the comparison between all three partitioning strategies, NRRP-Precise, 3D-NRRP-Precise and NRRP-Precise-Redux. If we put Static with 3D-NRRP-Precise aside, NRRP-based strategies

are significantly more effective (Static with 3D-NRRP-Precise can be slightly more communication-effective than NRRP-Precise, in the case $N = 15360$ for example, but at the cost of a bad makespan as shown in the previous section). If we compare (coarsely) the results of EffectiveSteal (that seems to be a good trade-off between makespan effectiveness and communication effectiveness) for NRRP-Precise and the second most effective, 3D-NRRP-Precise, we note that the second one can transfer up to around 33% more data, see Table 3.1.

For the other notable observations, the poor makespan results of ChoiceSteal and RandSteal when 3D-NRRP-Precise algorithm is used, in comparison to the ones of EffectiveSteal with the same algorithm, likely come from a bad choice of stolen tasks (at least one of the memory node is only functioning with work-stealing) that are here catastrophic (more than 2.5 as much data transfer as EffectiveSteal in the case $N = 30720$). Second, the reduction tasks have a clear impact on the NRRP-Precise-Redux results. A good way to evaluate the cost of reduction is to compare the cost of Static with 3D-NRRP-Precise and 3D-NRRP-Precise-Redux, see Table 3.2. With high costs (around 44% more communications for $N = 30720$), the reduction tasks have a huge impact that make the reduction option not satisfying for both metrics.

	NRRP-Precise	3D-NRRP-Precise	
$N = 7680$	$\simeq 2.75\ GB$	$\simeq 3.25\ GB$	$\simeq +18\%$
$N = 15360$	$\simeq 17\ GB$	$\simeq 18\ GB$	$\simeq +6\%$
$N = 23040$	$\simeq 24\ GB$	$\simeq 30\ GB$	$\simeq +25\%$
$N = 30720$	$\simeq 45\ GB$	$\simeq 60\ GB$	$\simeq +33\%$

Table 3.1: Comparison of the two partitioning algorithm NRRP-Precise and 3D-NRRP-Precise with use of EffectiveSteal.

	3D-NRRP-Precise	3D-NRRP-Precise-Redux	Reduction cost
$N = 7680$	$\simeq 2.75\ GB$	$\simeq 3.5\ GB$	$\simeq 0.75\ GB$ (+27%)
$N = 15360$	$\simeq 16\ GB$	$\simeq 17.5\ GB$	$\simeq 1.5\ GB$ (+9%)
$N = 23040$	$\simeq 25\ GB$	$\simeq 30\ GB$	$\simeq 5\ GB$ (+20%)
$N = 30720$	$\simeq 45\ GB$	$\simeq 65\ GB$	$\simeq 20\ GB$ (+44%)

Table 3.2: Comparison of the two partitioning algorithms NRRP-Precise and 3D-NRRP-Precise-Redux with use of Static. The difference represents the reduction cost.

Note that FirstDyn is also disappointing for both metrics, see Figure 3.10(b). FirstDyn can transfer more than 4 times as much data than the other strategies. For the rest of the dynamic strategies (more precise view on Figure 3.11), ChoiceDyn-50 and EffectiveDyn produce more or less the same amount of communications. Therefore the difference between them from the makespan

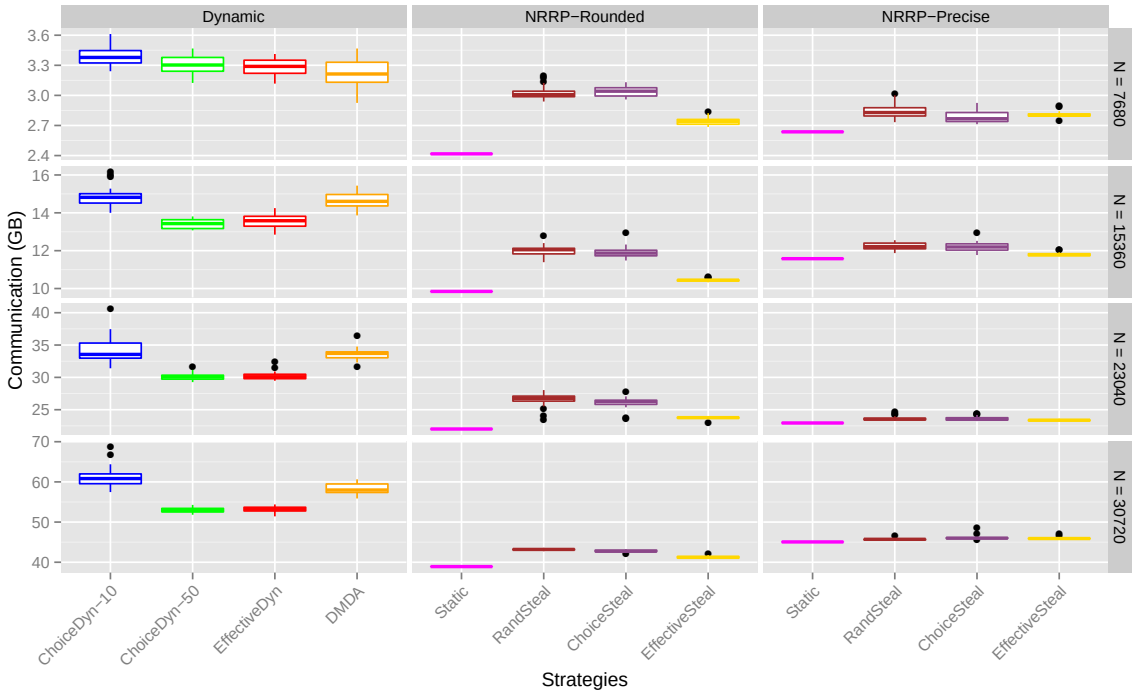


Figure 3.11: Communication amount of strategies using NRRP-Precise, NRRP-Rounded and or purely dynamic strategies.

point of view may be a consequence of ChoiceDyn being a bit faster to compute, however this assumption has to be confirmed by further investigation. Both ChoiceDyn-50 and EffectiveDyn strategies are also more communication-effective than DMDA, whose results are comparable to the ones of ChoiceDyn-10.

The comparison between Rounded and Precise is more even. Precise is expected to achieve a better load-balancing, inducing less work-stealing (that can be seen in Figure 3.11 with few differences between EffectiveSteal, ChoiceSteal, RandSteal and Static when NRRP-Precise is used), but with an increase of the communication volume, notably in the initial allocation. On Figure 3.11 we can observe that this increase can be significant. For example, in the case $N = 30720$, EffectiveSteal produces around $41GB$ of transferred data (with reasonable makespan, unlike Static) when NRRP-Rounded is used, against $45GB$ for Static with NRRP-Precise (+9.75%). However in some other cases, $N = 7980$ or $N = 23040$, NRRP-Precise performs better than NRRP-Rounded. Although the difference in these cases is less important than in the case $N = 30720$, this shows there is no clear winner between Rounded and Precise approaches (for makespan and communication metrics), the difference is probably highly correlated to the initial allocation, depending on the error coming from the rounding.

Finally, we compare dynamic and work-stealing strategies, always on Figure 3.11. We already showed that, on the makespan metric, the results of both NRRP-Rounded and NRRP-Precise are very satisfying, with a slight improvement over the native DMDA. In the case of communication metric we can highlight that the EffectiveSteal strategies provide a significant improvement with sometimes up to a 25% of reduction in amount of transferred data. Therefore the use of static allocation appears to bring a strong gain in communication with no loss of makespan-efficiency. The slightly better results for the makespan likely also comes from this important reduction of communications.

	DMDA	NRRP-Rounded	NRRP-Precise
$N = 7680$	$\simeq 3.2\ GB$	$\simeq 2.7\ GB\ (-15.6\%)$	$\simeq 2.8\ GB\ (-12.5\%)$
$N = 15360$	$\simeq 14.5\ GB$	$\simeq 10.5\ GB\ (-27.6\%)$	$\simeq 12\ GB\ (-17.2\%)$
$N = 23040$	$\simeq 33\ GB$	$\simeq 24\ GB\ (-27.2\%)$	$\simeq 23.5\ GB\ (-28.8\%)$
$N = 30720$	$\simeq 58\ GB$	$\simeq 41\ GB\ (-29.3\%)$	$\simeq 45\ GB\ (-24.1\%)$

Table 3.3: Comparison of the two partitioning algorithms NRRP-Rounded and NRRP-Precise with use of EffectiveSteal and DMDA (Between parenthesis the gain from DMDA is given).

3.5 Conclusion and Perspectives

In this chapter we focus on the practical implementation of the algorithms that we designed in Chapter 1 and Chapter 2. In Section 3.1 we recall the problem and discuss the assumptions we made during the theoretical study. Then, in Section 3.2, we present StarPU, the library we use to implement our strategies. In Section 3.3 we present these strategies, mainly based on the algorithms of Chapter 1 and Chapter 2. Finally, in Section 3.4 we present the results obtained on a heterogeneous platform. These results are very encouraging: hybrid strategies that use static algorithm, and work-stealing, slightly improve the makespan of the native algorithm while significantly improving the amount of communication, showing the strength of our approach.

Globally we also see that even if some assumptions made during the theoretical part are not perfectly true in practice (the absence of communication cost for example), the resulting hybrid algorithms are reliable on this problem. However, we identified other assumptions, especially considering that $C_{i,j}$ blocks have the same cost as $A_{i,k}$ and $B_{k,j}$ blocks in Chapter 2, that can induce important misbehaviours that are not perfectly corrected (3D-NRRP-based strategies are disappointing).

On the perspectives of this work, the first thing is to continue the analysis of the current results. We already prove the strength of our approach, although some behaviours opens up questions that may require further analysis

(some unexpected hierarchies between strategies for example) and can bring interesting information in order to improve future algorithms. In addition we have tested these strategies on one platform only. In order to have an idea of the behaviour on more general architectures, further experiments have to be done.

Among the other elements on which we have little information is the scheduling cost. We can easily evaluate the computation time of the allocation but currently we have no information on the cost of the different calls to pop_task functions. Some results seem to indicate that some strategies have a significant additional cost (for example EffectiveDyn against ChoiceDyn), but this is a by-default explanation that needs to be confirmed.

Finally, an important improvement on our model would be to add energy consumption estimation. Currently we have no such estimation, but the gain on communications gives us great hope on significantly decreasing the energy consumption during parallel matrix multiplication.

Chapter 4

Matching-Based Assignment Strategies for Improving Data Locality of Map Tasks in MapReduce

4.1 Introduction

In this chapter we consider a new problem in which there is an upstream replication of the input files. Therefore, replication is here an input of the problem and the goal will be to use this redundancy in order to improve data locality. In the current section we first introduce the problem and its motivations. In Section 4.2 we present some related works, in particular for the improvement of locality in Map-Reduce and the search for matching in bipartite graphs. In Section 4.3 we relate a classical dynamic scheduler with the Balls-In-Bins modelling, providing probabilistic results for the scheduler. In Section 4.4 we propose matching-based algorithms to solve the problem for the two proposed metrics in polynomial time. Finally, in Section 4.5, we present some simulation results to compare our original strategy with pre-existing solutions.

4.1.1 MapReduce and HDFS

MapReduce is a well-known framework for distributing data-processing computations onto parallel clusters that has been introduced by Google and has been popularized by implementations like Hadoop (White [2012]). In MapReduce, a large computation is broken into small tasks that run concurrently on multiple machines. MapReduce is a very successful example of a dynamic scheduler, as one of its crucial feature is its inherent capability of handling hardware failures and processing capability heterogeneity, thus hiding this complexity to the programmer, by relying on on-demand assignments and the online detection

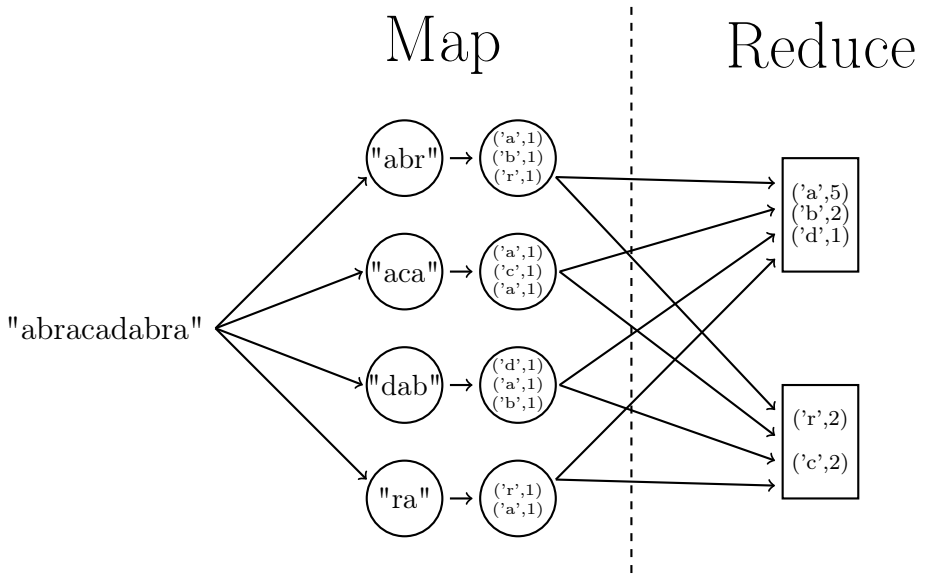


Figure 4.1: Illustration of MapReduce principle.

of nodes that perform poorly.

In a classical MapReduce application, the original dataset is first split into data chunks and distributed onto the computing nodes. Then, computation is decomposed into two phases: a Map phase followed by a Reduce phase, each of them being composed of several tasks. Let us consider a textbook example: letter count, computing the number of occurrences of each letter in a word. The Map phase of MapReduce splits the word in small pieces and processors compute the number of occurrences of each letter in their fraction of word. Then, during the Reduce phase, MapReduce partially aggregates these partial results, see Figure 4.1.

However, MapReduce is also widely used to distribute bag-of-tasks applications, which are composed of Map tasks only. Such applications represent 77% of the MapReduce jobs studied in Kavulya *et al.* [2010]. For these applications, data locality is the main source of communications. There have been relatively few theoretical studies of data locality in MapReduce and its impact on communications, see Guo *et al.* [2012].

In MapReduce, minimizing the amount of communications performed at runtime is a crucial issue. The initial distribution of the chunks onto the platform is performed by a distributed filesystem such as HDFS, Borthakur [2008]. By default, HDFS replicates randomly data chunks several times onto the nodes (usually 3 times). This replication has two main advantages. First, it improves the reliability of the process, limiting the risk of losing input data. Second, replication tends to minimize the number of communications at runtime. Indeed, by default, each node is associated to a given number of Map and Reduce slots (usually two of each kind). Whenever a Map slot becomes

available, the default scheduler first determines which job should be scheduled, given the job priority and history. Then, it checks whether the job has a local unprocessed data chunk on the processor. If yes, such a local task is assigned, and otherwise, a non-local task is assigned and the associated data chunk is sent from a distant node. Therefore, intuitively, having more replicas provides more opportunities for a given chunk of being processed locally.

It has been shown that depending on the size of the job, Ibrahim *et al.* [2012], the fraction of non-local tasks can be around 12-17%, and their processing takes between 1.2 to 2 times longer due to communications of remote data chunks. The quality of the data locality of a scheduling policy, given a replication mechanism, is therefore a crucial issue.

In this chapter we focus on the Map phase of MapReduce, the distribution of independent and presumed-homogeneous tasks between processors.

4.1.2 Metric: Communication vs Makespan

The problem we address is by nature a bi-criteria optimisation problem, the first one being the makespan (total completion time) of the Map phase and the second one being the number of non-local tasks (*i.e.* tasks that are not processed on one of the processors that holds the corresponding data chunk). In the case of homogeneous tasks, it is easy to achieve optimal makespan (never leave a processor idle by assigning non local tasks, as done in MapReduce) and conversely it is easy to perform only local tasks (keep a processor idle if it does not own unprocessed chunks anymore).

Both of these metrics will be considered, always under the assumption that the other one is optimized. More precisely,

- if we consider the makespan metric, we assume that only local tasks are performed,
- if we consider the communication metric, we assume that if a processor is idle it will begin an unprocessed task.

Throughout this chapter we use the following modelling. We represent the initial placement of data chunks by a bipartite graph $G = (T, P, E)$ with a set T of n task nodes and a set P of m processor nodes. An edge $e \in E$ between task node t_j and processor node p_i indicates the presence of a chunk of task t_j on processor p_i. Let σ be a function from T to P (the tasks allocation). With the assumption of homogeneity, our two metrics are equivalent to the following two problems.

Problem 4.1 (MAKESPAN-MAPREDUCE)**.** Given a bipartite graph $G = (T, P, E)$ find a function $\sigma : T \to P$ such that

- For every $t_j \in T$, $\{\sigma(t_j), t_j\} \in E$.

- $\max_{p_i \in P} |\{t_j,\ \sigma(t_j) = p_i\}|$ is minimized.

Problem 4.2 (COMMUNICATION-MAPREDUCE)**.** Given a bipartite graph $G = (T, P, E)$ find a function $\sigma : T \to P$ such that:

- $\max_{p_i \in P} |\{t_j,\ \sigma(t_j) = p_i\}| \leq \left\lceil \frac{|T|}{|P|} \right\rceil$.
- $|\{t_j,\ \{\sigma(t_j), t_j\} \notin E\}|$ is minimized.

In MAKESPAN-MAPREDUCE the first condition ensures that all tasks are local and $\max_{p_i \in P} |\{t_j,\ \sigma(t_j) = p_i\}|$ represents the makespan of the allocation (as all tasks have the same processing time, the most loaded processor will be the last to end and thus will determine the makespan). In COMMUNICATION-MAPREDUCE the first condition ensures the optimality of the makespan (if all processors perform no more than $\left\lceil \frac{|T|}{|P|} \right\rceil$ tasks, the load balancing is perfect) and $|\{t_i,\ \{\sigma(t_j), t_j\} \notin E\}|$ is the number of non-local tasks.

To help to construct the schedule σ, we propose to define a new object for this graph-based modelling.

Definition 4.1 (Assignment)**.** Let $G = (P, T, E)$ be a bipartite graph. An *assignment* of G is a subset A of E such that:

$$\forall t_j \in T,\ \exists \text{ a unique } p_i \in P \text{ such that } \{p_i, t_j\} \in A.$$

An assignment is thus a subset of the edges such that each task node has a unique edge in this subset. Therefore every assignment is a valid schedule. In particular this schedule respects the locality forced in MAKESPAN-MAPREDUCE. We now add metrics to evaluate the quality of an assignment.

Definition 4.2 (Degree in an assignment, maximum degree and total load imbalance)**.** Let A be an assignment of $G = (P, T, E)$.

- The *degree in* A of a vertex p_i of P, denoted $d_A(p_i)$, is the degree of p_i in $G' = (P, T, A)$, the sub-graph of G induced by A.
- the *maximum degree* $d(A)$ of A is defined as $D(A) = \max\limits_{p_i \in P} d_A(p_i)$.
- The *total load imbalance* $\mathrm{Imb}(A)$ of A is given by

$$\mathrm{Imb}(A) = \sum_{\substack{p_i \in P \\ d_A(p_i) > \left\lceil \frac{|T|}{|P|} \right\rceil}} \left(d_A(p_i) - \left\lceil \frac{|T|}{|P|} \right\rceil \right).$$

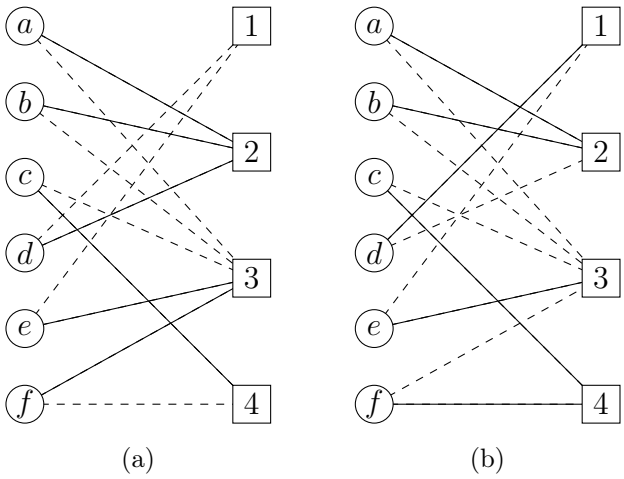


Figure 4.2: Two examples of assignments for the same input graph.

These notions are illustrated on Figure 4.2, where tasks are on the left hand side and processors on the right. Solid edges represent an assignment and dashed ones are the initial edges that are not used in the assignment. Each task can be uniquely associated to one solid edge, but processors can be assigned to more than one task. On Figure 4.2(a), the maximum degree is 3 (reached on 2) and the total load imbalance is 1. On Figure 4.2(b), the maximum degree is 2 (reached on 2 and 4) and the total load imbalance is 0.

Every schedule that is a solution of MAKESPAN-MAPREDUCE is a valid assignment (and every assignment is a schedule that is a solution of MAKESPAN-MAPREDUCE) and if $D(A)$ is minimized, then $\max_{p_i \in P} |\{t_j,\ A(t_j) = p_i\}|$ is minimized. Therefore, to solve MAKESPAN-MAPREDUCE we can focus on assignments with minimal maximal degree only.

For COMMUNICATION-MAPREDUCE we proceed as follows.

- If A is an assignment, then for every processor p_i such that $d_A(p_i) > \left\lceil \frac{|T|}{|P|} \right\rceil$ we unassign $d_A(p_j) - \left\lceil \frac{|T|}{|P|} \right\rceil$ tasks and attribute them to processors $p_{i'}$s such that $d_A(p_{i'}) < \left\lceil \frac{|T|}{|P|} \right\rceil$. At the end we have a schedule σ such that $\max_{p_i \in P} |\{t_j,\ \sigma(t_j) = p_i\}| \leq \left\lceil \frac{|T|}{|P|} \right\rceil$ and $|\{t_j,\ \{\sigma(t_j), t_i\} \notin E\}| \leq Imb(A)$ as non-local tasks may only be tasks whose assignment has been modified by the previous transformation.

- If σ is a schedule such that $\max_{p_i \in P} |\{t_j,\ \sigma(t_j) = p_i\}| \leq \left\lceil \frac{|T|}{|P|} \right\rceil$, then for every non-local task we attribute this task to an arbitrary processor such that there is an edge between them. With this transformation we have a valid assignment A and $Imb(A) \leq |\{t_j,\ \{\sigma(t_j), t_j\} \notin E\}|$. Indeed the set $\{p_i,\ d_A(p_i) > \left\lceil \frac{|T|}{|P|} \right\rceil\}$ is empty before the transformation (as

$\{t_j,\ \sigma(t_j) = p_i\}| \leq \left\lceil \frac{|T|}{|P|} \right\rceil$) and thus during the transformation $Imb(A)$ only increases by at most 1 at each new attribution of the t_j.

So from every solution of COMMUNICATION-MAPREDUCE with C communications we can produce an assignment A such that $\text{Imb}(A) \leq C$ and from every assignment A we can produce a solution of COMMUNICATION-MAPREDUCE with less than $Imb(A)$ communications. Thus, to solve COMMUNICATION-MAPREDUCE, we focus on assignment with minimum total load imbalance.

In the following, in particular in Section 4.4, we mainly use assignments as solutions of COMMUNICATION-MAPREDUCE and MAKESPAN-MAPREDUCE.

To finish this introduction, two remarks on the models:

- In COMMUNICATION-MAPREDUCE we consider that all non-local tasks as equivalent. In practice, processors are grouped into racks and among non-local tasks the communication cost is cheaper when data comes from a processor from the same rack. Some studies, Yekkehkhany [2017] or Isard *et al.* [2009], propose such refinements but it is beyond the scope of this chapter.

- We are not considering failures. Some heuristics (Ibrahim *et al.* [2012] for example) propose, if there is an idle processor, to launch a speculative task that is a duplicate of an unfinished task whose processing time is abnormally long. As our algorithms are based on a static approach this possibility is hard to model and we choose not to consider it in this chapter.

4.2 Related Work

4.2.1 Locality in Map-Reduce

A number of papers have studied the data locality in MapReduce. Note that most studies that aim at minimizing the communications focus on the *Shuffle* phase of a MapReduce application: in this phase, the scheduler transfers the output data of the Map tasks to create the input data of the Reduce tasks. Minimizing the communication of this data-intensive phase has been the target of many studies, such as with coflow scheduling, Chowdhury and Stoica [2012], Chowdhury and Stoica [2015] or Qiu *et al.* [2015]. Some papers have also proposed to place Reduce tasks close to their input to reduce the communications of the shuffle phase, Hammoud and Sakr [2011] or Tan *et al.* [2013].

However, as outlined in the previous section, we concentrate in this paper on the Map phase. Several studies have already tried to minimize the data exchange in this phase.

Zaharia *et al.* [2010] first proposed the *Delay* scheduler to improve data locality for several job competing on a cluster. In their strategy, if a given job

has a free slot for a new task on a processor buts owns no local chunk, instead of running a non-local task, leading to data movement (as in the classical scheduler), this job waits for a small amount of time, allowing other jobs to run local tasks instead (if they have some). The authors show that this improves data locality while preserving fairness among jobs.

Ibrahim *et al.* [2012] also outline that, apart from the shuffling phase, another source of excessive network traffic is the high number of non-local map tasks. They propose *Maestro*, an assignment scheme that intends to maximize the number of local tasks. To this end, *Maestro* estimates which processors are expected to be assigned the smallest number of local tasks, given the distribution of the replicas. These nodes are then selected first to assign local tasks. They experimentally show that this reduces the number of non-local map tasks.

Xie and Lu [2012] propose a simple assignment strategy based on the degree of each processor to overcome the problem of non-local map tasks. The idea is to give priority to processors with the smallest non-zero number of unprocessed replicas, so that they could be assigned a local task. They propose a "peeling" algorithm that first serves the processor with a single unprocessed replica (and thus assigns to them their unique local task), and then switches to a classical random assignment for other processors. Using queuing theory and assuming that processing times are given by geometric distribution, they prove that their strategy offers close to optimal assignment for small to medium load. In a similar context, Wang *et al.* [2013] propose a new queuing algorithm to simultaneously maximize throughput and minimize delay in heavily loaded conditions.

Guo *et al.* [2012] consider the locality of map tasks. They propose an algorithm based on the Linear Sum Assignment Problem to compute an assignment with minimal communications. Unfortunately, in the case where there are more tasks than processors, their formulation is obviously wrong: they add fictitious processors to get back to the case with equal number of tasks and processors, solve the problem, and then remove the fictitious processors without taking care of the task reassignment.

Isard *et al.* [2009] propose a flow-based scheduler: Quincy. More precisely they consider the case of concurrent jobs and want to ensure that if a job is submitted to the platform its computation time would be smaller than Jt second where t is its computation time with exclusive access to the platform and J the number of jobs running on the platform. To respect this deadline, they propose a sophisticated model with many parameters (size of the input files, bandwidth, data transfer, possible pre-emption of tasks,...) and transform this problem into a flow problem (see Ford and Fulkerson [2015]). Recently this algorithm's computation time has been significantly improved by Gog *et al.* [2016] in order to have sub-second scheduling time at each submission of a new job.

4.2.2 Matchings in Bipartite Graphs

A first research direction (more precision in Section 4.4) deals with the existence of matchings, in particular perfect matchings, *i.e.* matchings whose size is the number of vertices. For instance, the work of Erdös and Rényi [1964] on random bipartite graphs proves that there exists a perfect matching of a bipartite graph of $2n$ vertices with asymptotic probability $e^{-2e^{-c}}$ as soon as the number of edges is $n \ln n + cn + o(n)$. Walkup [1980], instead of an assumption on the number of edges, uses a condition on the minimum degree of the vertices. In this model, asymptotically, a regular bipartite graph (*i.e.* whose both sets of vertices are of equal size) has a perfect matching if its minimum degree is at least 2.

A second research direction deals with the efficient computation of matchings. Many algorithms rely on augmenting paths, see Ford and Fulkerson [2015]. An augmenting path is a path that induces an improvement of the current existing flow (or matching) by permuting some edges of the path with some of the actual solution. For bipartite graphs, very efficient algorithms exist to find an optimal matching, such as the Hopcroft-Karp Algorithm, Hopcroft and Karp [1973], with a complexity of $O(m\sqrt{n})$, where n denotes the number of vertices and m the number of edges, or the one proposed by Goel *et al.* [2013] for regular bipartite graphs, whose expected complexity is $O(n \log n)$. There also exist approximation algorithms with better computation time that no longer guarantee the computation of a perfect matching, Langguth *et al.* [2010] or Dufossé *et al.* [2015].

4.3 Greedy Approach

In this section, we consider MAKESPAN-MAPREDUCE only, *i.e.* non-local tasks are forbidden. We are interested in analysing the Hadoop standard scheduler, a greedy algorithm that picks a random local task as soon as a processor is idle. We prove that this scheduler can be modelled by a Balls-In-Bins process and thus its expected makespan is $\frac{n}{m} + O(\log \log m)$ where n is the number of task, and m the number of processors.

4.3.1 Balls-into-Bins

In the context of MAKESPAN-MAPREDUCE, processors are only allowed to compute the chunks they own locally. This might generate some load imbalance, since some of the processors may stop their computations early.

Such a process is closely related to Balls-In-Bins problems, Raab and Steger [1998]. More specifically, we prove in Section 4.3.2 that it is possible to simulate the greedy algorithm for assigning Map tasks with a variant of a Balls-In-Bins game. In this randomized process, n balls are placed randomly

into m bins and the expected load of the most loaded bin is considered. In a process where data chunks are not replicated, chunks correspond to balls, processing resources correspond to bins, and if tasks have to be processed locally, then the maximum load of a bin is equal to the makespan achieved by greedy assignment. The case of weighted balls, that corresponds to tasks whose lengths are not of unitary length, has been considered by Berenbrink *et al.* [2008]. It is shown by Berenbrink *et al.* [2008] that when assigning a large number of small balls with total weight W, one ends up with a smaller expected maximum load than the assignment of a smaller number of uniform balls with the same total weight. In the case of identical tasks, Raab and Steger [1998] provide value on the expected maximum load, depending on the ratio $\frac{m}{n}$. For example, in the case $m = n$, the expected maximum load is $\frac{\log n}{\log \log n}(1 + o(1))$.

Balls-into-bins techniques have been extended to multiple choice algorithms, where r random candidate bins are pre-selected for each ball, and then the ball is assigned to the candidate bin whose load is minimum. It is well known that having more than one choice strongly improves load balancing. We refer the interested reader to Mitzenmacher [2001] and Richa *et al.* [2001] for surveys that illustrate the power of two choices. Typically, combining previous results with those of Berenbrink *et al.* [2000], it can be proved that whatever the expected number of balls per bin, the expected maximum load is of order $n/m + O(\log \log m)$ even in the case $r = 2$, what represents a very strong improvement over the single choice case. Peres *et al.* [2010] study cases with a non-integer r. In this case, with a given parameter β, for each ball, with probability β the assignation is made after choosing between two bins or, with probability $(1 - \beta)$, the assignation is made like for a regular Balls-In-Bins process. In this case, for $0 < \beta < 1$, the expected maximum load is $\frac{n}{m} + O(\frac{\log m}{\beta})$. Thus, the exact $\beta = 1$ (*i.e.* $r = 2$) is needed to reach the $O(\log \log m)$ regular Balls-In-Bins gap. The combination of multiple choice games with weighted balls has also been considered by Peres *et al.* [2010]. In this case, each ball comes with its weight w_i and is assigned, in the r-choices case, to the bin of minimal weight, where the weight of a bin is the sum of the weights of the balls assigned to it. Both the results for $(1 + \beta)$ and for $r = 2$ have been extended.

4.3.2 Reduction to Balls-into-Bins

We consider here a simple dynamic scheduler to solve MAKESPAN-MAPREDUCE, called Greedy and inspired by the MapReduce scheduler and detailed in Algorithm 4.1. If there exists local not yet processed tasks, one such local task is chosen at random, performed locally and marked as processed on all processors. If no such unprocessed local task exists, then the processor stops its execution. Note that it reduces to the straightforward version of the MapRe-

duce scheduler when communications are forbidden.

Let us consider a more general context for the analysis of this algorithm, where tasks have heterogeneous processing times (w_i is the duration of task t_i) and processors have slightly different initial availability times (ϵ_j is the availability time of processor j). However, the GREEDY scheduler has no knowledge of the task durations before their execution. We assume, as in Berenbrink *et al.* [2008] or Peres *et al.* [2010], that this heterogeneity in task durations and processor availability times allows us to consider that no ties have to be broken. Note that in GREEDY, the initial chunk distribution is given by n random sets of r choices: $\mathcal{C}_1, \ldots, \mathcal{C}_n$. Together with the initial processor availability times and the task durations, these random choices entirely define the scheduler behavior.

Algorithm 4.1 : GREEDY($\mathcal{C}_1, \ldots, \mathcal{C}_n, \epsilon_1, \ldots, \epsilon_m$)

Input : $\mathcal{C}_1, \ldots, \mathcal{C}_n$: sets of r random choices
Input : $\epsilon_1, \ldots, \epsilon_m$: initial processor availability times
Initial chunk distribution:
for $i = 0 \ldots n$ **do**
  Place a copy of chunk i onto processors $\{p_{i^{(1)}}, \ldots, p_{i^{(r)}}\}$ where $\mathcal{C}_i = \{i^{(1)}, \ldots, i^{(r)}\}$
Task assignment:
while *there exists an unprocessed task* **do**
  Whenever a processor p_k completes a task, assign to this processor the local task t_i with smallest index, if any

We now prove that in presence of replication, the expected makespan of the GREEDY scheduler is closely related to the balls-into-bins problem with r multiple choices. Algorithm 4.2 shows the process of distributing n balls of sizes $w_1, \ldots, w_n$ into m bins whose initial loads are given by $\epsilon_1, \ldots, \epsilon_m$. This distribution is done as follows: for each ball, r bins are selected at random (using the random choices $\mathcal{C}_i$) and the ball is placed in the least loaded of these r bins. The following theorem shows the relation between the simple dynamic scheduler and the BALLS-IN-BINS process.

Algorithm 4.2 : BALLS-IN-BINS($\mathcal{C}_1, \ldots, \mathcal{C}_n, \epsilon_1, \ldots, \epsilon_m$)

for $i = 0 \ldots n$ **do**
  Place ball b_i of weight w_i into the least loaded bin among bins $\{B_{i^{(1)}}, \ldots, B_{i^{(r)}}\}$ where $\mathcal{C}_i = \{i^{(1)}, \ldots, i^{(r)}\}$;

Theorem 4.1. *Let us denote by* MAXLOAD *the maximal load of a bin using*

Balls-In-Bins *and by* $C_{\max}$ *the makespan achieved using* Greedy. *Then,*

$$\textsc{MaxLoad}(\mathcal{C}_1, \ldots, \mathcal{C}_n, \epsilon_1, \ldots, \epsilon_m) = C_{\max}(\mathcal{C}_1, \ldots, \mathcal{C}_n, \epsilon_1, \ldots, \epsilon_m).$$

Proof. In order to prove above result, let us prove by induction on i the following Lemma.

Lemma 4.2. *Let $j_b(i)$ denote the index of the bin where ball b_i is placed and let $j_p(i)$ denote the index of the processor where task t_i is processed, then $j_b(i) = j_p(i)$.*

Proof. Let us consider ball b_1 and task t_1. b_1 is placed in the bin such that ϵ_k is minimal, where $k \in C_1$. Conversely, t_1 is replicated onto all the processors p_k, where $k \in C_1$. Since each processor executes its tasks following their index, t_1 is processed on the first processor owning t_1 that looks for a task, *i.e.* the processor such that ϵ_k is minimal. This achieves the proof in the case $n = 1$.

Let us assume that the lemma holds true for all indexes $1, \ldots, i-1$, and let us consider the set of bins B_k and the set of processors p_k such that $k \in C_i$. By construction, at this instant, processors $p_k,\ k \in C_i$ have only processed tasks whose index is smaller than i. Let us denote by $S_i = \{t_{i_1}, \ldots, t_{i_{n_i}}\}$ this set of tasks, whose indexes i_k's are smaller than i. These tasks have been processed on the processors whose indexes are the same as those of the bins on which balls $\{b_{i_1}, \ldots, b_{i_{n_i}}\}$ have been placed, by induction hypothesis. Therefore, for each $p_k,\ k \in C_i$, the time at which p_k ends processing the tasks assigned to it and whose index is smaller than i is exactly the weight of the balls with index smaller than i placed in B_k. Therefore, the processor p_k that first tries to compute t_i is the one such that ϵ_k plus the weight of the balls with index smaller than i placed in B_k is minimal, so that $j_b(i) = j_p(i)$, what achieves the proof of the lemma. □

Therefore, the makespan achieved by Greedy on the inputs $(\mathcal{C}_1, \ldots, \mathcal{C}_n, \epsilon_1, \ldots, \epsilon_m)$ is equal to the load of most loaded bin in Balls-In-Bins on the same input, which achieves the proof of the theorem. □

Thanks to this result, we can apply known bounds on the maximum load for Balls-In-Bins processes derived in the literature, as related in the previous section. In particular, going back to the case of tasks with identical processing times, the expected makespan when $r \geq 2$ is known to be of order $n/m + O(\log \log m)$ (with high probability).

4.4 Matching-Based Approach

In this section we are interested in optimal solutions for MAKESPAN-MAPREDUCE and COMMUNICATION-MAPREDUCE. First, in Section 4.4.1, we link MAKESPAN-MAPREDUCE to an already studied problem: GRAPH-ORIENTIABILITY

and present the pre-existing results, notably the existence, with high probability, of an assignment with makespan smaller or equal than $\left\lceil \frac{|T|}{|P|} \right\rceil + 1$. Finally, in Section 4.4.2 and 4.4.3, we focus on COMMUNICATION-MAPREDUCE and present two algorithms to solve it in polynomial time. Note that these two algorithms are also proven optimal for MAKESPAN-MAPREDUCE, therefore, on every bipartite graph, there exists an assignment that is optimal for both MAKESPAN-MAPREDUCE and COMMUNICATION-MAPREDUCE.

4.4.1 Results for Makepan Metric

Let us first introduce the GRAPH-ORIENTIABILITY problem. We distinguish (undirected) edges, denoted $\{u, v\}$ from (directed) arcs, denoted (u, v).

Problem 4.3 (GRAPH-ORIENTIABILITY)**.** Given an undirected graph $G = (V, E)$, find a directed version $G' = (V, E')$, with $\{u, v\} \in E \Leftrightarrow (u, v) \in E'$ or $(v, u) \in E'$, such that $\max_{v \in V} |\{u \in V,\ (u, v) \in E'\}|$ is minimized.

GRAPH-ORIENTIABILITY can be used to solve MAKESPAN-MAPREDUCE in the case where the degree of each task is 2. Let $G = (P, T, E)$ be such an instance of MAKESPAN-MAPREDUCE. Let $G' = (P, E')$ be an undirected graph such that $\{p_i, p_{i'}\} \in E'$ if and only if $\{p_i, t_j\}$ and $\{p_{i'}, t_j\}$ are in E (we allow multiple edges). Note that $|E'| = |T|$. If we consider a directed version $G'' = (P, E'')$ of G' then A, defined such that $\{p_i, t_j\} \in A$ if and only if $(p_{i'}, p_i) \in E''$, it is a valid assignation and its maximal degree is the maximal number of incident arc to a vector in G'' (the orientation of an edge represents a choice between two vertices), see Figure 4.3. Note that GRAPH-ORIENTIABILITY can be extended to hypergraphs to model versions of MAKESPAN-MAPREDUCE with arbitrary r (where r is the number of replicas of each task).

One of the main contributions to this problem comes from Sanders *et al.* [2003]. In this paper they provide polynomial algorithms for GRAPH-ORIENTIABILITY and its hypergraph version and a probabilistic evaluation on the expected value of the optimal value of an input of GRAPH-ORIENTIABILITY. More precisely, with high probability, the optimal solution is at least *near-perfect* (*i.e.* $\max_{v \in V} |\{u \in V,\ (u, v) \in E'\}| \leq 1 + \left\lceil \frac{|E'|}{|P|} \right\rceil$).

Theorem 4.3 (Sanders *et al.* [2003])**.** *Let $G = (V, E)$ be an undirected graph. With high probability there exists a directed version of G such that the vertex with the most in-incident arcs has an in-degree inferior or equal to $1 + \left\lceil \frac{|E|}{|V|} \right\rceil$.*

Therefore, under the assumption that each task has at least two replicas (otherwise we are in a case close to the Balls-In-Bins modelling without the power of r choices) Theorem 4.3 implies directly that there exists a near-perfect assignment for the MAKESPAN-MAPREDUCE problem, as stated in the following corollary.

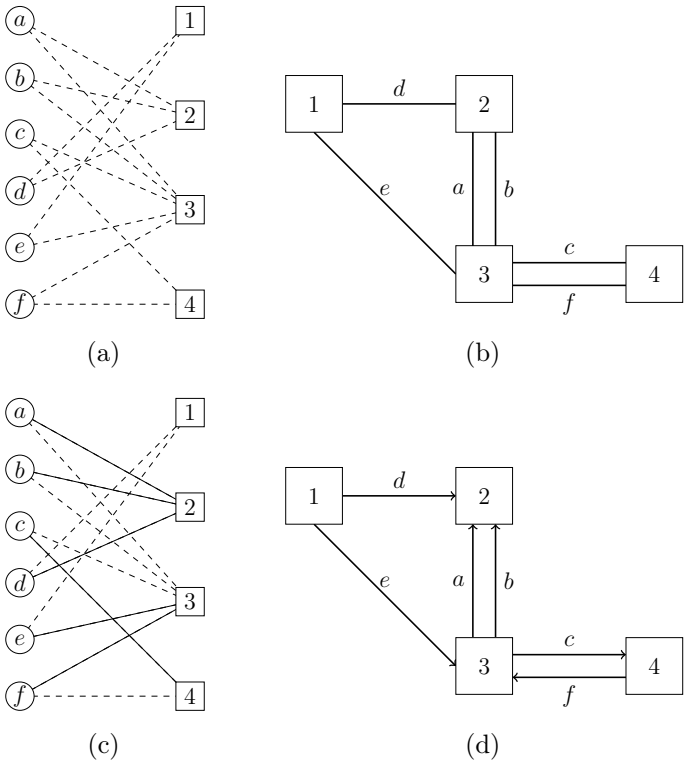


Figure 4.3: Figure 4.3(a) represents an input of MAKESPAN-MAPREDUCE which equivalent version of an input of GRAPH-ORIENTIABILITY is on Figure 4.3(b). The assignment on Figure 4.3(c) is equivalent to the orientation of Figure 4.3(d).

Corollary 4.4. *Let $G = (P, T, E)$ be a bipartite graph such that the degree of every element of T is at least 2. Then, with high probability there exists an assignation with maximal degree smaller or equal to* $1 + \left\lceil \frac{|T|}{|P|} \right\rceil$.

Proof. By randomly removing excess edges we place ourselves in the case where for all $t_j \in T$, the degree of t_j is 2. Therefore we can transform G into an undirected graph $G' = (P, E')$ as described above (see Figure 4.3). Thanks to Theorem 4.3, we know that there is a directed version of G' such that the vertex with the most in-incident arcs has an in-degree inferior or equal to $1 + \left\lceil \frac{|E'|}{|P|} \right\rceil = 1 + \left\lceil \frac{|T|}{|P|} \right\rceil$. This oriented version of G' has an induced assignment for G with the transformation also described above and in Figure 4.3, and this assignment has the same maximal degree as the maximal in-degree in G'', what achieves proof of Corollary 4.4. □

Furthermore, Theorem 4.3 has been extended by Czumaj *et al.* [2003] by proving that for a small number of tasks the optimal solution of GRAPH-ORIENTIABILITY is exactly $1 + \left\lceil \frac{|T|}{|P|} \right\rceil$ and for large numbers it is $\left\lceil \frac{|T|}{|P|} \right\rceil$.

Theorem 4.5 (Czumaj *et al.* [2003])**.** *Let $G = (V, E)$ be an undirected graph. There exists two positive constants λ, c such that:*

- *If $|E| \geq c|V|\log|V|$ then with high probability there exists a directed version of G such that the vertex with the most in-incident arcs has an in-degree equal to $\left\lceil \frac{|E|}{|V|} \right\rceil$.*
- *If $|E| \leq \lambda|V|\log|V|$ then with high probability there exists a directed version of G such that the vertex with the most in-incident arcs has an in-degree equal to $1 + \left\lceil \frac{|E|}{|V|} \right\rceil$.*

With this theorem, the probabilistic study of the expected optimal value of the optimal maximal degree of an assignation is closed. Very efficient algorithms have been proposed to polynomially solve GRAPH-ORIENTIABILITY, Sanders [2004] or Cain *et al.* [2007], with very good complexity (near-linear in the case of graphs, *i.e.* $r = 2$). To finish this section, we propose a sketch of a direct proof of Corollary 4.4, without the transformation into an instance of GRAPH-ORIENTIABILITY, because some of the intermediate results will prove useful for the following sections.

Sketch of Proof of Corollary 4.4

As a first step, we establish the relationship between assignments and matchings. We recall that a matching of a bipartite graph $G = (P, T, E)$ is a subset M of E such that each node of $P \cup T$ is incident to at most one edge in M. There are some notable differences between an assignment and a matching. First, in an assignment, two edges may be incident to the same processor node. Second, a task node may not be covered by an edge of a matching, while it is necessary connected to a processor node in an assignment.

For a given integer l, we build an auxiliary bipartite graph G^l from the bipartite graph representation of our problem, by replicating l times the set of processors, see Figure 4.4. We show in the following lemma that the existence of an assignment of maximum degree l in G is directly related to the existence of a matching of size $|T|$ in G^l.

Definition 4.3 (G^l)**.** Let $G = (P, T, E)$ be a bipartite graph and l be an integer. We define $G^l = (P^l, T, E^l)$ with:

- $P^l = \bigcup_{p_i \in P} \{p_i^1, \ldots, p_i^l\}$.
- $(p_i^k, t_j) \in E^l$ if and only if $(p_i, t_j) \in E$.

Lemma 4.6. *Let $G = (P, T, E)$ be a bipartite graph with $|T| = n$ and let l be an integer. There exists an assignment of maximum degree l if and only if there exists a matching of size n in G^l.*

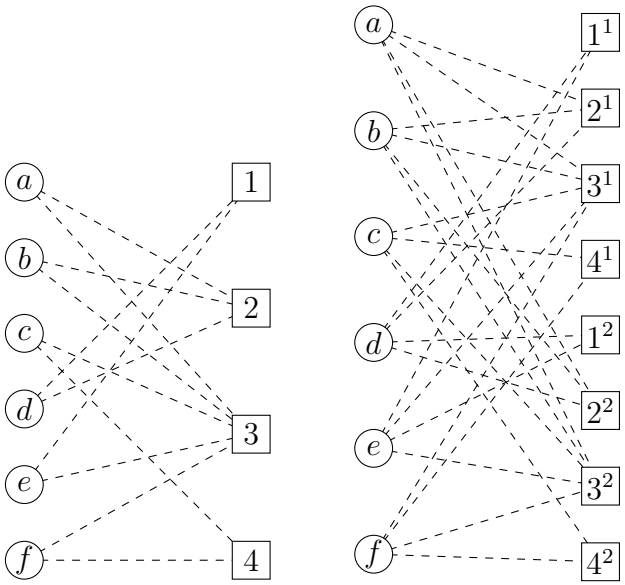


Figure 4.4: An example of replication of a bipartite graph. On the left G, on the right G^2.

Proof. Let us first suppose that there exists an assignment A of maximum degree l of G. For $p_i \in P$ let $v_A(p_i)$ denotes its neighbourhood in the sub-graph induced by A, *i.e.* $v_A(p_i) = \{t_j \in T,\ (p_i, t_j) \in A\}$ (and $|v_A(p_i)| = d_A(p_i)$). Let M be a subset of E^l with $M = \bigcup_{p_i \in P} M_{p_i}$ with

$$M_{p_i} = \big\{(p_i^1, t_1), \ldots, (p_i^{d_A(p_i)}, t_{d_A(p_i)}), \\ \{t_1, \ldots, t_{d_A(p_i)}\} = v_A(p_i)\big\}.$$

Since $\forall p_i \in P$, $d_A(p_i) \leq l$, then $\forall p_i$, $\{p_i^1, \ldots, p_i^{d_A(p_i)}\}$ is a valid subset of P^l. Let (p_i^k, t_j) and $(p_{i'}^{k'}, t_{j'})$ denote any two edges of M.

- If $i \neq i'$, then $p_i^k \neq p_{i'}^{k'}$. In addition, by definition of an assignment, $v_A(p_i) \cap v_A(p_{i'}) = \emptyset$ (otherwise there is a contradiction with $\exists! p_i \in P, (p_i, t_j) \in A$) and then $t_j \neq t_{j'}$.
- If $i = i'$, then $t_j = t_{j'}$ if and only if $k = k'$.

Therefore, M is a valid matching. Moreover, since $v_A(p_i) \cap v_A(p_{i'}) = \emptyset$, then $M_{p_i} \cap M_{p_{i'}} = \emptyset$. Therefore, $|M| = \sum |M_{p_i}| = \sum d_A(p_i) = n$ by definition of an assignment. Thus, there exists a matching of cardinal n in G^l.

Let us now assume that there exists a matching M of G^l with $|M| = n$. Let us build a subset A of E such that

$$(p_i, t_j) \in A \Leftrightarrow \exists k,\ (p_i^k, t_j) \in M.$$

Let (p_i, t_j) and $(p_{i'}, t_{j'})$ be any two edges of A. There exists (k, k') such that $(p_i^k, t_j), (p_{i'}^{k'}, t_{j'}) \in M$. By definition of a matching, $t_j = t_{j'}$ if and only $p_i^k = p_{i'}^{k'}$. Hence $t_j = t_{j'}$ if and only if $p_i = p_{i'}$. Moreover, $|A| = |M| = n$ and therefore, $\forall t_j$, there exists p_i such that $(p_i, t_j) \in A$. Thus A is an assignment of G and $D(A) \leq l$ as $d_A(p_i) \leq l$ for all $p_i \in P$. □

Note that Lemma 4.6 states that finding an optimal assignment for a bipartite graph G is equivalent to finding a maximal matching (in G^l). So this implies a possible method, summarized in Algorithm 4.3, to solve MAKESPAN-MAPREDUCE: (i) building G^l, (ii) finding a maximal matching in G^l, and (iii) turning it into an assignment for G.

Algorithm 4.3 : NaiveAssignment(G)

Input : A bipartite graph $G = (P, T, E)$
Output : An assignment A of G of minimum maximal degree
$l = \left\lceil \frac{|T|}{|P|} \right\rceil$;
$A = \emptyset$;
while *A is not an assignment* **do**
    Find a maximal matching M of G^l ;
    if $|M| = |T|$ **then**
        $A \leftarrow$ the conversion of M into an assignment of size l ;
    $l \leftarrow l + 1$;
return A

Thanks to Lemma 4.6 we can now use a classical result on matching in bipartite graphs, namely Hall's Theorem (Theorem 4.7).

Theorem 4.7 (Hall [1935])**.** *Let $G = (P, T, E)$ be a bipartite graph. There exists a perfect matching (of cardinal $min(|P|, |T|)$) of G if and only if for all subset T' of T, its neighbourhood P' verifies $|P'| \geq |T'|$.*

By using Lemma 4.6 and Theorem 4.7 we prove a sufficient and necessary condition to have a assignment with maximal degree l.

Lemma 4.8. *Let $G = (P, T, E)$ be a bipartite graph. There is an assignment of G of maximum degree l if and only if for all subset T' of T, its neighbourhood P' satisfies $l|P'| \geq |T'|$.*

Proof. Let T' be a subset of T and let $v_G(T')$ be its neighbourhood in G. By construction of $G^l = (P^l, T, E^l)$, $|v_{G^l}(T')| = l|v_G(T')|$. In addition, thanks to Lemma 4.6, there exists an assignment of G of maximum degree l if and only if there is a perfect matching of G^l, what is equivalent (Theorem 4.7) to $\forall T' \subseteq T$, $|T'| \leq |v_{G^l}(T')| = l|v_G(T')|$. □

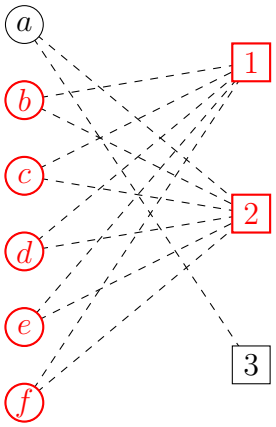


Figure 4.5: An illustration of Lemma 4.8. The set of red tasks is composed of 5 tasks whose neighbourhood is a set of only two processors. Therefore, as $5 > 2 \times 2$ there is no assignment with maximal degree 2.

In the following, for a given l, we want to evaluate the probability of existence of a configuration which forbids assignment of maximal degree l. Thanks to Lemma 4.8 we know that if there is a subset T' of T such that its neighbourhood P' verifies $|T'| > l|P'|$ then there is no such assignment, as illustrated in Figure 4.5.

Let t_j be a task of T and let P' be a subset of P of cardinal q. The probability that the entire neighbourhood of t_j is included in P' is $\left(\frac{q}{m}\right)^r$ (since $m = |P|$) by using a draw with replacement (a more pessimistic hypothesis than the real draw without replacement).

Let $X_{P'}$ be the random variable that represents the number of tasks whose neighbourhood is included in P'. $X_{P'}$ follows a binomial law of parameter n (the number of tasks) and $\left(\frac{q}{m}\right)^r$ (the probability that a task has its neighbourhood included in P'), where q denotes the cardinal of P'. Therefore,

$$Pr(X_{P'} = k) = \binom{n}{k} \left(\left(\frac{q}{m}\right)^r\right)^k \left(1 - \left(\frac{q}{m}\right)^r\right)^{n-k}.$$

Let $Y_{q,k}$ denote the number of subsets of P of size q such that exactly k elements of T have their neighbourhood included in this subset, *i.e.*

$$Y_{q,k} = \sum_{\substack{P' \subseteq P \\ |P'|=q}} \mathbb{1}_{X_{P'}=k}$$

where $\mathbb{1}_{X_{P'}=k}$ is the random variable equal to 1 when $X_{P'} = k$ and to 0 otherwise. Therefore, $E(\mathbb{1}_{X_{P'}=k}) = Pr(X_{P'} = k)$ and, by linearity of expectations,

$$
\begin{aligned}
E(Y_{q,k}) &= E\left(\sum_{\substack{P' \subseteq P \\ |P'|=q}} \mathbb{1}_{X_{P'}=k}\right) \\
&= \sum_{\substack{P' \subseteq P \\ |P'|=q}} E(\mathbb{1}_{X_{P'}=k}) \\
&= \sum_{\substack{P' \subseteq P \\ |P'|=q}} Pr(X_{P'}=k) \\
&= \sum_{\substack{P' \subseteq P \\ |P'|=q}} \binom{n}{k} \left(\left(\frac{q}{m}\right)^r\right)^k \left(1-\left(\frac{q}{m}\right)^r\right)^{n-k} \\
&= \binom{m}{q}\binom{n}{k} \left(\frac{q}{m}\right)^{rk} \left(1-\left(\frac{q}{m}\right)^r\right)^{n-k}.
\end{aligned}
$$

Thanks to Lemma 4.8 and as it is explained before, there is no assignment of maximal degree l if and only if there exists T' and its neighbourhood P' such that $l|P'| < |T'|$ and thus if and only if there exists (q,i), with $q \in [0,p], i \in \mathbb{N}^*$, such that $Y_{q,lq+i} \geq 1$.

Note that the condition $(Y_{q,lq+l+i} \geq 1)$ is included in the condition $(Y_{q+1,l(q+1)+i} \geq 1)$. Indeed, if there is a set of q processors that contains the entire neighbourhood of $lq+l+i$ tasks, there also exists a set of $q+1$ processors that contains this neighbourhood (obtained by adding any processor not already in the subset). Thus, $Y_{q,lq+l+i} \geq 1$ implies $Y_{q+1,l(q+1)+i} \geq 1$. On the contrary, $Y_{q+1,l(q+1)+i} = 0$ implies $Y_{q,lq+l+i} = 0$. Therefore, we can focus on the events $(Y_{q,lq+i} \geq 1)$ with $i \in [1,l]$ only. Let us define the random variable $Z_l = \sum_{q=0}^{m}\sum_{i=1}^{l} Y_{q,lq+i}$. If $Z_l = 0$, then $Y_{q,lq+i} = 0$ for all (q,i) and there exists an assignment of maximum degree l. Otherwise, if $Z_l \geq 1$, there exists a (q,i) such that $Y_{q,lq+i} \geq 1$ and then there is no assignment of maximum degree l.

Using the Markov inequality, we obtain

$$
\begin{aligned}
Pr(Z_l \geq 1) &\leq \frac{E(Z_l)}{1} \\
&\leq \sum_{q=0}^{m}\sum_{i=1}^{l} E(Y_{q,lq+i}) \\
&\leq \sum_{q=0}^{m}\sum_{i=1}^{l} \binom{m}{q}\binom{n}{lq+i} \left(\frac{q}{m}\right)^{r(lq+i)} \left(1-\left(\frac{q}{m}\right)^r\right)^{n-lq-i}.
\end{aligned}
$$

To achieve the proof it can be proven that if $r = 2$ and m divides n (every input of MAKESPAN-MAPREDUCE that fulfils the condition of Corollary 4.4 can be placed under these two conditions by removing edges) then $Pr(Z_l \geq 1) = O(\frac{1}{n})$. However, the proof is mainly computational and technical, so we just provide the main steps to this result, avoiding proofs of technical claims. The interested reader can go to Beaumont *et al.* [2017] for the complete proof.

First Claim 4.1 states that instead of dealing with a double sum $\sum_{q=0}^{m}\sum_{i=1}^{l} E(Y_{q,lq+i})$, we can focus on a simple sum, $\sum_{q=1}^{m} E(Y_{q,lq+1})$.

Claim 4.1 (Beaumont *et al.* [2017], p26)**.**

$$Pr(Z_l \geq 1) < 2\sum_{q=1}^{m} E(Y_{q,lq+1}).$$

In order to bound $Pr(Z_m \geq 1)$, we want then to bound $E(Y_{q,lq+1})$. Let us recall the following well known upper and lower bounds on the factorial (Bollobás [2013]). For any $n \in \mathbb{N}^*$,

$$\sqrt{2\pi n}\left(\frac{n}{e}\right)^n < n! < \sqrt{2\pi n}\left(\frac{n}{e}\right)^n e^{\frac{1}{12n}}$$

and thus, for any $n > m > 0$,

$$\begin{aligned}\binom{n}{m} &= \frac{n!}{m!(n-m)!} \\ &< \frac{\sqrt{2\pi n}\left(\frac{n}{e}\right)^n e^{\frac{1}{12n}}}{\sqrt{2\pi m}\left(\frac{m}{e}\right)^m \sqrt{2\pi(m-p)}\left(\frac{n-m}{e}\right)^{m-p}} \\ &< \frac{1}{\sqrt{2\pi}} \frac{n^{n+\frac{1}{2}}}{m^{m+\frac{1}{2}}(n-m)^{n-m+\frac{1}{2}}} e^{\frac{1}{12n}}.\end{aligned}$$

Due to the domain of validity of above inequality, the rest of the proof will be split into two parts. The case $lq + 1 = n$ solved by Claim 4.2 and the case $mq + 1 < n$ is considered in Claim 4.3.

Claim 4.2 (Beaumont *et al.* [2017], p27)**.** If $lq + 1 = n$, then $E(Y_{q,lq+1}) = O\left(\frac{1}{n}\right)$.

Claim 4.3 (Beaumont *et al.* [2017], p29)**.** If $lq + 1 < n$,

$$E(Y_{q,lq+1}) < \frac{e^{\frac{1}{12m}} e^{\frac{1}{12n}}}{2\pi\sqrt{nm}} e^{n\left(f_{0,n}(x,\alpha)+\frac{1}{n}f_{1,n}(x,\alpha)\right)}$$

where $x = \frac{q}{p}$, $\alpha = \frac{m}{n}$,

$$f_{1,n}(x,\alpha) \leq K,$$

with $K \in \mathbb{N}$, and

$$f_{0,n}(x,\alpha) \leq f(x) + C_n,$$

where $C_n = O\left(\frac{1}{n}\right)$ and $f(x) = x\ln(x) - \frac{(1-x)^2}{x}\ln(1-x) + \ln(x + (1-x)^{\frac{1}{x}}(1+x))$.

Claim 4.3 proves that

$$E(Y_{q,lq+1}) < \frac{e^{\frac{1}{12m}} e^{\frac{1}{12n}}}{2\pi\sqrt{nm}} e^{nf(x)+nC_n+K}$$

for $x \in I_n = \left[\frac{1}{\alpha n}, \frac{n-1}{(\alpha+1)n} - \frac{1}{\alpha n}\right]$.

Thus,

$$\begin{aligned}
\sum_{q=1}^{\frac{n-1}{l}} E(Y_{q,lq+1}) &\leq \sum_{x=q/m \in I_n} \frac{e^{\frac{1}{12m}} e^{\frac{1}{12n}}}{2\pi\sqrt{nm}} e^{nf(x)+nC_n+K} \\
&\leq \frac{e^{\frac{1}{12m}} e^{\frac{1}{12n}}}{2\pi n\sqrt{\alpha}} e^{nC_n+K} \sum_{x=q/m \in I_n} e^{nf(x)} \\
&\leq O\left(\frac{1}{n}\right) \sum_{x=q/m \in I_n} e^{nf(x)}.
\end{aligned}$$

Therefore the end of this proof relies on the study of the function f. This function is strictly negative on $]0,1[$ (Claim 4.4) and is equivalent to $x\ln(x)$ when x goes to 0 (Claim 4.5). Note that we suppose $\alpha = \frac{m}{n} < 1$. The case $m = n$ has to be treated separately (Claim 4.6).

Claim 4.4 (Beaumont *et al.* [2017], p37)**.** $\forall x \in]0,1[$, $f(x) < 0$.

Claim 4.5 (Beaumont *et al.* [2017], p41)**.** For all $\epsilon \in]0, 1-\alpha[$, there exists x_0 such that $x \leq x_0$ implies $f(x) \leq (1-\epsilon)x\ln(x)$.

Claim 4.6 (Beaumont *et al.* [2017], p23)**.** Let us assume that (i) $r = 2$, (ii) $n = m$ and (iii) $l = 2$. Then $Pr(Z_l \geq 1) = O\left(\frac{1}{n}\right)$.

Without loss of generality, we can assume that the x_0 from Claim 4.5 is smaller than e^{-1}. As $x \mapsto x\ln(x)$ is decreasing on $]0, e^{-1}]$, $\forall x \in \left[\frac{1}{\alpha n}, x_0\right]$, $f(x) \leq -(1-\epsilon)\frac{1}{\alpha n}\ln(\alpha n)$. In addition, f is continuous on $\left[x_0, \frac{1}{1+\alpha}\right]$ and thus, there exists $x_1 \in \left[x_0, \frac{1}{1+\alpha}\right]$ such that $f(x) \leq f(x_1)$. Thanks to Claim 4.4, $f(x_1) < 0$ and $\exists K' < 0$ such that $\forall x \in \left[x_0, \frac{1}{1+\alpha}\right]$, $f(x) \leq K'$. Thus,

$$\begin{aligned}
\sum_{x=q/m\in I_n} e^{nf(x)} &= \sum_{x=q/m\in\left[\frac{1}{\alpha n},x_0\right]} e^{nf(x)} + \sum_{x=q/m\in\left[x_0,\frac{1}{1+\alpha}\right]} e^{nf(x)} \\
&= \sum_{x=q/m\in\left[\frac{1}{\alpha n},x_0\right]} e^{-\frac{1-\epsilon}{\alpha}ln(\alpha n)} + \sum_{x=q/m\in\left[x_0,\frac{1}{1+\alpha}\right]} e^{nK'} \\
&= \frac{1}{(\alpha n)^{\frac{1-\epsilon}{\alpha}}}\left(\sum_{x=q/m\in\left[\frac{1}{\alpha n},x_0\right]} 1\right) + e^{nK'}\left(\sum_{x=q/m\in\left[x_0,\frac{1}{1+\alpha}\right]} 1\right) \\
&= \frac{1}{(\alpha n)^{\frac{1-\epsilon}{\alpha}}} \times n + e^{nK'} \times n \\
&= O(n^{\frac{\alpha-1+\epsilon}{\alpha}}) + O(ne^{nK'}) \\
&= O(1)
\end{aligned}$$

because $\alpha - 1 + \epsilon \leq 0$, what ends the case $lq + 1 < n$.

Thus we can conclude

$$\begin{aligned}
Pr(Z_l \geq 1) &\leq 2\sum_{q=1}^{\frac{n-1}{l}} E\left(Y_{q,lq+1}\right) \\
&\leq 2\sum_{q=1}^{\frac{n-1}{l}-1} E\left(Y_{q,lq+1}\right) + 2E\left(Y_{\frac{n-1}{l},n}\right) \\
&\leq O\left(\frac{1}{n}\right) \sum_{x=q/m\in I_n} e^{nf(x)} + O\left(\frac{1}{n}\right) \\
&\leq O\left(\frac{1}{n}\right)
\end{aligned}$$

what achieves the proof.

□

4.4.2 A First Communication-Optimal Algorithm

In this section we focus on COMMUNICATION-MAPREDUCE problem. In what follows, we propose a polynomial algorithm to polynomially solve COMMUNICATION-MAPREDUCE, following classical ideas from the literature on the matching and flows problem (see Ford and Fulkerson [2015] for instance).

Alternating Paths

Let us first adapt the notion of alternating path.

Definition 4.4. Let $G = (P, T, E)$ be a bipartite graph and let A be a subset of E.

- A *path* of G is a sequence of vertices $(x_1, \ldots, x_k)$ such that $\forall i \in [1, k-1]$, $(x_i, x_{i+1}) \in E$. Note that in a path of a bipartite graph the vertices switch between T and P.
- An *alternating path* of G according to A is a path $(x_1, \ldots, x_k)$ of G such that:
  - If $x_i \in T$, then $(x_i, x_{i+1}) \in A$.
  - If $x_i \in P$, then $(x_i, x_{i+1}) \notin A$.

An example of an alternating path is described in Figure 4.6. On the left hand side, solid edges represent an assignment, and dashed ones the unused edges. On the right hand side, the proposed path (to improve the clarity of the scheme, edges that are not in the path have been removed) is an alternating one according to the previous assignment.

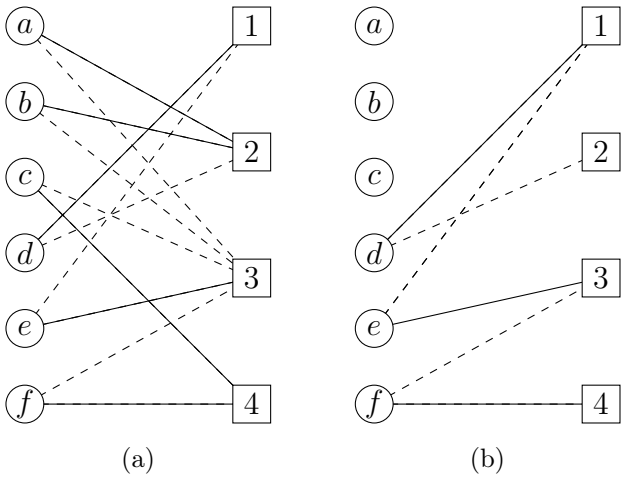


Figure 4.6: Example of alternating path.

Alternating paths can be used to improve an existing assignment. Indeed, Lemma 4.9 states that if the starting and the ending vertices of an alternating path are in P, then it is possible to build an assignment that improves the degree of the last vertex and increases by one the degree of the first one.

Lemma 4.9. *Let $G = (P, T, E)$ be a bipartite graph, let A be an assignment of G and let $x = (p_{i_1}, t_{j_1}, \ldots, p_{i_k})$ be an alternating path of G according to A. Let*

$\otimes$ be the xor operation ($A \otimes B = (A \cup B) \setminus (A \cap B)$) and let x be assimilated to its edges, then,

- *$A \otimes x$ is an assignment.*
- $d_{A\otimes x}(p_{i_1}) = d_A(p_{i_1}) + 1$
- $d_{A\otimes x}(p_{i_k}) = d_A(p_{i_k}) - 1$
- $\forall p_i \in P \setminus \{p_{i_1}, p_{i_k}\},\ d_{A\otimes x}(p_i) = d_A(p_i)$.

Proof. Let $G = (P, T, E)$ be a bipartite graph, let A be an assignment of G and let $x = (p_{i_1}, t_{j_1}, \ldots, t_{j_{k-1}}, p_{i_k})$ be an alternating path of G according to A. Let $t_j \in T$:

- If $t_j = t_{j_l} \in \{t_{j_1}, \ldots, t_{j_{k-1}}\}$, then, by definition of an alternating path, $(t_{j_l}, p_{i_{l+1}})$ is in A (and, by definition of an assignment, it is the only edge from t_j in A) and (p_{i_l}, t_{j_l}) is not. Therefore (p_{i_l}, t_{j_l}) is in $A \otimes x$ (and it is the only edge from t_j in $A \otimes x$) and $(t_{j_l}, p_{i_{l+1}})$ is not.
- Otherwise, the unique edge in A from t_j is not in x (and x has no edges from t_j) and therefore is also present in $A \otimes x$.

Therefore, $\forall t_j \in T$, there is an unique edge from t_j in $A \otimes x$ that is thus an assignment.

Furthermore, let p_i be in P, then

- if $p_i = p_{i_1}$, then (p_{i_1}, t_{j_1}) is added to its neighbourhood in $A \otimes x$,
- if $p_i = p_{i_k}$, then $(t_{j_{k-1}}, p_{i_k})$ is removed from its neighbourhood in $A \otimes x$,
- if $p_i = p_{i_l} \in \{p_{i_2}, \ldots, p_{i_{k-1}}\}$, then (p_{i_l}, t_{j_l}) is added to its neighbourhood and the edge $(t_{j_{l-1}}, p_{i_l})$ is removed from its neighbourhood in $A \otimes x$,
- otherwise there is no modification of its neighbourhood from A to $A \otimes x$.

Therefore,

- $d_{A\otimes x}(p_{i_1}) = d_A(p_{i_1}) + 1$
- $d_{A\otimes x}(p_{i_k}) = d_A(p_{i_k}) - 1$
- $\forall p_i \in P \setminus \{p_{i_1}, p_{i_k}\},\ d_{A\otimes x}(p_i) = d_A(p_i)$.

□

From now on, we focus on finding an alternating path that improves the whole assignment. For this purpose we define improving path.

Definition 4.5. Let $G = (P, T, E)$ be a bipartite graph and A be an assignment of G. An *improving path* according to A is an alternating path $x = (p_{i_1}, t_{j_1}, \ldots, p_{i_k})$ such that $d_A(p_{i_1}) + 1 < d_A(p_{i_k})$.

Lemma 4.9 implies that an improving path can be used to build an assignment that can even decrease the maximum degree if p_{i_k} is the only vertex such that $d_A(p_{i_k}) = D(A)$. Similarly, an improving path can also be used to build an assignment that decreases the total load imbalance if $d_A(p_{i,k}) > \left\lceil \frac{|T|}{|P|} \right\rceil$ and $d_A(p_{i,1}) < \left\lceil \frac{|T|}{|P|} \right\rceil$.

Optimality

In fact, a stronger property holds true: if there is no improving path, then the assignment has a minimum total load imbalance. This result is formalized in Theorem 4.10.

Theorem 4.10. *Let $G = (P, T, E)$ be a bipartite graph, let A be an assignment of G. If there is no improving path according to A then A has a minimum total load imbalance.*

Proof. First let us introduce a first lemma to state the different cases that may happen if we use an alternating path to change an assignment, with regards on load imbalance.

Lemma 4.11. *Let $G = (P, T, E)$ be a bipartite graph and A be an assignment of G. Let $x = (p_d, \ldots, p_f)$ be an alternating path according to A. Then,*

- *if $d_A(p_d) < \left\lceil \frac{|T|}{|P|} \right\rceil < d_A(p_f)$ then $Imb(A \otimes x) = Imb(A) - 1$,*
- *if $d_A(p_d) > \left\lceil \frac{|T|}{|P|} \right\rceil > d_A(p_f)$ then $Imb(A \otimes x) = Imb(A) + 1$,*
- *otherwise $Imb(A \otimes x) = Imb(A)$.*

Proof. By direct application of Lemma 4.6. □

Note that an alternating path $x = (p_d, \ldots, p_f)$ such that $d_A(p_d) < \left\lceil \frac{|T|}{|P|} \right\rceil < d_A(p_f)$ is an improving path.

We will rely on the three following lemmas to prove Theorem 4.10.

Lemma 4.12. *(Berge Berge [1957]) Let $G = (P, T, E)$ be a bipartite graph and M be a matching of G. M is maximal if and only if there no alternating path $x = (p_{i_1}, t_{j_1}, \ldots, t_{j_{k-1}})$ such that there is no edge from p_{i_1} and from $t_{j_{k-1}}$ in M.*

Lemma 4.13. *Let A and A' be two assignments of a same bipartite graph $G = (P, T, E)$. If there exists p_i such that $d_A(p_i) > d_{A'}(p_i)$, then there exists an alternating path $x = (p_d, \ldots, p_f)$ according to A such that $d_A(p_d) < d_{A'}(p_d)$ and p_f is the vertex verifying $d_A(p_f) > d_{A'}(p_f)$ with the largest degree in A.*

Proof. To prove the above lemma, we need a more general definition of a replicated graph. Instead of replicating each vertex of P l times, we replicate p_i l_{p_i} times (the number of replications can be different from a vertex to another). In this case, Lemma 4.6 can be adapted: let $G = (P, T, E)$ be a bipartite graph with $|T| = n$, $|P| = m$ and $(l_i)_{1 \leq i \leq m}$ be a set of integers. Then, there exists an assignment A with, for all $p_i \in P$, $d_A(p_i) \leq l_i$ if and only if there exists a matching of size n in the replicated graph of G where each p_i is replicated l_i times.

Let now A and A' be two assignments of a bipartite graph G such that $\exists p_i$ that fulfils $d_A(p_i) > d_{A'}(p_i)$. Let p_f be the vertex satisfying $d_A(p_f) > d_{A'}(p_f)$ with the largest degree in A. Let us consider the graph G' that is a replicated graph of G where p_f is replicated $d_A(p_f) - 1$ times and where $p_i \in P \setminus \{p_f\}$ is replicated $max(d_A(p_i), d_{A'}(p_i))$. Let $t \in T$ be such that $(p_f, t) \in A$. Thanks to the modified version of Lemma 4.6, we can define a matching M' of G' of size $|T|$ and a matching M of size $|T| - 1$ associated to the partial assignment $A \setminus (p_f, t)$.

M is not a matching of maximum size since M' is larger. Therefore, according to Lemma 4.12, there exists an alternating path $x = (p_{i_1}, t_{j_1}, \ldots, t_{j_{k-1}})$ such that there is no edge from p_{i_1} and from $t_{j_{k-1}}$ in M. The only possible free vertex from T is t, thus this alternating path must fulfil $x = (p_d^{k_d}, \ldots, t)$. In addition, to have a free replicate in G' according to M, p_d must fulfil $d_A(p_d) < \max(d_A(p_d), d_{A'}(p_d))$ and therefore $d_A(p_d) < d_{A'}(p_d)$.

We then easily check that the path $(p_d, \ldots, t, p_f)$ is an alternating path and, as $d_A(p_d) < d_{A'}(p_d)$, thus we have the claimed result. □

Lemma 4.14. *Let $G = (P, T, E)$ be a bipartite graph and let A, A' be two assignments of G. Therefore there are finite sequences of assignments $(A_k)_{k \leq l}$ and paths $(x_k)_{k \leq l-1}$ (l is the length of the sequence) such that*

- $A_0 = A$
- $\forall p_i \in P,\ d_{A_l}(p_i) = d_{A'}(p_i)$
- $\forall k \leq l$, *x_k is alternating according to A_k and $A_{k+1} = A_k \otimes x_k$*
- *If $x_k = (p_{d_k}, \ldots, p_{f_k})$, then $d_{A_k}(p_{d_k}) < d_{A'}(p_{d_k})$ and $d_{A_k}(p_{f_k}) > d_{A'}(p_{f_k})$ and p_{f_k} is the p_i with larger degree in A_k such that $d_{A_k}(p_i) > d_{A'}(p_i)$.*
- $\forall p_i \in P$, *the sequence $(|d_{A_k}(p_i) - d_{A'}(p_i)|)_{k \leq l}$ is decreasing.*

Proof. Let us suppose that these sequences have been built until rank k. If $\forall p_i \in P,\ d_{A_k}(p_i) = d_{A'}(p_i)$, then sequences are built. Otherwise, we recall that $\sum_{p_i \in P} d_{A_k}(p_i) = |T| = \sum_{p_i \in P} d_{A'}(p_i)$. Thus if $\exists p_i$ such that $d_{A_k}(p_i) \neq d_{A'}(p_i)$, then $d_{A_k}(p_i) > d_{A'}(p_i)$ or $\exists p'_i$ is such that $d_{A_k}(p'_i) > d_{A'}(p'_i)$. Therefore, according to Lemma 4.13, there exists an alternating path $x_k = (p_{d_k}, \ldots, p_{d_f})$ such that $d_{A_k}(p_{d_k}) < d_{A'}(p_{d_k})$ and p_{f_k} is the vertex with the larger degree in A_k verifying $d_{A_k}(p_{f_k}) > d_{A'}(p_{f_k})$. Let us define $A_{k+1} = A_k \otimes x_k$.

Let us prove that sequences are finite. Let us consider the sequence $(u_k)_{k\in\mathbb{N}}$ defined by $u_k = \sum_{p_i \in P} |d_{A_k}(p_i) - d_{A'}(p_i)|$.

$$
\begin{aligned}
u_{k+1} &= \sum_{p_i \in P} |d_{A_{k+1}}(p_i) - d_{A'}(p_i)| \\
&= \sum_{p_i \in P \setminus \{p_{d_k}, p_{f_k}\}} |d_{A_{k+1}}(p_i) - d_{A'}(p_i)| + |d_{A_{k+1}}(p_{d_k}) - d_{A'}(p_{d_k})| + |d_{A_{k+1}}(p_{f_k}) - d_{A'}(p_{f_k})| \\
&= \sum_{p_i \in P \setminus \{p_{d_k}, p_{f_k}\}} |d_{A_k}(p_i) - d_{A'}(p_i)| + |d_{A_k}(p_{d_k}) + 1 - d_{A'}(p_{d_k})| + |d_{A_k}(p_{f_k}) - 1 - d_{A'}(p_{f_k})| \\
&= \sum_{p_i \in P \setminus \{p_{d_k}, p_{f_k}\}} |d_{A_k}(p_i) - d_{A'}(p_i)| + |d_{A_k}(p_{d_k}) - d_{A'}(p_{d_k})| - 1 + |d_{A_k}(p_{f_k}) - d_{A'}(p_{f_k})| - 1 \\
&= u_k - 2
\end{aligned}
$$

Therefore there exists an index l such that $u_l = 0$ and hence $\forall p_i \in P,\ d_{A_l}(p_i) = d_{A'}(p_i)$ and the sequences are finite. Note that similar calculations prove that $(|d_{A_k}(p_i) - d_{A'}(p_i)|)_{k \leq l}$ is decreasing for all p_i. □

Let us now finish the proof of Theorem 4.10. Let $G = (P, T, E)$ be a bipartite graph and let A be an assignment of G. Let us suppose that A has not an optimal total load imbalance. In this case, there exists an assignment A' such that $\mathrm{Imb}(A) > \mathrm{Imb}(A')$. Thanks to Lemma 4.14, we can build two finite sequences $(A_k)_{k \leq l}$ and $(x_k)_{k \leq l-1}$ such that

- $A_0 = A$,
- $A_l = A'$,
- $\forall p_i \in P,\ d_{A_l}(p_i) = d_{A'}(p_i)$,
- $\forall k < m$, x_k is alternating and $A_{k+1} = A_k \otimes x_k$.

In particular $\mathrm{Imb}(A_0) > \mathrm{Imb}(A_l)$. Therefore, there exists k_0 such that $\mathrm{Imb}(A_{k_0}) > \mathrm{Imb}(A_{k_0+1}) = \mathrm{Imb}(A_{k_0} \otimes x_{k_0})$. Let $x_{k_0} = (p_{d_{k_0}}, \ldots, p_{f_{k_0}})$, then, thanks to Lemma 4.11, $d_{A_{k_0}}(p_{d_{k_0}}) < \left\lceil \frac{|T|}{|P|} \right\rceil < d_{A_{k_0}}(p_{f_{k_0}})$. Moreover, by construction of the sequences, $d_{A_{k_0}}(p_{f_{k_0}}) > d_{A'}(p_{f_{k_0}})$.

Thus, we have proven that there exists k_0 such that $x = (p_d, \ldots, p_f)$ is an alternating path according to A_{k_0} and such that $d_{A_{k_0}}(p_d) < \left\lceil \frac{|T|}{|P|} \right\rceil < d_{A_{k_0}}(p_f)$ and $d_{A_{k_0}}(p_f) > d_{A'}(p_f)$. Let k_0 be the smallest k such that there exists such a path according to A_k (not necessarily x_k). In order to prove that $k_0 = 0$, let us assume by contradiction that $k_0 > 0$.

Let $x = (p_d, \ldots, p_f)$ be an alternating path according to A_{k_0} such that $d_A(p_d) < \left\lceil \frac{|T|}{|P|} \right\rceil < d_A(p_f)$ and $d_{A_{k_0}}(p_f) > d_{A'}(p_f)$.

Let us suppose that x and x_{k_0-1} are disjoint. In this case, x is a valid alternating path according to A_{k_0-1}, $d_{A_{k_0-1}}(p_d) = d_{A_{k_0}}(p_d) < \left\lceil \frac{|T|}{|P|} \right\rceil$ and $d_{A_{k_0-1}}(p_f) = d_{A_{k_0}}(p_f) > \left\lceil \frac{|T|}{|P|} \right\rceil$. Thus, we reach a contradiction with the definition of k_0.

Let us assume that x and x_{k_0-1} are not disjoint. In this case, let us define v_i and v_j such that

- v_i is the first vertex in x to be in x_{k_0-1},
- v_j is the last vertex in x to be in x_{k_0-1}.

If v_i is before v_j in x_{k_0-1}, then $(p_d, \ldots, v_i, \ldots, v_j, \ldots p_f)$ is a valid alternating path according to A_{k_0-1}. Furthermore, if $v_i \neq p_d$, p_d is not in x_{k_0-1} and $d_{A_{k_0-1}}(p_d) = d_{A_{k_0}}(p_d) < \left\lceil \frac{|T|}{|P|} \right\rceil$. Otherwise, $d_{A_{k_0-1}}(p_d) = d_{A_{k_0}}(p_d) - 1 < \left\lceil \frac{|T|}{|P|} \right\rceil$. Thus, in all cases $d_{A_{k_0-1}}(p_d) < \left\lceil \frac{|T|}{|P|} \right\rceil$ and similarly, we prove that $\left\lceil \frac{|T|}{|P|} \right\rceil < d_{A_{k_0-1}}(p_f)$ and reach a contradiction with the definition of k_0.

Therefore, v_j is before v_i in x_{k_0-1}. Let us consider the path $y = (p_d, \ldots, v_i, \ldots, p_{f_{k_0-1}})$, that is a valid alternating path according to A_{k_0-1}. As previously, $d_{A_{k_0-1}}(p_d) < \left\lceil \frac{|T|}{|P|} \right\rceil$. In addition, $d_{A'}(p_f) < d_{A_{k_0}}(p_f) \leq d_{A_{k_0-1}}(p_f)$ by assumption and because $(|d_{A_k}(p_i) - d_{A'}(p_i)|)_{k \leq l}$ is decreasing. Thus, by construction of the p_{f_k}'s, $d_{A_{k_0-1}}(p_f) \leq d_{A_{k_0-1}}(p_{f_{k_0-1}})$. Therefore, $\left\lceil \frac{|T|}{|P|} \right\rceil < d_{A_{k_0-1}}(p_{f_{k_0-1}})$ and y is an alternating path with all desired properties. Therefore, we reach a contradiction with the definition of k_0.

Hence, we prove by contradiction that $k_0 = 0$ and thus that there exists an alternating path $x = (p_d, \ldots, p_f)$ according to A such that $d_A(p_d) < \left\lceil \frac{|T|}{|P|} \right\rceil < d_A(p_f)$. Therefore, there exists an improving path according to A, what achieves the proof of Theorem 4.10 by contraposition.

□

Therefore, by searching for an improving path until there is none left, COMMUNICATION-MAPREDUCE can be optimally solved. Theorem 4.10 then give us a first algorithm: FindAssignment (see Algorithm 4.4).

In addition, Theorem 4.10 can be adapted to MAKESPAN-MAPREDUCE. Indeed, if there is no improving path according to an assignment A, then A has

Algorithm 4.4 : FINDASSIGNMENT(G)

Input : A bipartite graph $G = (P, T, E)$
Output : An assignment A of G of minimum maximal degree and minimum total load imbalance

$A = \emptyset$;
foreach $t_j \in T$ **do**
  Choose a random p_i such that $(p_i, t_j) \in E$ and add this edge to A;
while *there exists an improving path according to* A **do**
  Compute an improving path x according to A;
  $A \leftarrow A \otimes x$;
return A

a minimum maximum degree, what is stated in Theorem 4.15. Therefore FINDASSIGNMENT solves MAKESPAN-MAPREDUCE and COMMUNICATION-MAPREDUCE simultaneously and can be used in both cases.

Theorem 4.15. *Let $G = (P, T, E)$ be a bipartite graph, let A be an assignment of G. If there is no improving path according to A, then A has a minimum maximum degree.*

Proof. This proof is very similar to the proof of Theorem 4.10 and relies on the same lemmas, in particular Lemma 4.14.

Let $G = (P, T, E)$ be a bipartite graph and let A be an assignment of G. Let us suppose that A has not a minimum maximum degree. Therefore, there exists A' such that $D(A') < D(A)$.

Thank to Lemma 4.14, we know that there are finite sequences of assignments $(A_k)_{k \leq l}$ and paths $(x_k)_{k<l}$ such that

- $A_0 = A$,
- $A_l = A'$,
- $\forall p_i \in P,\ d_{A_l}(p_i) = d_{A'}(p_i)$,
- $\forall k < m$, x_k is alternating and $A_{k+1} = A_k \otimes x_k$.

In particular $D(A_0) > D(A_l)$. Therefore, there exists k_0 such that $D(A_{k_0}) > D(A_{k_0+1}) = D(A_{k_0} \otimes x_{k_0})$. Let $x_{k_0} = (p_{d_{k_0}}, \ldots, p_{f_{k_0}})$. Since x_{k_0} is alternating

- $d_{A_{k_0} \otimes x_{k_0}}(p_{d_{k_0}}) = d_{A_{k_0}}(p_{d_{k_0}}) + 1$,
- $d_{A_{k_0} \otimes x_{k_0}}(p_{f_{k_0}}) = d_{A_{k_0}}(p_{f_{k_0}}) - 1$,
- $\forall p_i \in P \setminus \{p_{d_{k_0}}, p_{f_{k_0}}\},\ d_{A_{k_0} \otimes x_{k_0}}(p_i) = d_{A_{k_0}}(p_i)$.

Yet, $D(A_{k_0}) > D(A_{k_0} \otimes x_{k_0})$ and, since $p_{f_{k_0}}$ is the only processor with decreasing degree, $D(A_{k_0}) = d_{A_{k_0}}(p_{f_{k_0}})$. Moreover,

$$\begin{aligned} &d_{A_{k_0}}(p_{d_{k_0}}) + 1 = d_{A_{k_0} \otimes x_{k_0}}(p_{d_{k_0}}), \\ &d_{A_{k_0}}(p_{d_{k_0}}) + 1 \leq D(A_{k_0} \otimes x_{k_0}), \\ &d_{A_{k_0}}(p_{d_{k_0}}) + 1 < D(A_{k_0}), \\ &d_{A_{k_0}}(p_{d_{k_0}}) + 1 < d_{A_{k_0}}(p_{f_{k_0}}). \end{aligned}$$

Therefore, x_{k_0} is improving, $D(A_{k_0}) = d_{A_{k_0}}(p_{f_{k_0}})$ and $d_{A_{k_0}}(p_{d_{k_0}}) < d_{A'}(p_{d_{k_0}})$.

We have proved if there exists an index k_0 such that $D(A_{k_0}) > D(A')$, there exists an improving path according to A_{k_0} with some additional properties. Let us define k_0 as the smallest k such that $D(A_k) > D(A')$ and there exists an improving path $x = (p_d, \ldots, p_f)$ according to A_k (and that improving path could differ from x_k) such that $d_{A_k}(p_f) = D(A_k)$ and $d_{A_{k_0}}(p_d) < d_{A'}(p_d)$. In order to prove that $k_0 = 0$ by contradiction, let us assume that $k_0 > 0$.

Let $x = (p_d, \ldots, p_f)$ be such an improving path in A_{k_0}, $D(A_{k_0}) > D(A')$.

First, let us note that $D(A_{k_0-1}) \geq D(A_{k_0}) > D(A')$. Indeed, for all $p_i \neq p_{d_{k_0-1}}$, $d_{A_{k_0-1}}(p_i) \geq d_{A_{k_0}}(p_i)$.

Second, let us assume that x and x_{k_0-1} are disjoint. In this case, $d_{A_{k_0-1}}(p_d) = d_{A_{k_0}}(p_d) < d_{A'}(p_d)$ and $d_{A_{k_0-1}}(p_f) = d_{A_{k_0}}(p_f) = D(A_{k_0}) \leq D(A_{k_0-1})$. If $D(A_{k_0}) = D(A_{k_0-1})$, then, x is a valid improving path and $d_{A_{k_0-1}}(p_f) = D(A_{k_0-1})$. Otherwise, $D(A_{k_0-1}) = d_{A_{k_0-1}}(p_{f_{k_0-1}})$, indeed $D(A_{k_0-1}) > D(A_{k_0}) > D(A')$, and thus, there exists at least one p_i such that $d_{A_{k_0-1}}(p_i) > d_{A'}(p_i)$ and by construction this is also the case for $p_{f_{k_0-1}}$), so that

$$d_{A_{k_0-1}}(p_{d_{k_0-1}}) + 1 = d_{A_{k_0}}(p_{d_{k_0-1}}) \leq D(A_{k_0}) < d_{A_{k_0-1}}(p_{f_{k_0-1}}).$$

Hence, x_{k_0-1} is an improving path with $D(A_{k_0-1}) = d_{A_{k_0-1}}(p_{f_{k_0-1}})$ and $d_{A_{k_0-1}}(p_{d_{k_0-1}}) < d_{A'}(p_{d_{k_0-1}})$ by construction. Thus x and x_{k_0-1} are not disjoint since otherwise, we would reach a contradiction with the definition of k_0.

Therefore, we know that x and x_{k_0-1} are not disjoint. Let v_i and v_j be two vertices of G such that

- v_i is the first vertex in x to be in x_{k_0-1},
- v_j is the last vertex in x to be in x_{k_0-1}.

Let us suppose that v_i is before v_j in x_{k_0-1}. Then, $(p_d, \ldots, v_i, \ldots, v_j, \ldots, p_f)$ is a valid alternating path in A_{k_0-1}. Because $(|d_{A_k}(p_i) - d_{A'}(p_i)|)_{k \leq l}$ is decreasing, $d_{A_{k_0-1}}(p_d) \leq d_{A_{k_0}}(p_d) < d_{A'}(p_d)$. Similarly $d_{A_{k_0-1}}(p_f) \geq d_{A_{k_0}}(p_f)$ and thus $d_{A_{k_0-1}}(p_d) + 1 < d_{A_{k_0-1}}(p_f)$ and $(p_d, \ldots, v_i, \ldots, v_j, \ldots, p_f)$ is improving. If $D(A_{k_0-1}) = D(A_{k_0})$, then we reach a contradiction with the definition of k_0. Otherwise we know that $D(A_{k_0-1}) = d_{A_{k_0-1}}(p_{f_{k_0-1}})$ and can prove that $(p_d, \ldots, v_i, \ldots, v_j, \ldots, p_{f_{k_0-1}})$ is an improving path with all the properties we want, thus another contradiction.

Finally, we prove that v_i is after v_j in x_{k_0-1}. Therefore, $y = (p_{d_{k_0}}, \ldots, v_j, \ldots, p_f)$ is an alternating path according to A_{k_0-1}. As previously, $d_{A_{k_0-1}}(p_{d_{k_0}}) < d_{A'}(p_{d_{k_0}}) \leq D(A')$. Thus $d_{A_{k_0-1}}(p_{d_{k_0}}) + 1 \leq D(A')$. Similarly, $D(A') < D(A_{k_0}) = d_{A_{k_0}}(p_f) \leq d_{A_{k_0-1}}(p_f)$. Thus, y is improving and if $D(A_{k_0-1}) = D(A_{k_0})$, then we reach a contradiction with the definition of k_0. Otherwise, we consider $y' = (p_d, \ldots, v_i, \ldots, p_{f_{k_0}})$ and reach a similar contradiction.

Hence, we prove that, in any case, $k_0 > 0$ implies the existence of an improving path according to A_{k_0-1} with all desired properties. Therefore, $k_0 = 0$ and there is an improving path according to $A_0 = A$, which achieves the proof of the theorem.

□

Correctness and Complexity

To prove the termination of FindAssignment, let us prove that there is no loops (an improving path creating another improving and so goes on) and thus the number of improving paths goes to 0. Let us consider $\sum d_A(p_i)^2$. If $x = (p_d, \ldots, p_f)$ is an improving path, then

$$\begin{aligned}\sum d_{A\otimes x}(p_i)^2 - \sum d_A(p_i)^2 &= (d_A(p_d)+1)^2 + (d_A(p_f)-1)^2 - d_A(p_d)^2 - d_A(p_f)^2 \\ &= 2(d_A(p_d) + 1 - d_A(p_f)) < 0\end{aligned}$$

Therefore, $\sum d_A(p_i)^2$ is decreasing during the execution of FindAssignment. Since this value is bounded (trivially by 0), there is an instant where there is no longer an improving path and FindAssignment terminates, returning an assignment with minimum total load imbalance and minimum maximum degree.

Similarly to Ford-Fulkerson Algorithm (Ford and Fulkerson [1956]), the search for an improving path can be done by breadth-first-search in $O(|E|)$. Furthermore $0 \leq \sum d_A(p_i)^2 \leq \left(\sum d_A(p_i)\right)^2 = |T|^2$ and $\sum d_A(p_i)^2$ strictly decreases at each step. Thus there is at most $|T|^2$ steps and each needs $|E|$ operations. Therefore the worst case complexity of FindAssignment is $O(|E||T|^2)$.

Theorem 4.16. FindAssignment *terminates in at most* $O(|E||T|^2)$ *operations.*

Note that in the application problem $|E| = r|T|$ where r is the number of replications of each chunk. Then the worst case complexity is $O(|T|^3)$.

4.4.3 A Faster Communication-Optimal Algorithm

However, in FindAssignment, the search of an improving path can be expensive and the problem of producing fast scheduling algorithms has been recently highlighted by Gog *et al.* [2016]. This is why we propose a faster algorithm, BestAssignment.

Presentation

Instead of going from any assignment and improving it step by step, like FindAssignment, BestAssignment goes from a processor after another, each time searching for the smallest alternating path from this processor to a non-assigned task (if any) and applying it to the current partial assignment, see Algorithm 4.5.

Algorithm 4.5 : BestAssignment (G)

Input : A bipartite graph $G = (P, T, E)$
Output : An assignment A of G of minimum maximal degree and minimum total load imbalance

```
A = ∅ ;
while A is not an assignment do
    foreach p_i ∈ P do
        if There is an alternating path according to A from p_i to an
        unassigned t_j then
            x = the smallest such path ;
            A ← A ⊗ x ;
        else
            Remove p_i from P ;
return A
```

Correctness and Complexity

As for FindAssignment the search for alternating path can be done in $O(|E|)$. Furthermore at each step, either a task is assigned or a processor is removed, thus the number of steps is $O(|T|+|P|)$. Thus BestAssignment has a worst case complexity of $O((|T|+|P|)|E|)$. Furthermore, we prove the optimality of BestAssignment by showing there is no improving path according to the returned assignment.

Theorem 4.17. BestAssignment *returns an Assignment with no improving path.*

Proof. Let A_∞ be the final set returned by BestAssignment. If we suppose that there is always an edge incident to each task (else there is no existing assignment anyway), trivially A_∞ is an assignment. It is constructed only using alternating path and there is always, for each task, a such path from a processor (in particular a simple edge from one of its neighbours). Let us now prove the optimality of this assignment by showing there is no improving path according to A_∞.

Let p_{i_1} be the first processor, if any, such that there is no alternating path from p_{i_1} to an unassigned task according to the current state of A that we denote A_1. Let us now define P_1 as the subset of P that contains all the processors that can be reached with an alternating path from p_{i_1} according to A_1. Let T_1 be the neighbourhood of P_1, *i.e.* $T_1 = \{t_j \in T,\ \exists p_i \in P_1,\ (p_i, t_j) \in E\}$.

We now want to show that for all $t_j \in T_1$ there exists p_i in P_1 such that $(p_i, t_j) \in A_1$. Let t_j be in T_1 and $p_i \in P_1$ be in its neighbourhood. By definition of P_1 there is an alternating path $(p_{i_1}, \ldots, p_i)$ according to A_1. If $(p_i, t_j) \in A_1$ we have our result, else $(p_{i_1}, \ldots, p_i, t_j)$ is a valid alternating path according to A_1. By definition of p_{i_1}, t_j is assigned and then there is $p_{i'}$ such that $(p_{i'}, t_j) \in A_1$ and thus $x(p_{i_1}, \ldots, p_i, t_j, p_{i'})$ is also a valid alternating path and $p_{i'} \in P_1$. Therefore we have for all $t_j \in T_1$ there exists p_i in P_1 such that $(p_i, t_j) \in A_1$.

Let now A_{1,T_1} (respectively A_{∞,T_1}) be $\{(p_i, t_j) \in A_1,\ t_j \in T_1\}$ (respectively $\{(p_i, t_j) \in A_\infty,\ t_j \in T_1\}$). We also denote similarly A'_{T_1} for any partial assignment A'. Then, for every partial assignment A' after A_1, we prove $A_{1,T_1} = A'_{T_1}$. Let us suppose otherwise and search for a contradiction. Let A' be the first partial assignment after A_1 such that $A_{1,T_1} \otimes A'_{T_1} \neq \emptyset$. Because all the edges of A_1 from T_1 goes to an element of P_1, we know that there is $(p_i, t_j) \in A_{1,T_1} \otimes A'_{T_1}$ such that $p_i \in P_1$. In addition, in the assignment just before A' (we denote it A''), there is an alternating path that contains (p_i, t_j). Let $x = (p_d, \ldots, p_i, t_j, \ldots, t_{j'})$ be this path ($A' = A'' \otimes x$). We know that $t_{j'}$ is unassigned in A'' and thus $t_{j'} \notin T_1$ and so $x = (p_d, \ldots, p_i, t_j, \ldots, p_f, t_{j'})$ with $p_f \notin P_1$. Hence there is $p_{i'}$ such that $p_{i'}$ is the first vertex in $P \setminus P_1$ after p_i in x. Without loss of generality we can assume that $p_{i'}$ is after t_j. By definition $A''_{T_1} = A_{1,T_1}$ and then $(p_{i_1}, \ldots, p_i, t_j, p_{i'})$ is a valid alternating path according to A_1 and A'' and therefore $p_{i'} \in P_1$ that is a contradiction. Then, for every partial assignation A' after A_1, $A_{1,T_1} = A'_{T_1}$. More precisely, after A_1, all processors in P_1 will not be in any alternating path (this is why they are removed from P in BestAssignment) and all tasks in T_1 will stay assigned to processors from P_1.

Similarly we also prove that $d_{A_1}(p_i) = d_{A'}(p_i)$ for all p_i in P_1 and, by construction $|d_{A_1}(p_i) - d_{A_1}(p_{i'})| \leq 1$ for all $p_i, p_{i'} \in P_1^2$ (at each step of the algorithm the degree of the departure vertex increase by one).

Now we define p_{i_k} as the first processor of $P \setminus \bigcup_{1 \leq l < k} P_l$ if any, such that there is no alternating path from p_{i_k} to an unassigned task. Let P_k the subset of $P \setminus \bigcup_{1 \leq l < k} P_k$ that contains all the processors that can be reached with an alternating path from p_{i_k} according to the current partial assignation, denoted A_k. We also define T_k its neighbourhood in $T \setminus \bigcup_{1 \leq l < k} T_k$. Finally we also denote $P_\infty = P \setminus \bigcup P_k$ and T_∞ its neighbourhood in $T \setminus \bigcup T_k$.

With the same reasoning than for P_1, T_1 we proved:

- For all $t_j \in T_k$ there is p_i in P_k such that $(p_i, t_j) \in A_k$.
- For every partial assignation A' after A_k, $A_{k,T_k} = A'_{T_k}$.
- For every partial assignation A' after A_k, $d_{A_k}(p_i) = d_{A'}(p_i)$ for all p_i in P_k and $|d_{A_k}(p_i) - d_{A_k}(p_{i'})| \leq 1$ for all $p_i, p_{i'} \in P_k^2$.

Note that $|d_{A_k}(p_i) - d_{A_k}(p_{i'})| \leq 1$ for all $p_i, p_{i'} \in P_k^2$ and for all $t_j \in T_k$ there is p_i in P_k such that $(p_i, t_j) \in A_k$ holds true for $k = \infty$. Note also that P_∞ can be equal to P in the case where an alternating path is found at each step.

We now want to prove that there is no improving path according to A_∞. First, as $|d_{A_k}(p_i) - d_{A_k}(p_{i'})| \leq 1$ for all $p_i, p_{i'} \in P_k^2$ and $d_{A_k}(p_i) = d_{A_\infty}(p_i)$ for all p_i in P_k. Thus there is no improving path inside the P_ks.

In addition we easily note that $|(\max\ d_{A_k}(p_i)) - (\min\ d_{A_{k'}}(p_{i'}))| \leq 1$ for $k < k'$. Therefore there is no improving path from a vertex of $P_{k'}$ to a vertex of P_k if $k < k'$. Hence the only possible improving path are from P_k to $P_{k'}$ with $k < k'$. However, in this case, there is t_j in T_k such there exists p_i in $P \setminus P_k$ such that $(p_i, t_j) \in A_\infty$ (a necessary condition to a such alternating path). Yet we prove for all $(p_i, t_j) \in A_{\infty,T_k} = A_{k,T_k}$, $p_i \in P_k$ and thus we have our proof of optimality of BestAssignment. □

With Theorem 4.17 and the previous remark, we prove that BestAssignment solves COMMUNICATION-MAPREDUCE and MAKESPAN-MAPREDUCE with a worst case complexity of $O((|T|+|P|)|E|)$ ($O(|T|^2+|T||P|)$ in the case $|E| = r|T|$).

In practice, we produce a C implementation, in particular for simulations of Section 4.5, and measure its computation time. The results are on Figure 4.7. Globally the average computation time is below $1s$ (as in Gog *et al.* [2016], but the difference with their model brings a need for further investigations to do a fair comparison), except for very high values of $|T|$, more than 250000 tasks, that is 3 times the number of tasks of the largest job in Kavulya *et al.* [2010]. Anyway, even in these cases, the average computation time is, at most, around $3s$. Note that a cluster of 10000 machines is not unrealistic, a full Google cluster is composed of 12500 machines (Gog *et al.* [2016]).

4.5 Simulations

In this section, we are interested in both MAKESPAN-MAPREDUCE and COMMUNICATION-MAPREDUCE problems. As said previously, both problems can be solved in polynomial time. In a first time we consider settings where all tasks have the same uniform computation time, the perfect case on

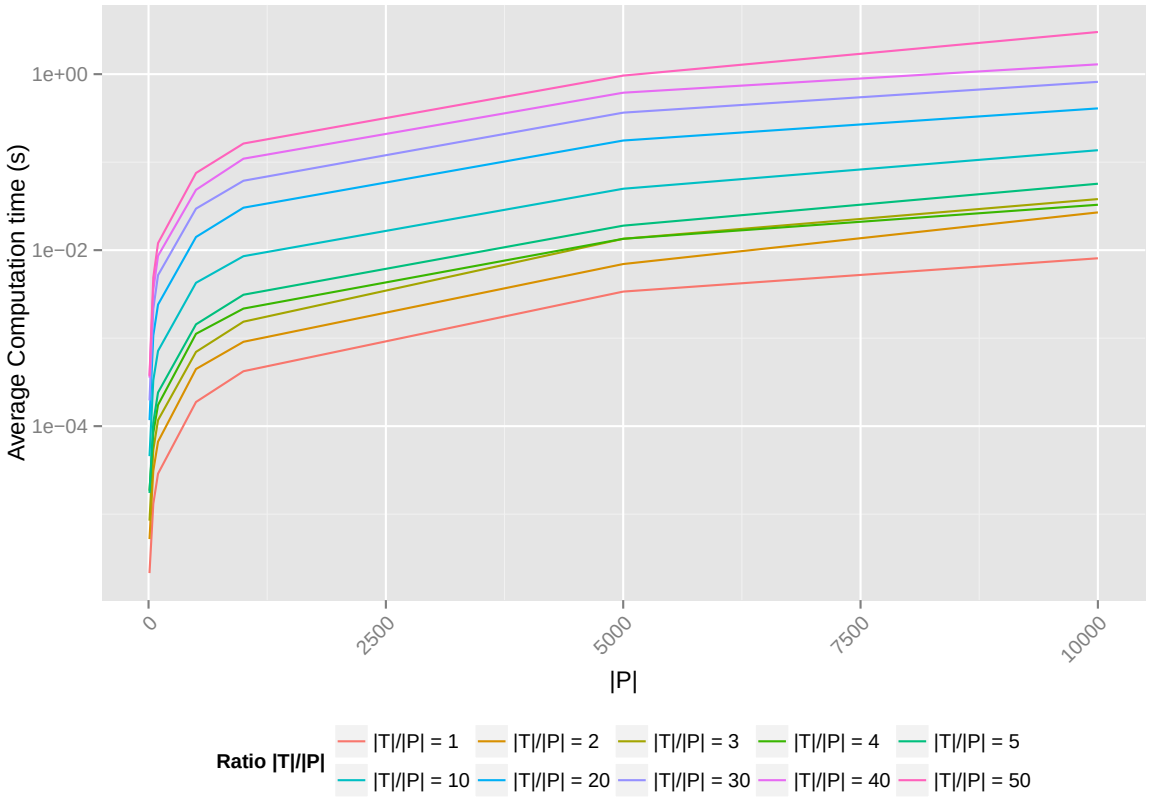


Figure 4.7: Average computation time in second of BestAssignment for different values of $\frac{|T|}{|P|}$ and $|P|$.

which BestAssignment is optimal. In this situation we want to compare it with pre-existing strategies in order to evaluate the gain from this optimal strategy. Secondly, we work on the modelling with on-line heterogeneous tasks, *i.e.* tasks for which the duration is not known and may be different from one to another. In this case we want to test the resilience of BestAssignment-based strategies in practical settings.

4.5.1 Settings

In this section we present the settings we use later for our simulations.

Strategies

We compare three main strategies: Greedy-based, BestAssignment-based and Maestro-based. The first two ones are based on Greedy and BestAssignment defined early. The last one uses a strategy proposed in Ibrahim

et al. [2012]. Basically the Maestro is a greedy two-waves scheduler. In order to explain the functioning of MAESTRO, we introduce the parameters defined by the authors

- For a processor p_i, HCN_i denotes its host chunk number, *i.e.* the number of data chunks from unprocessed tasks present on the processor.
- For a task t_j, its chunk weight is $Cw_j = 1 - \left(\sum_{i=i_1}^{i=r} \frac{1}{HCN_i} \right)$ where $p_{i_1}, \ldots, p_{i_r}$ are the processors that host the duplicates of the data chunk of t_j. This value represents the probability for t_j to be non-locally executed. If a chunk is on a processor with many other chunks, then it is more likely to be executed by another processor (because the first processor may not finish its other tasks on time).
- For two processors $p_i, p_{i'}$, $Sc_i^{i'}$ denotes the number of shared chunks between p_i and $p_{i'}$. With the graph notation, if $G = (P, T, E)$ is the graph representing the chunk repartition, $Sc_i^{i'} = |\{t_j,\ \{p_i, t_j\} \in E \text{ and } \{p_{i'}, t_j\} \in E\}|$.
- For a processor p_i, its node weight is $NodeW_i = \sum_{j=1}^{HCN_i} \left(1 - \sum_{k=1}^{r} \frac{1}{HCN_{i_{j,k}} + Sc^i_{i_{j,k}}} \right)$ where $(p_{j,1}, \ldots, p_{j,r})$ are the processors that host data chunk of t_j. Quickly, the higher is $NodeW_i$, the less p_i has a chance to process non-local task.

During the first phase, MAESTRO assigns $|P|$ tasks, one by processor. To do this, it first chooses the processor p_i with the lowest $NodeW_i$ (a processor with few hosted chunks or/and many shared chunks with other critical processors) and allocates to p_i the task whose chunk is present on p_i with the largest chunk weight (*i.e.* the task with the most chance to be non-locally executed). More precisely MAESTRO gives to the processor with highest chance to process non-local tasks the task it hosts with the highest chance to be processed non-locally, see Algorithm 4.6.

During its second phase, that is purely dynamic, each time a processor is idle, the local task with the highest chunk weight is assigned to this processor and processed. If there is no local task, an arbitrary unprocessed task is processed after the loading of the data chunk (if non-local tasks are allowed, otherwise the processor stays idle), see Algorithm 4.7.

We can now define the strategies. First if non-local tasks are forbidden (makespan metric):

- GREEDY-Makespan: uses GREEDY algorithm to dynamically perform the assignment.
- Static-BESTASSIGNMENT-Makespan: uses the assignment statically computed by BESTASSIGNMENT. Note that on uniform settings, any optimal-certified algorithm will perform with the same efficiency.

Algorithm 4.6 : MAESTRO-First-Wave

```
for k = 1 to |P| do
    p_i = the processor with highest NodeW_i ;
    t_j = the task present on p_i with the highest Cw_j ;
    execute t_j on p_i ;
    remove t_j ;
    remove p_i ;
    foreach p_i' do
        compute NodeW_i' ;
    foreach t_j' do
        compute Cw_j' ;
```

- BESTASSIGNMENT-Makespan: uses the assignment statically computed by BESTASSIGNMENT. If a processor is idle with no pre-assigned tasks available, it randomly chooses an unprocessed task among those for which a chunk is present on it. This strategy is only used in Section 4.5.3 to improve adaptability for online heterogeneous scheduling.

- MAESTRO-Makespan: uses the two waves from MAESTRO to statically and then dynamically compute the assignment. Note that for the second wave non-local-tasks are forbidden and thus processors stay idle when all their tasks have been performed.

In the second case (communication metric):

- GREEDY-Communications: uses GREEDY algorithm to dynamically perform the assignment. If a processor is idle without local tasks, it randomly picks a task among the ones present on the processor with the largest number of chunks at this instant.

- BESTASSIGNMENT-Communications: uses the assignment statically computed by BESTASSIGNMENT. If a processor is idle with no assigned tasks, it processes one of its local tasks. If there are no longer unprocessed local tasks, then the processor steals a task among the local tasks of the processor with the largest number of unprocessed assigned tasks. This stealing procedure is close to the transformation in Section 4.1.2. If a processor is idle, this is the case where its degree is below $\left\lceil \frac{|T|}{|P|} \right\rceil$. As the stolen processor has the largest degree, it is above $\left\lceil \frac{|T|}{|P|} \right\rceil$. If the stolen task is local, the stolen processor degree may be below $\left\lceil \frac{|T|}{|P|} \right\rceil$ but in this case this is equivalent to applying an alternating path without changing the load imbalance.

Algorithm 4.7 : MAESTRO-Second-Wave

```
while There is unprocessed tasks do
    Wait for an idle processor p_i ;
    if There is local chunk on p_i then
        t_j = the task present on p_i with the highest Cw_j ;
        execute t_j on p_i ;
        remove t_j ;
        foreach t_j' do
            compute Cw_j' ;
    else
        t_j = a random non-local task ; execute t_j on p_i ;
        remove t_j ;
        foreach t_j' do
            compute Cw_j' ;
```

- MAESTRO-Communications: uses the two waves from MAESTRO to statically and then dynamically computes the assignment.

Note that in both cases, the MAESTRO strategies slightly differ from Ibrahim *et al.* [2012] as we do not launch speculative tasks (duplicate tasks in case of presumed failure). Therefore the results of MAESTRO in our simulations are slightly better for the communications metric.

Traces

In order to produce realistic cases for the heterogeneous settings we use the traces from Kavulya *et al.* [2010]. These traces come from a recording of jobs during 10 months on a Hadoop cluster. In order to emulate the on-line aspect, the computational time of each task is randomly picked at the beginning of its simulated execution. In the following, we focus on jobs with at most 10000 tasks (more than 95% of the jobs). In addition we classify the jobs depending on their Normalized Standard Deviation (NSD), *i.e.* standard deviation of the computation times divided by the mean of these computation times, see Table 4.1.

4.5.2 Homogeneous Settings

In this section we focus on the case where all tasks have the same computational time (so not based on traces). For each strategy we run 250 simulations for $|P| \in \{10, 50, 100\}$, for $\frac{|T|}{|P|} \in \{1, 2, 3, 4, 5, 10, 20, 30, 40, 50\}$ and for

	$NSD < 0.05$	$NSD \in [0.05, 0.1[$	$NSD \in [0.1, 0.25[$	$NSD \in [0.25, 0.5[$	$NSD \in [0.5, 1[$	$NSD \geq 1$	Total
$\|T\| < 50$	49 (2,41%)	109 (5,35%)	123 (6,04%)	60 (2,95%)	39 (1,92%)	26 (1,28%)	406 (19,94%)
$\|T\| \in [50, 100[$	18 (0,89%)	50 (2,46%)	93 (4,57%)	61 (3,00%)	34 (1,67%)	23 (1,13%)	279 (13,70%)
$\|T\| \in [100, 250[$	11 (0,54%)	75 (3,68%)	205 (10,07%)	110 (5,40%)	78 (3,83%)	25 (1,23%)	504 (24,75%)
$\|T\| \in [250, 500[$	10 (0,49%)	55 (2,70%)	105 (5,16%)	68 (3,34%)	50 (2,46%)	17 (0,83%)	305 (14,98%)
$\|T\| \in [500, 1000[$	1 (0,05%)	5 (0,25%)	21 (1,03%)	44 (2,16%)	33 (1,62%)	18 (0,88%)	122 (5,99%)
$\|T\| \in [1000, 5000[$	1 (0,05%)	10 (0,49%)	43 (2,11%)	32 (1,57%)	33 (1,62%)	76 (3,73%)	195 (9,58%)
$\|T\| \in [5000, 10000[$	0 (0,00%)	16 (0,79%)	57 (2,80%)	33 (1,62%)	16 (0,79%)	10 (0,49%)	132 (6,48%)
$\|T\| \geq 10000$	0 (0,00%)	9 (0,44%)	4 (0,20%)	14 (0,69%)	39 (1,91%)	27 (1,33%)	93 (4,57%)
Total	90 (4,42%)	329 (16,16%)	651 (31,97%)	422 (20,73%)	322 (15,82%)	222 (10,90%)	2036 (100%)

Table 4.1: Repartition in function of the number of tasks ($|T|$) and the normalized standard deviation (NSD) of the jobs from Kavulya *et al.* [2010].

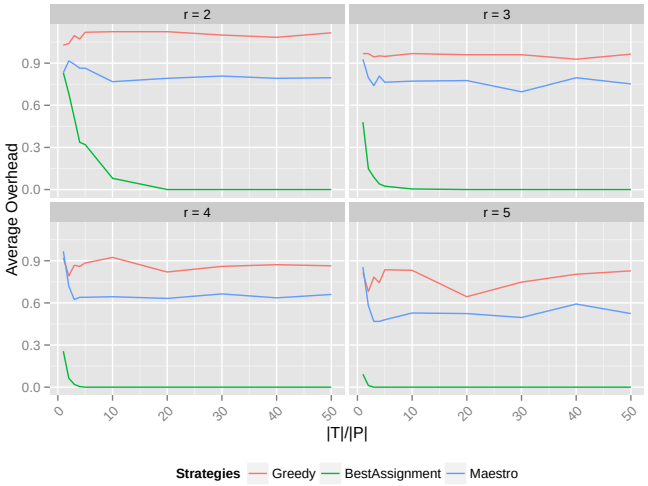
$r \in \{2, 3, 4, 5\}$ where r is the number of time each chunk is replicated ($r = 3$ in default HDFS, Borthakur [2008]).

Makespan

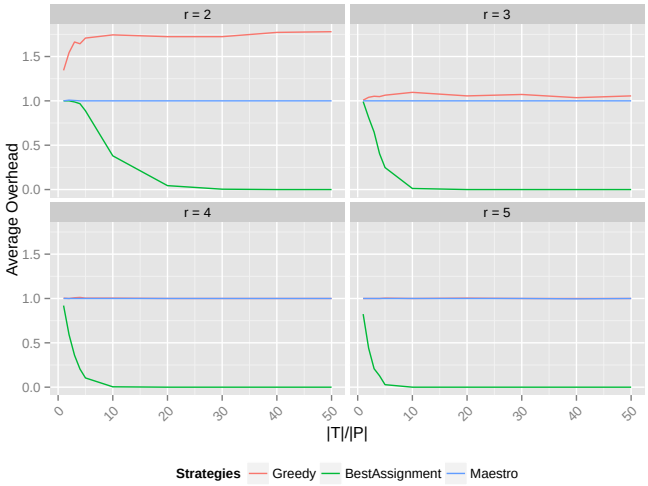
We first focus on the makespan metric, *i.e.* the case where non-local tasks are forbidden. As stated in Section 4.4.1, the asymptotically expected optimal makespan is $\frac{|T|}{|P|} + 1$ (small number of tasks per processor) or $\frac{|T|}{|P|}$ (large number of tasks per processor) and as proven in Sections 4.4.2 and 4.4.3 BestAssignment is optimal.

The results of our simulations are depicted on Figure 4.8. We present on each figure the average overhead, *i.e.* the makespan minus $\frac{|T|}{|P|}$ (the perfect load-balancing). First we can notice the reliability of the asymptotic theoretical results for "small" values. Indeed, even in the case of small $|P|$ or $|T|$ the makespan is below $\frac{|T|}{|P|} + 1$ (for more than 10^5 runs, there were only two cases where the optimal makespan was $\frac{|T|}{|P|} + 2$). Moreover, we can see a three phases behaviour of this optimal value (represented by BestAssignment). First, for small $|T|$ (*i.e.* small $\frac{|T|}{|P|}$) the optimal is $\frac{|T|}{|P|} + 1$. Then, during a transition phase, the average of this optimal value is between $\frac{|T|}{|P|}$ and $\frac{|T|}{|P|} + 1$ and finally, for large values of $|T|$ the optimal is $\frac{|T|}{|P|}$. We clearly see what Theorem 4.5 states, the case $|T| \leq \lambda |P| \log |P|$ at first and the case $|T| \geq c|P| \log |P|$ at the end. Note that according to our results, constants c and λ depend on the value of r.

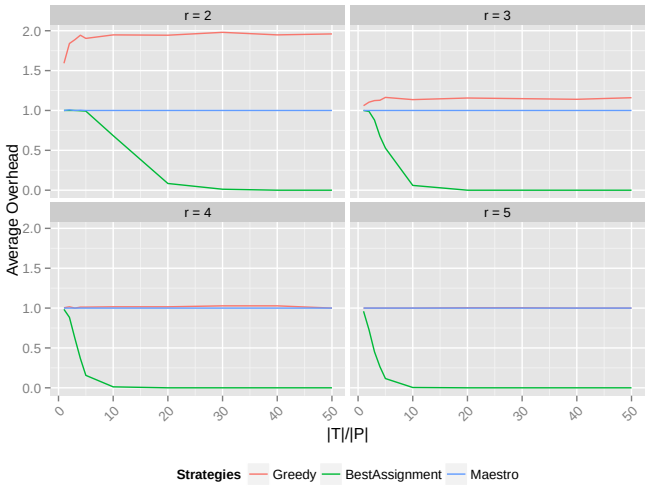
For the results of Greedy and Maestro we note that both are, in average, above $\frac{|T|}{|P|} + 1$ and, except sometimes for $|T| = |P|$, never optimal (*i.e.* equal to BestAssignment result). Except for $|P| = 10$ and for small values of $|T|$, both are really stable, in particular Maestro whose makespan is almost always equal to $\frac{|T|}{|P|} + 1$ for $|P| = 50$ and $|P| = 100$. Globally we can see a clear gain from the use of an optimal static strategy in place of a dynamic one (Maestro strongly relies on its second wave, that is purely dynamic, when $|T| > |P|$).



(a) $|P| = 10$



(b) $|P| = 50$



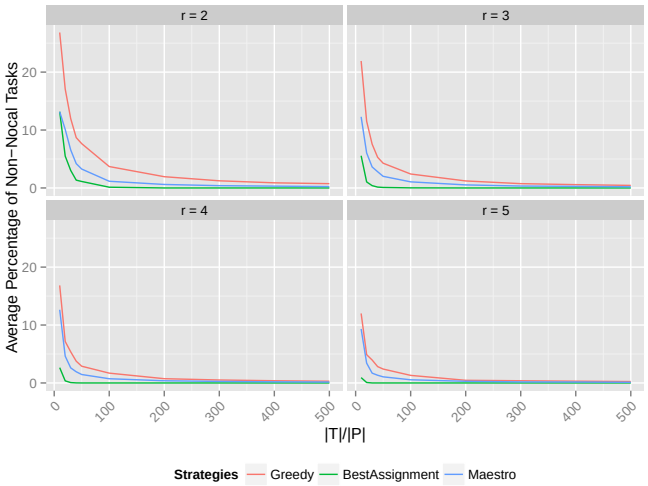
(c) $|P| = 100$

Figure 4.8: Results for the makespan metric and homogeneous settings.

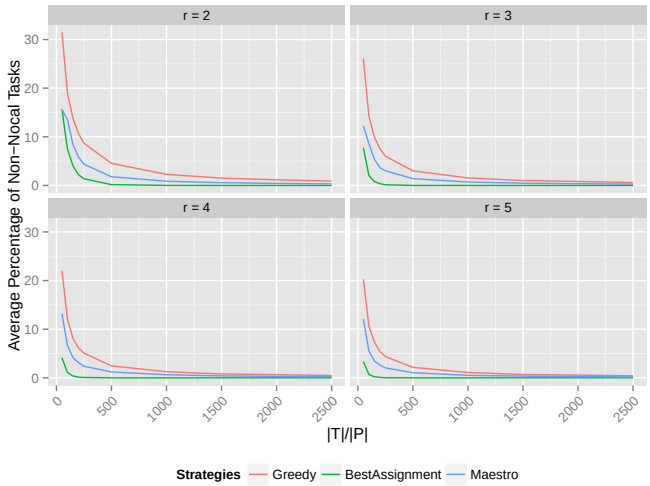
Communication

We now focus on the case of the communication metric, *i.e.* the case where we allow non-local tasks. Unlike the makespsan metric, there are few theoretical studies that we can relate to this problem. First we already proved the optimality of BestAssignment (in Sections 4.4.2 and 4.4.3). Secondly, Berenbrink *et al.* [2008] introduce the number of "holes" in an assignment, which corresponds to $\sum_{p_i \in P} \max(0, \left\lceil \frac{|T|}{|P|} \right\rceil - |\{t_j \text{ assigned to } p_i\}|)$. When $\frac{|T|}{|P|}$ is an integer, this is equal to the load imbalance. In Berenbrink *et al.* [2008], it

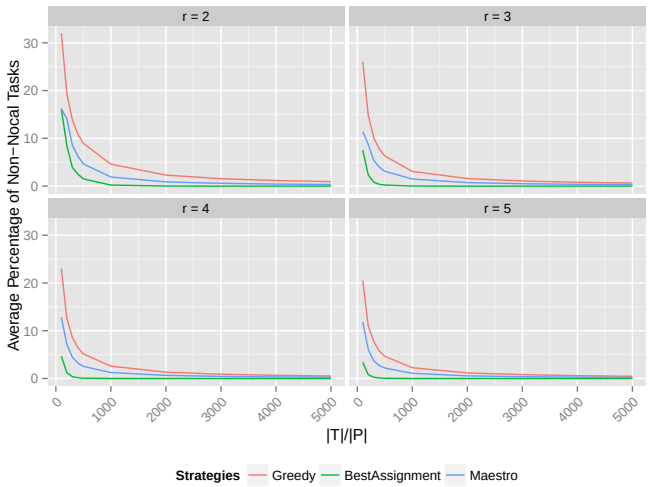
is proven that in the case of Balls-In-Bins processes with multiple choices (or, as proven in Section 4.3.2, in the case of Greedy), the above metric is bounded by a linear function of $|P|$. To finish, we notice that an assignment of maximal degree $\frac{|T|}{|P|}$ has a load imbalance of 0. Thus, we know that there exists a constant c such that, with high probability, there exists a solution with no communication if $|T| \geq c|P| \log |P|$.



(a) $|P| = 10$



(b) $|P| = 50$



(c) $|P| = 100$

Figure 4.9: Results for the communication metric and homogeneous settings.

The results of our simulations are depicted on Figure 4.9. Like for the makespan metric there are several phases for the BestAssignment strategy. In the first phase there is not always a solution without communications, *i.e.*

there is not always a solution of MAKESPAN-MAPREDUCE with maximal degree equal to $\frac{|T|}{|P|}$ (there is superposition of this phase with the two first observed in the makespan metric). In the second phase, BESTASSIGNMENT manages to completely avoid communications if there are enough tasks in comparison of the number of processors. Note that even in the first case the number of non-local-task is below 15% in the worst case and more likely below 10% or 5% when r is above 2.

Like for the makespan metric, there is a clear hierarchy between GREEDY and MAESTRO: MAESTRO performs slightly better, in particular for small number of tasks per processor (where GREEDY can reach more than 30% of non-local-tasks). However if MAESTRO and GREEDY seem to approach BESTASSIGNMENT in the case of high number of tasks per processor, the percentage of non-local tasks is always strictly above 0. This fact can be seen clearly in Figure 4.10 but also on Figure 4.8 (if there is no communication, then the results would be optimal for the makespan metric). In addition, we also notice the small impact of $\frac{|T|}{|P|}$ on the performance of MAESTRO and GREEDY (except for small values). Their performance seem to be more linked to $|P|$ and r, what is consistent with the claims from [Berenbrink *et al.*, 2008].

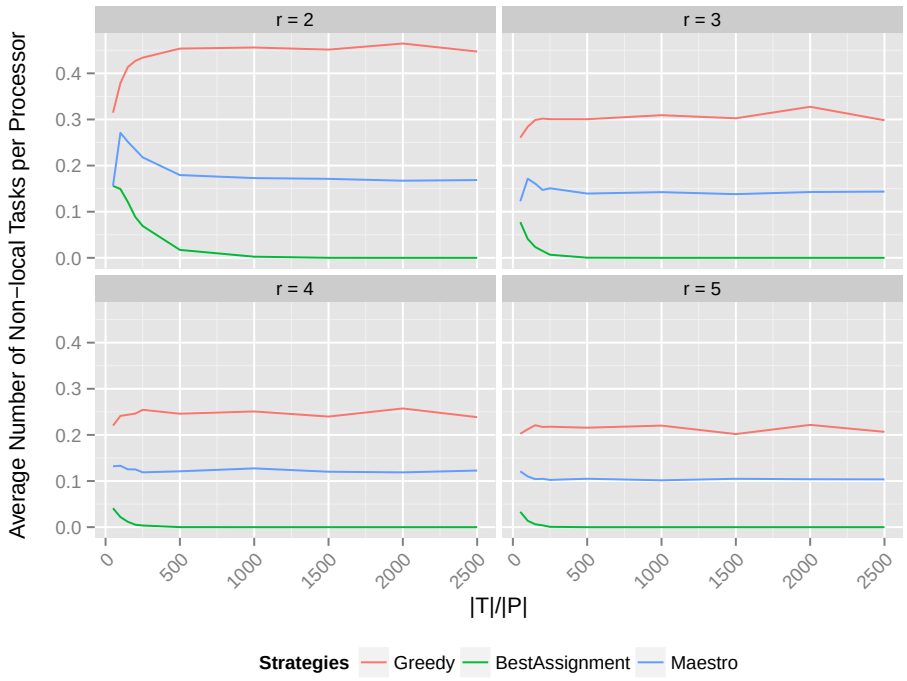
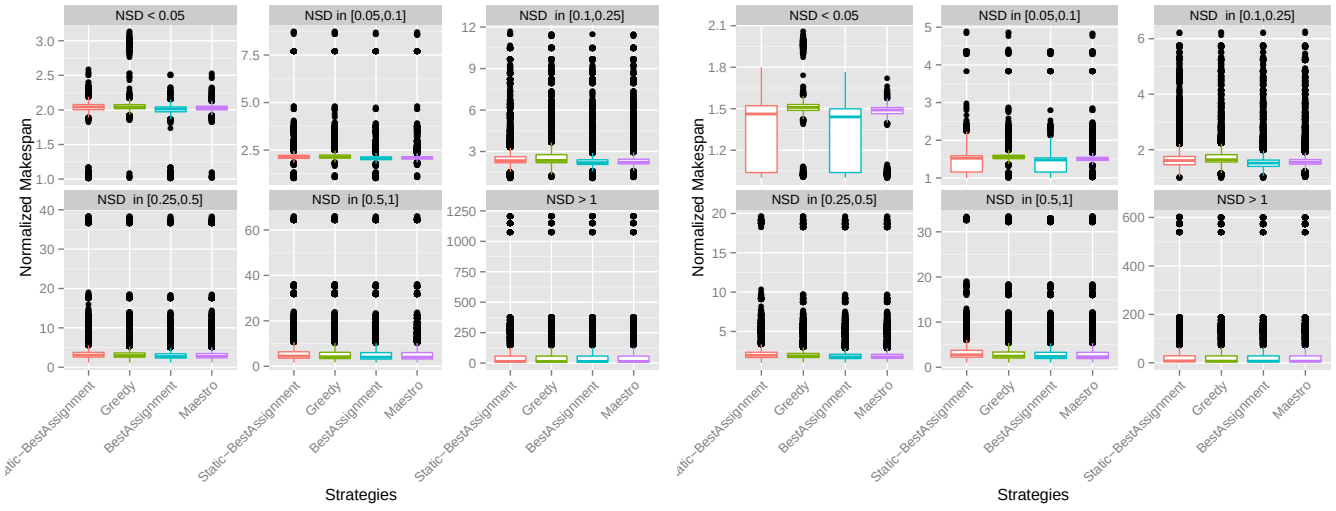


Figure 4.10: Non-local tasks per processor for homogeneous settings and $|P| = 50$.

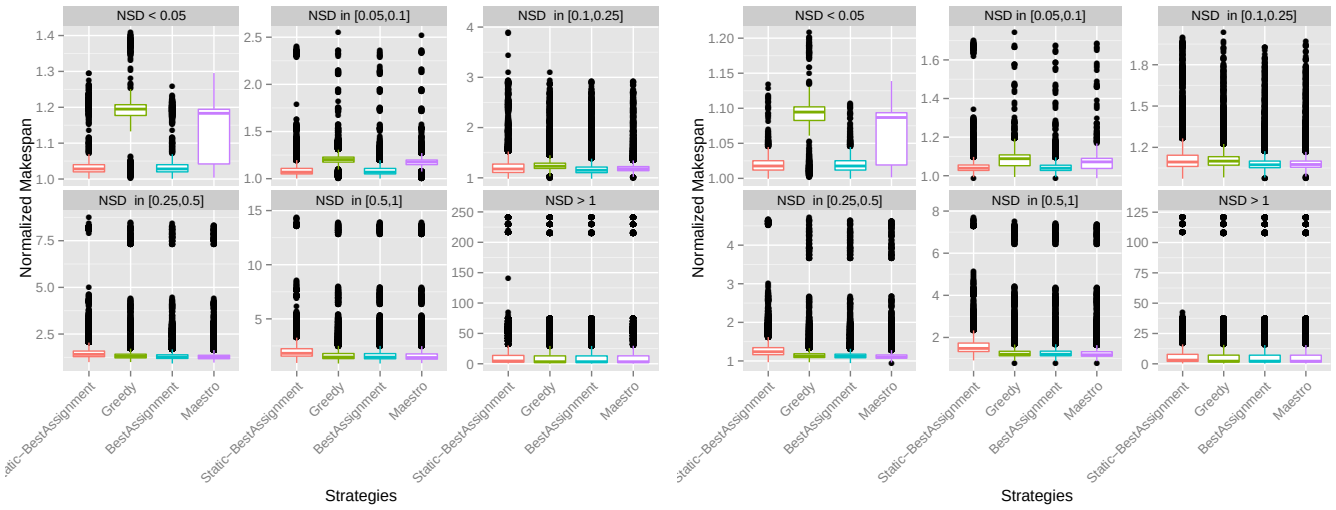
In a more general way, for both metrics, the increase of r slightly improves

the results (and mainly benefits to BestAssignment), but the gain becomes less and less important. Therefore, the value $r = 3$ proposed by HDFS seems to be a good trade-off between replication cost and communication-avoiding parallel execution.

4.5.3 Heterogeneous Settings



(a) $\frac{|T|}{|P|} = 1$ (b) $\frac{|T|}{|P|} = 2$



(c) $\frac{|T|}{|P|} = 5$ (d) $\frac{|T|}{|P|} = 10$

Figure 4.11: Results for the makespan metric and heterogeneous settings.

In this section we propose simulation on the traces presented in Section 4.5.1. As stated the scheduling algorithms do not use the information we have on the

execution time (neither for static or dynamic part) in order to emulate the online aspect of MapReduce scheduling. In this section we also set the replication ratio r to 3. Furthermore, for each job, we perform simulations for different values of $\frac{|T|}{|P|}$, $|T|$ being set by the traces. Finally, for each job, each $\frac{|T|}{|P|} \in \{1, 2, 5, 10\}$ and each strategy we perform 50 simulations.

Makespan

In order to provide a fair comparison between jobs with different average computational times, we use the normalized makespan, *i.e.* the actual makespan divided by the ideal one (the purely sequential makespan divided by the number of processors), as a metric. The results are shown on Figure 4.11.

Many outliers can be noticed in these results, with important normalized makespan in particular in the case of high variance. The explanation is that, in high variance jobs, there are tasks that last really longer than the other ones. Therefore these tasks have a significantly more important impact on the makespan than the others and then they partially cancel the efforts made to balance the number of tasks between processors (this is also why the outliers are present on every strategy when the variance is high enough).

Globally the general behaviour is more or less the same for all $\frac{|T|}{|P|}$. For small variance ($NSD \leq 0.1$), we are close to the result for homogeneous settings. Thus BestAssignment and Static-BestAssignment are more efficient than Greedy and Maestro. Otherwise, all three strategies with dynamic part have similar efficiency and Static-BestAssignment suffers from the homogeneity assumption.

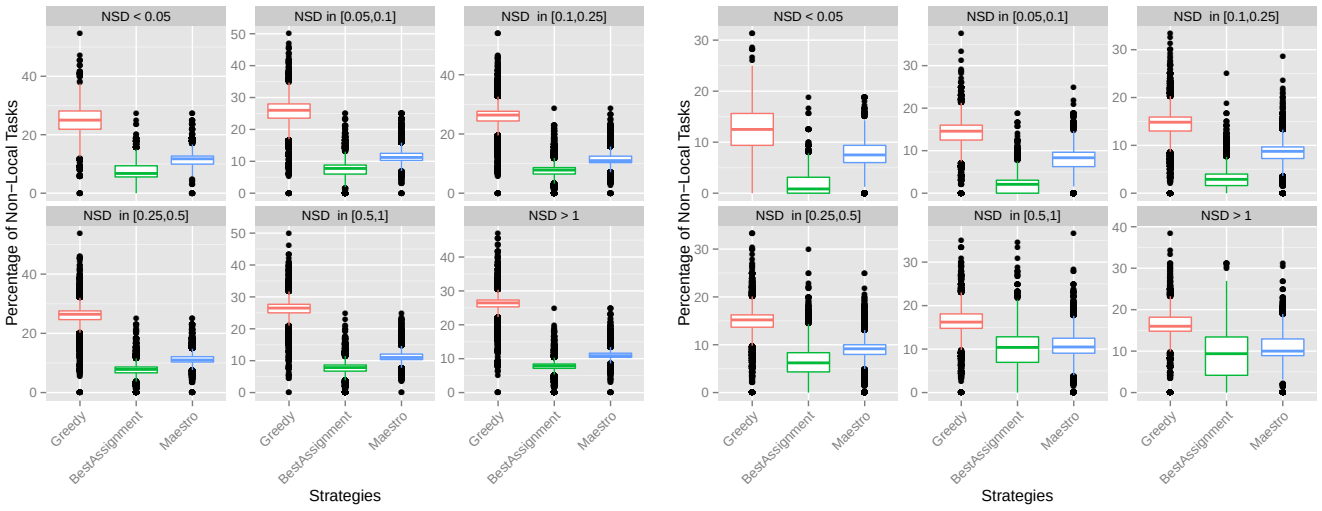
Therefore, in any case, BestAssignment, with its dynamic part, appears to be the better choice. It is the best strategy for small variances and it performs as well as the other dynamic strategies for large variances.

Communications

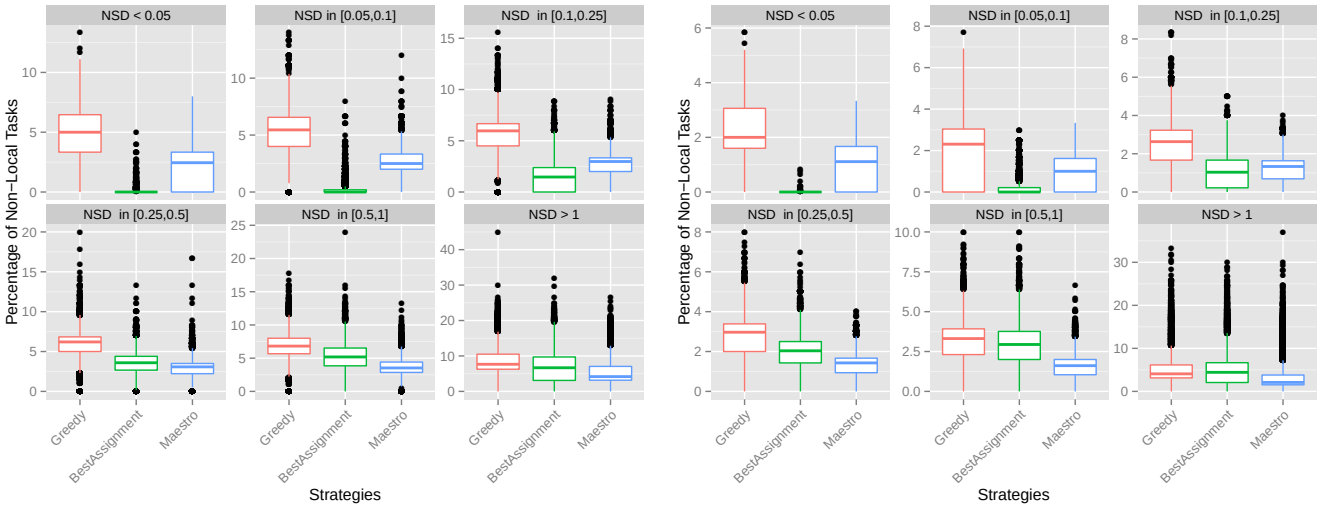
As for the homogeneous settings we use the percentage of non-local tasks as a metric. Results can be found on Figure 4.12.

Let us first have a look at the case $|T| = |P|$ (Figure 4.12(a)). According to our strategies there can be no idle processor as soon as there is at least one unprocessed task. Therefore each processor executes exactly one task (if the assignment gives no tasks to a processor, this one will automatically steal one to another processor). Thus, the heterogeneity of the computational times has no effect here on communication cost. Thus, for all standard deviation, BestAssignment performs well with an average cost around 7.5% of non-local tasks against 11.4% for Maestro and 25.8% for Greedy. Note that the small differences that appear between the results of two classes with different standard deviations might probably come from the repartition of the value $|T|$

(the number of tasks of a job, which, according to the results with homogeneous settings, impacts the communication ratio, even with $|T| = |P|$) inside each class, which differs from a class to another (see Table 4.1).



(a) $\frac{|T|}{|P|} = 1$ (b) $\frac{|T|}{|P|} = 2$



(c) $\frac{|T|}{|P|} = 5$ (d) $\frac{|T|}{|P|} = 10$

Figure 4.12: Results for the communication metric and heterogeneous settings.

In the other cases, $\frac{|T|}{|P|} > 1$, the results are not as one-sided. First, as expected, the lower is the variance, the better BESTASSIGNMENT is (the assumption stating that every task has the same computational time, assumption made by the static scheduler, becomes wrong when standard deviation increases). For the two mostly-dynamic strategies, GREEDY and MAESTRO, we

can notice a stability of the results that are no so much impacted by the heterogeneity. This stability added with the degradation of the results of BestAssignment implies that Maestro can be more efficient in the case of a huge variance. For example, in the case $\frac{|T|}{|P|} = 5$ (Figure 4.12(c)), for $NSD < 0.1$, BestAssignment is close to its results in the homogeneous settings with, on average, only 0.13% of non-local-tasks when Maestro reaches 2.61% and Greedy is close to 4.75%. However the gap decreases for $NSD \in [0.1, 0.25[$ (1.51% against 2.67%) and finally, for greater NSD, Maestro performs better (5.13% for BestAssignment against 3.96% for Maestro). Note that jobs with NSD below 0.25 represent slightly more than half of the jobs (see Table 4.1).

The other important factor on the efficiency of the different strategies is the ratio $\frac{|T|}{|P|}$. It appears clearly that the strategy globally produce less communication (proportional to the number of tasks) when the ratio $\frac{|T|}{|P|}$ increases (this is not strict, for example, for $NSD > 1$, BestAssignment has only, in average, 7.62% of non-local tasks when $\frac{|T|}{|P|} = 1$ against 9.28% when $\frac{|T|}{|P|} = 2$). This was already the case for homogeneous settings, but here, coupled with the influence of variance, this makes BestAssignment's efficiency decreases in comparison to dynamic strategies. For example, for $\frac{|T|}{|P|} = 1$ and $\frac{|T|}{|P|} = 2$, BestAssignment is the most effective strategy even for large standard deviation. However, as stated above, for $\frac{|T|}{|P|} = 5$ there are cases where BestAssignment is less efficient than Maestro and for $\frac{|T|}{|P|} = 10$ BestAssignment is even challenged by Greedy in the case $NSD > 1$. In order to see if this degradation of performance for BestAssignment continues for larger ratio $\frac{|T|}{|P|}$, we propose in Figure 4.13 an illustration of the case $\frac{|T|}{|P|} = 50$. In this case, the only case where BestAssignment is the indisputably better strategy is when $NSD < 0.05$, that represents few cases.

This phenomenon comes from the opposition between static and dynamic. In BestAssignment the scheduler, in its first phase, tries to assign to each processor exactly $\frac{|T|}{|P|}$ tasks. In Maestro or Greedy, tasks are assigned when a processor needs one. When heterogeneity increases, the flexibility of dynamic strategies is compulsory, BestAssignment is too rigid and its dynamic wave starts too late. However, even in the case where BestAssignment is no the best strategy, its performance is not so bad. For example, in the case $\frac{|T|}{|P|} = 50$, BestAssignment is on average below 2% of non local tasks. More generally, a number of tasks below 10% is a good expectation (on Figure 4.12, the only time the average is above 10% is for $\frac{|T|}{|P|} = 2$ and $NSD > 0.5$, but in this case Maestro is even worse).

Therefore, in conclusion of this section, BestAssignment appears to be a good solution if either:

- the number of tasks per processor is low,

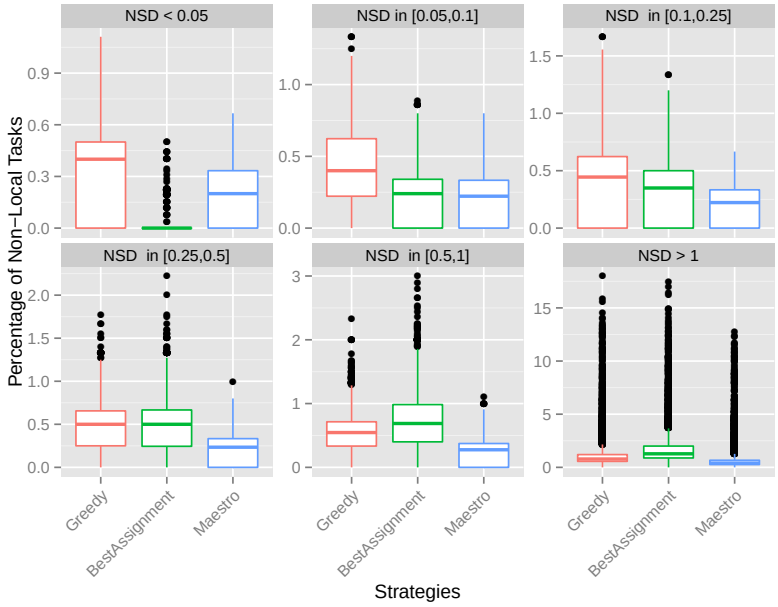


Figure 4.13: Results for the communication metric for $\frac{|T|}{|P|} = 50$.

- the task can be guaranteed to have similar computation time.

If one or two of these conditions apply, BestAssignment outperforms Maestro. Otherwise, BestAssignment may not be the best strategy but still ensures few communications.

4.6 Conclusion and Perspectives

In this chapter, during Section 4.1, we first introduced MAKESPAN-MAPREDUCE and COMMUNICATION-MAPREDUCE, the two theoretical problems we studied to improve makespan and data locality during the Map phase in MapReduce. We also provided a modelling purely based on bipartite graph that use a simple object, assignment, to solve MAKESPAN-MAPREDUCE and COMMUNICATION-MAPREDUCE. Then, in Section 4.2 we presented pre-existing studies on similar problems and on matching in bipartite graph, a problem that we used later to solve MAKESPAN-MAPREDUCE and COMMUNICATION-MAPREDUCE. In Section 4.3 we proposed a theoretical study of a basic dynamic scheduler, Greedy, by relating it to the classical Balls-In-Bins process, allowing to use important probabilistic results from the literature, in particular an expected makespan of $\frac{|T|}{|P|} + O(\log\log|P|)$. In Section 4.4 we related MAKESPAN-MAPREDUCE to the GRAPH-ORIENTIABILITY problem. From GRAPH-ORIENTIABILITY we deduced a polynomial time algorithm and adapt a result from the literature to prove the existence, with

high probability, of near-perfect assignment (*i.e.* with maximum degree inferior to $\left\lceil \frac{|T|}{|P|} \right\rceil + 1$). Then, in the same section, using techniques close to the ones used in flow problems, we proposed FindAssignment and BestAssignment, two polynomial algorithms that are optimal for both problems, MAKESPAN-MAPREDUCE and COMMUNICATION-MAPREDUCE. Finally, in Section 4.5, we provided simulation results and proved that with the addition of a small dynamic part, BestAssignment provides very efficient schedules, except when the variance between computation times in a same job is too high, where it can be outperformed by a (mostly) dynamic scheduler like Maestro.

For the continuation of this work there are several work directions. First, if optimal solutions can be quickly computed on static settings, Section 4.5 shows that these solutions can suffer in the case of heterogeneous processing times (but supposed homogeneous during the scheduling). An intuitive improvement of the model to solve this problem could be to add weights on task nodes representing computation times of each task. We can even be more general and consider a computation time for each task on each processor. Note that in both cases, the problems of minimizing makespan or communication (in the second case the condition "with optimal makespan" shall become "with makespan below a given deadline") are NP-Complete as they contain $P||C_{\max}$, the problem of scheduling tasks on P machines minimizing the makespan, that is itself NP-complete with heterogeneous task-durations, see Garey and Johnson [2002]. The resolution of such problems makes a use of the literature on weighted Balls-In-Bins, see Berenbrink *et al.* [2008]. However the literature on weighted matching on bipartite graphs, see Chapter 3 of West *et al.* [2001], is more focused on weights on edges.

However the design of MapReduce is based on equal splitting of the workload, aiming at homogeneous tasks. The heterogeneity observed in practice is often a consequence of unexpected behaviour and thus unpredictable. Therefore, having this computation time as input is unrealistic (the problem can still be interesting to look at, but its direct application to the Map phase scheduling might not be the good motivation). Therefore another approach might be to work on more resilient algorithms, probably with improvement of the dynamic part of hybrid algorithms.

Among the other changes we can make to the model is the addition of weights on edges (and not on task nodes). As stated in the introduction, the processors are split into racks, and the communication inside a rack is cheaper than the communication between racks. Therefore, to refine the model, weighted edges can bring this new information. In return, the bipartite graph will have more edges (we need to add edges between tasks and non-local processors with a weight that depend on the presence of this processor on a rack that owns the corresponding data chunk).

Always on this same problem of Map phase scheduling, the transformation

of our model into one closer of the one of Isard *et al.* [2009] and Gog *et al.* [2016], *i.e.* a model with only a makespan objective (more precisely a deadline) where each communication has a cost expressed in time, is an interesting perspective. In addition, the model is also made to consider concurrent jobs on a same cluster (with different begin times), which our model does not take into account.

To conclude this section, we propose another perspective with a new problem that could be considered as a merge of the problem of this chapter and the one of Chapter 1. Let us now assume that each task needs two data chunks as input instead of one. In this case, to process a task on a processor, this processor has to own both data chunks. If we now suppose that we are in a HDFS-like system, *i.e.* with data chunks already replicated and distributed among the processors, several questions can be asked. First, assuming we cannot dynamically add replicates (assumption that is equivalent to the interdiction of non-local tasks in MAKESPAN-MAPREDUCE), what level of replication is necessary to be able to compute the whole set of tasks with high probability? Indeed, each task needs two data chunks and if no processor simultaneously owns the two chunks, then this task cannot be processed in this configuration. Secondly, assuming it is possible to compute this set, or at least a part of it, what are the schedules that minimize the makespan or the number of duplicates created during the execution?

Conclusion

Contributions

The contributions of this thesis are the following.

In Chapter 1, we focus on PERI-SUM, a square partitioning problem that is used as model for communication-avoiding parallel matrix multiplication. The main contribution of this chapter is the improvement of the theoretical results of the literature on PERI-SUM with the design of two approximation algorithms, NRRP and SFCP. In addition, we propose comparison between static and dynamic approaches for this problem thanks to a set of simulations. We proved that static strategies, augmented with work-stealing technique, can be reliable for the particular problem of parallel matrix multiplication.

In Chapter 2, we introduce MSCubeP, an alternative model for communication-avoiding parallel matrix multiplication. Shortly, in MSCubeP model, the entire set of elementary tasks is partitioned while, with PERI-SUM, only the output matrix is partitioned. In this chapter, we provide an NP-completeness proof for MSCuboidP, a more general variant of MSCubeP. In addition, we also design two approximation algorithms: 3D-NRRP and 3D-SFCP, the 3D-counterparts of SNRRP (a simplified version of NRRP) and SFCP respectively. Finally, we compare the efficiency of solutions of PERI-SUM with solutions of MSCubeP and prove that the second ones are significantly better than the first ones in terms of induced communications, except for small instances where solutions of PERI-SUM perform slightly better.

The algorithms from Chapter 1 and Chapter 2 are compared in Chapter 3 in a practical implementation of a parallel matrix multiplication scheduler using the StarPU library (Augonnet *et al.* [2011]). We propose a set of experiments on a computational platform (PlaFRIM2). From these experiments, we show that our mixed approach (a static scheduler augmented with work-stealing strategies) brings a small speed-up and a significant decrease in communications if compared with the default dynamic scheduler.

In Chapter 4, we introduce MAKESPAN-MAPREDUCE and COMMUNICATION-MAPREDUCE, two graph problems used as models of scheduling of the Map phase of MapReduce. These two problems differ only in their objective functions (makespan versus communication) and both output an assignment, *i.e.*

for each task an assigned processor. We relate MAKESPAN-MAPREDUCE with the probabilistic Balls-In-Bins model and with GRAPH-ORIENTIABILITY, allowing us to use the existing literature, in particular the existence of an optimal algorithm with polynomial computation time. In addition, we provide two algorithms with polynomial computation time (FindAssignment and BestAssignment) that are optimal for both MAKESPAN-MAPREDUCE and COMMUNICATION-MAPREDUCE. In the case of heterogeneous settings (different computation times for the tasks), we use simulations to evaluate the reliability of BestAssignment. In this case, the results are very encouraging: the number of non-local tasks decreases when the variance between the computation times is small enough.

In a more general way, one of the common threads between the different chapters is the static/dynamic opposition, as stated in the introduction. One of the goals of this thesis was to use the strengths of static strategies while limiting their weaknesses, notably by using dynamic components. In this respect, the results of the first three chapters are strong. In particular, in Chapter 3, we obtain with a practical implementation a clear gain both in terms of speed-up and communication decrease. In the matrix multiplication case, the very good quality of performance evaluation might partially explain the positive results (note that Section 1.3 seems to indicate a good reliability even with less favourable scenario on the quality of performance evaluation). Another explanation might also be the myopic vision of dynamic strategies.

In Chapter 4, we obtain more mitigated results for the Map phase scheduling problem. In a perfect scenario, with all tasks being exactly the same, we achieve optimal results. In a more realistic scenario, that we emulate with traces from a real Hadoop cluster, our static approach, even with a dynamic component, is sometimes less effective than pre-existing dynamic strategies, especially when the variance on the expected computation times is very large. Therefore we partially fail to avoid all the problems related to the static approach in this case. In addition to the good quality of the pre-existing algorithms, that make competition harder, another explanation might be that this time the assumption we made to work on our static strategy was too strict. More precisely, we assume a strong homogeneity and our strategy has important difficulties to recover when this assumption happens to be wrong. Yet, this pessimistic statement has to be tempered by the globally good results, in particular for low to medium variance cases.

Perspectives

In the different chapters we already elaborate many perspectives.

In Chapter 1, we first propose a possible theoretical improvement of partitioning algorithm by aiming at rectangles with lower aspect ratio. Secondly, we propose an extension of the PERI-SUM problem by adding heterogeneity in communication and computation costs. This extension could be used as a model of parallel sparse matrix multiplication or parallel FMM (Fast Multipole Method) computation.

In Chapter 2, in addition to further work on MSCubeP, we propose to extend the problem to hypercube (or hyperrectangle) partitioning. The main motivation of this future work would be the study of tensor contraction, an operation that can be seen as a generalisation of matrix product for multi-dimensional arrays.

In Chapter 3, we propose additional analysis in order to have a more complete view of the effect of our partitioning strategy, notably by working on different platforms.

In Chapter 4, there are two main perspectives. The first one is to work on the heterogeneity problem we encounter. This could be done by extending the model to heterogeneous computation times or by reinforcing hybrid strategies. The second perspective is an extension of the problem with each task having more than one input files, similarly to the tasks in the matrix multiplication case.

More generally, the strong results of static/dynamic mixed approach lead us to encourage the use of this strategy in other problems. This approach is more expensive than the use of a general dynamic scheduler, mainly because it requires important studies of a particular problem to design efficient static strategies. However, the gain from this theoretical work is significant and we want to keep going on this track, enjoying the many problems that parallel computing has to offer.

In addition, as stated in introduction, communication is a crucial issue that is common to HPC and Cloud computing. To make a perfect use of data, notably when replicated, the scheduler needs a large vision of the problem, that is in general one of the main limitations of dynamic approaches. With our static approach, *i.e.* augmented with work-stealing, we propose a research track to overrun this weakness, even in the case of general enough programming models like MapReduce. An interesting starting point for future studies would be to work on more general or complex models of parallel computing, with a focus on replication. Although building efficient static algorithms on general models can be difficult, the challenge is promising if we can provide reliable communication-avoiding solutions on models that could be, with the convergence between the two worlds, adapted to both HPC and Cloud computing.

Bibliography

Agullo, Emmanuel, Bramas, Bérenger, Coulaud, Olivier, Khannouz, Martin and Stanisic, Luka, 2016a. Task-Based Fast Multipole Method for Clusters of Multicore Processors. Technical report, Inria Bordeaux Sud-Ouest.

Agullo, Emmanuel, Buttari, Alfredo, Guermouche, Abdou and Lopez, Florent, 2016b. Implementing Multifrontal Sparse Solvers for Multicore Architectures with Sequential Task Flow Runtime Systems. *Transactions on Mathematical Software (TOMS)*, 43(2):13:1–13:22.

Albers, Susanne, 2003. Online algorithms: A survey. *Mathematical Programming*, 97(1):3–26.

Anderson, Michael, Ballard, Grey, Demmel, James and Keutzer, Kurt, 2011. Communication-avoiding QR Decomposition for GPUs. In *International Parallel and Distributed Processing Symposium (IPDPS)*, pages 48–58. IEEE.

Augonnet, Cédric, Thibault, Samuel, Namyst, Raymond and Wacrenier, Pierre-André, 2011. StarPU: A Unified Platform for Task Scheduling on Heterogeneous Multicore Architectures. *Concurrency and Computation: Practice and Experience*, 23(2):187–198.

Badia, R. M., Herrero, J. R., Labarta, J., Pérez, J. M., Quintana-Ortí, E. S. and Quintana-Ortí, G., 2009. Parallelizing Dense and Banded Linear Algebra Libraries Using SMPSs. *Concurrency and Computation: Practice and Experience*, 21(18):2438–2456.

Ballard, Grey, Demmel, James and Gearhart, Andrew, 2011a. Communication Bounds for Heterogeneous Architectures. Technical report, University of California at Berkeley.

Ballard, Grey, Demmel, James, Holtz, Olga, Lipshitz, Benjamin and Schwartz, Oded, 2012. Communication-optimal Parallel Algorithm for Strassen's Matrix Multiplication. In *Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 193–204. ACM.

Ballard, Grey, Demmel, James, Holtz, Olga and Schwartz, Oded, 2011b. Minimizing Communication in Linear Algebra. *Journal on Matrix Analysis and Applications*, 32(3):866–901.

Baptiste, Philippe, Le Pape, Claude and Nuijten, Wim, 2012. *Constraint-Based Scheduling: Applying Constraint Programming to Scheduling Problems*, volume 39. Springer.

Beaumont, Olivier, Boudet, Vincent, Petitet, Antoine, Rastello, Fabrice and Robert, Yves, 2001a. A Proposal for a Heterogeneous Cluster ScaLAPACK (dense linear solvers). *Transactions on Computers*, 50(10):1052–1070.

Beaumont, Olivier, Boudet, Vincent, Rastello, Fabrice and Robert, Yves, 2002. Partitioning a Square into Rectangles: NP-completeness and Approximation Algorithms. *Algorithmica*, 34(3):217–239.

Beaumont, Olivier, Cojean, Terry, Eyraud-Dubois, Lionel, Guermouche, Abdou and Kumar, Suraj, 2016a. Scheduling of Linear Algebra Kernels on Multiple Heterogeneous Resources. In *International Conference on High Performance Computing (HiPC 2016)*. IEEE.

Beaumont, Olivier, Eyraud-Dubois, Lionel, Guermouche, Abdou and Lambert, Thomas, 2015. Comparison of Static and Dynamic Resource Allocation Strategies for Matrix Multiplication. In *International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD), 2015*, pages 170–177. IEEE.

Beaumont, Olivier, Eyraud-Dubois, Lionel and Lambert, Thomas, 2016b. A New Approximation Algorithm for Matrix Partitioning in Presence of Strongly Heterogeneous Processors. In *International Parallel and Distributed Processing Symposium (IPDPS)*, pages 474–483. IEEE.

Beaumont, Olivier, Eyraud-Dubois, Lionel and Lambert, Thomas, 2016c. Cuboid Partitioning for Parallel Matrix Multiplication on Heterogeneous Platforms. In *International European Conference on Parallel and Distributed Computing (Euro-Par)*, pages 171–182. Springer.

Beaumont, Olivier, Lambert, Thomas, Marchal, Loris and Thomas, Bastien, 2017. Matching-Based Assignement Strategies for Improving Data Locality of Map Tasks in MapReduce. Technical Report RR-8968, Inria-Research Centre Grenoble–Rhône-Alpes; Inria Bordeaux Sud-Ouest.
URL `https://hal.inria.fr/hal-01386539v4/document`

Beaumont, Olivier, Larchevêque, Hubert and Marchal, Loris, 2013. Non Linear Divisible Loads: There is No Free Lunch. In *International Parallel and Distributed Processing Symposium (IPDPS)*, pages 863–873.

Beaumont, Olivier, Legrand, Arnaud, Rastello, Fabrice and Robert, Yves, 2001b. Static LU Decomposition on Heterogeneous Platforms. *International Journal of High Performance Computing Applications (IJHPCA)*, 15(3):310–323.

Beaumont, Olivier and Marchal, Loris, 2014. Analysis of Dynamic Scheduling Strategies for Matrix Multiplication on Heterogeneous Platforms. In *International Symposium on High-Performance Parallel and Distributed Computing (HPDC)*, pages 141–152. ACM.

Berenbrink, Petra, Czumaj, Artur, Steger, Angelika and Vöcking, Berthold, 2000. Balanced Allocations: The Heavily Loaded Case. In *Symposium on Theory of Computing (STOC)*, pages 745–754. ACM.

Berenbrink, Petra, Friedetzky, Tom, Hu, Zengjian and Martin, Russell, 2008. On weighted Balls-into-Bins Games. *Theoretical Computer Science (TCS)*, 409(3):511–520.

Berge, Claude, 1957. Two theorems in Graph Theory. *Proceedings of the National Academy of Sciences*, 43(9):842–844.

Blackford, L Susan, Choi, Jaeyoung, Cleary, Andy, D'Azevedo, Eduardo, Demmel, James, Dhillon, Inderjit, Dongarra, Jack, Hammarling, Sven, Henry, Greg, Petitet, Antoine *et al.*, 1997. *ScaLAPACK Users' Guide*, volume 4. SIAM.

Bleuse, Raphael, Kedad-Sidhoum, Safia, Monna, Florence, Mounié, Grégory and Trystram, Denis, 2015. Scheduling Independent Tasks on Multi-Cores with GPU Accelerators. *Concurrency and Computation: Practice and Experience*, 27(6):1625–1638.

Blumofe, Robert D and Leiserson, Charles E, 1999. Scheduling Multithreaded Computations by Work Stealing. *Journal of the ACM (JACM)*, 46(5):720–748.

Bollobás, Béla, 2013. *Modern Graph Theory*, volume 184. Springer.

Borthakur, Dhruba, 2008. HDFS Architecture Guide. *HADOOP http://hadoop. apache. org/common/docs/current/hdfs design. pdf*, page 39.

Bosilca, George, Bouteiller, Aurélien, Danalis, Anthony, Faverge, Mathieu, Hérault, Thomas and Dongarra, Jack, 2013. PaRSEC: A Programming Paradigm Exploiting Heterogeneity for Enhancing Scalability. *Computing in Science and Engineering*, 15(6):36–45.

Breinholt, Greg and Schierz, Christoph, 1998. Algorithm 781: Generating Hilbert's Space-filling Curve by Recursion. *Transactions on Mathematical Software (TOMS)*, 24(2):184–189.

Bueno, J., Planas, J., Duran, A., Badia, R.M., Martorell, X., Ayguade, E. and Labarta, J., 2012. Productive Programming of GPU Clusters with OmpSs. In *International Parallel and Distributed Processing Symposium (IPDPS)*, pages 557–568.

Butz, Arthur R, 1971. Alternative Algorithm for Hilbert's Space-Filling Curve. *Transactions on Computers*, 100(4):424–426.

Cain, Julie Anne, Sanders, Peter and Wormald, Nick, 2007. The random graph threshold for k-orientiability and a fast algorithm for optimal multiple-choice allocation. In *Symposium on Discrete Algorithms (SODA)*, pages 469–476. SIAM.

Chervenak, Ann, Foster, Ian, Kesselman, Carl, Salisbury, Charles and Tuecke, Steven, 2000. The Data Grid: Towards an Architecture for the Distributed Management and Analysis of Large Scientific Datasets. *Journal of Network and Computer Applications*, 23(3):187–200.

Choi, Jaeyoung, Demmel, James, Dhillon, Inderjiit, Dongarra, Jack, Ostrouchov, Susan, Petitet, Antoine, Stanley, Ken, Walker, David and Whaley, R Clinton, 1996. ScaLAPACK: A Portable Linear Algebra Library for Distributed Memory Computers: Design Issues and Performance. In *Computer Physics Communications*, volume 97, pages 1 – 15. Elsevier.

Chowdhury, Mosharaf and Stoica, Ion, 2012. Coflow: A Networking Abstraction for Cluster Applications. In *Workshop on Hot Topics in Networks (HotNets)*, pages 31–36.

Chowdhury, Mosharaf and Stoica, Ion, 2015. Efficient Coflow Scheduling Without Prior Knowledge. *Computer Communication Review (CCR)*, 45(5):393–406.

Cirne, Walfredo, Brasileiro, Francisco, Paranhos, Daniel, Góes, Luís Fabrício W and Voorsluys, William, 2007. On the Efficacy, Efficiency and Emergent Behavior of Task Replication in Large Distributed Systems. *Parallel Computing*, 33(3):213–234.

Clarke, David, Ilic, Aleksandar, Lastovetsky, Alexey and Sousa, Leonel, 2012. Hierarchical Partitioning Algorithm for Ccientific Computing on Highly Heterogeneous CPU + GPU Clusters. In *International European Conference on Parallel and Distributed Computing (Euro-Par)*, pages 489–501. Springer.

Cojean, Terry, Guermouche, Abdou, Hugo, Andra, Namyst, Raymond and Wacrenier, P-A, 2016. Resource Aggregation for Task-Based Cholesky Factorization on Top of Heterogeneous Machines. In *International European Conference on Parallel and Distributed Computing (Euro-Par) Workshop*. Springer.

Coppersmith, Don and Winograd, Shmuel, 1990. Matrix Multiplication via Arithmetic Progressions. *Journal of symbolic Computation*, 9(3):251–280.

Czumaj, Artur, Riley, Chris and Scheideler, Christian, 2003. Perfectly Balanced Allocation. *Approximation, Randomization, and Combinatorial Optimization (APPROX)*, pages 240–251.

Dean, Jeffrey and Ghemawat, Sanjay, 2008. MapReduce: Simplified Data Processing on Large Clusters. *Communications of the ACM*, 51(1):107–113.

DeFlumere, Ashley, 2014. *Optimal Partitioning for Parallel Matrix Computation on a Small Number of Abstract Heterogeneous Processors*. Thèse de doctorat, University College Dublin.

DeFlumere, Ashley and Lastovetsky, Alexey, 2014. Searching for the Optimal Data Partitioning Shape for Parallel Matrix-Matrix Multiplication on 3 Heterogeneous Processors. In *International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 17–28. IEEE.

DeFlumere, Ashley, Lastovetsky, Alexey and Becker, Brett, 2014. Optimal Data Partitioning Shape for Matrix Multiplication on Three Fully Connected Heterogeneous Processors. In *International European Conference on Parallel and Distributed Computing (Euro-Par) Workshop*, pages 201–214. Springer.

DeFlumere, Ashley, Lastovetsky, Alexey and Becker, Brett A, 2012. Partitioning for Parallel Matrix-Matrix Multiplication with Heterogeneous Processors: The Optimal Solution. In *International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 125–139. IEEE.

Demmel, James, Hoemmen, Mark, Mohiyuddin, Marghoob and Yelick, Katherine, 2008. Avoiding Communication in Sparse Matrix Computations. In *International Parallel and Distributed Processing Symposium (IPDPS)*, pages 1–12. IEEE.

Deveci, Mehmet, Rajamanickam, Sivasankaran, Devine, Karen D and Çatalyürek, Ümit V, 2016. Multi-Jagged: A Scalable Parallel Spatial Partitioning Algorithm. *Transactions on Parallel and Distributed Systems (TPDS)*, 27(3):803–817.

Dongarra, Jack and Sullivan, Francis, 2000. Guest Editors' Introduction: The Top 10 Algorithms. *Computing in Science & Engineering*, 2(1):22–23.

Dufossé, Fanny, Kaya, Kamer and Uçar, Bora, 2015. Two Approximation Algorithms for Bipartite Matching on Multicore Architectures. *Journal of Parallel and Distributed Computing*, 85:62 – 78.

Erdös, P and Rényi, A, 1964. On Random Matrices. *Studia Scientiarum Mathematicarum Hungarica*, 8:455–461.

Fiala, David, Mueller, Frank, Engelmann, Christian, Riesen, Rolf, Ferreira, Kurt and Brightwell, Ron, 2012. Detection and Correction of Silent Data Corruption for Large-Scale High-Performance Computing. In *International Conference on High Performance Computing (HiPC 2016)*, page 78. IEEE.

Ford, Lester R and Fulkerson, Delbert R, 1956. Maximal Flow through a Network. *Canadian Journal of Mathematics*, 8(3):399–404.

Ford, Lester Randolph and Fulkerson, Delbert Ray, 2015. *Flows in Networks*. Princeton University Press.

Fu, Huansong, Zhu, Yue and Yu, Weikuan, 2015. A Case Study of MapReduce Speculation for Failure Recovery. In *International Workshop on Data-Intensive Scalable Computing Systems (DISCS)*, pages 7:1–7:8. ACM.

Fügenschuh, Armin, Junosza-Szaniawski, Konstanty and Lonc, Zbigniew, 2014. Exact and Approximation Algorithms for a Soft Rectangle Packing Problem. *Optimization*, 63(11):1637–1663.

Garey, Michael R and Johnson, David S, 2002. *Computers and Intractability*, volume 29. W.H. Freeman.

Gautier, Thierry, Besseron, Xavier and Pigeon, Laurent, 2007. KAAPI: A Thread Scheduling Runtime System for Data Flow Computations on Cluster of Multi-processors. In *International Workshop on Parallel Symbolic Computation (PASCO)*, pages 15–23. ACM.

Geist, Al and Lucas, Robert, 2009. Major Computer Science Challenges at Exascale. *International Journal of High Performance Computing Applications (IJHPCA)*, 23(4):427–436.

Goel, Ashish, Kapralov, Michael and Khanna, Sanjeev, 2013. Perfect Matchings in $O(n \log n)$ Time in Regular Bipartite Graphs. *SIAM Journal on Computing (SICOMP)*, 42(3):1392–1404.

Gog, Ionel, Schwarzkopf, Malte, Gleave, Adam, Watson, Robert NM and Hand, Steven, 2016. Firmament: Fast, Centralized Cluster Scheduling at Scale. In *Symposium on Operating Systems Design and Implementation (OSDI)*, pages 99–115. USENIX.

Goto, Kazushige and Geijn, Robert A, 2008. Anatomy of High-Performance Matrix Multiplication. *Transactions on Mathematical Software (TOMS)*, 34(3):12:1–12:25.

Graham, Susan L, Patterson, Cynthia A, Snir, Marc *et al.*, 2005. *Getting Up to Speed: The Future of Supercomputing*. National Academies Press.

Greengard, Leslie and Rokhlin, Vladimir, 1987. A Fast Algorithm for Particle Simulations. *Journal of Computational Physics*, 73(2):325–348.

Guo, Zhenhua and Fox, Geoffrey, 2012. Improving MapReduce Performance in Heterogeneous Network Environments and Resource Utilization. In *International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, pages 714–716. IEEE.

Guo, Zhenhua, Fox, Geoffrey C. and Zhou, Mo, 2012. Investigation of Data Locality in MapReduce. In *International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, pages 419–426. IEEE/ACM.

Hall, Philip, 1935. On Representatives of Subsets. *Journal of the London Mathematical Society*, s1-10(1):26–30.

Hammoud, Mohammad and Sakr, Majd F., 2011. Locality-Aware Reduce Task Scheduling for MapReduce. In *International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 570–576. IEEE.

Heinecke, Alexander and Bader, Michael, 2008. Parallel Matrix Multiplication Based on Space-Filling Curves on Shared Memory Multicore Platforms. In *Workshop on Memory access on Future Processors (Maw)*, pages 385–392. ACM.

Hilbert, David, 1891. Ueber die stetige Abbildung einer Line auf ein Flächenstück. *Mathematische Annalen*, 38(3):459–460.

Hoemmen, Mark, 2010. *Communication-avoiding Krylov Subspace Methods*. Thèse de doctorat, University of California, Berkeley.

Hopcroft, John E. and Karp, Richard M., 1973. An $n^{5/2}$ Algorithm for Maximum Matchings in Bipartite Graphs. *SIAM Journal on Computing (SICOMP)*, 2(4):225–231.

Huang, Jianyu, Matthews, Devin A. and van de Geijn, Robert A., 2017. Strassen's Algorithm for Tensor Contraction. *Computing Research Repository (CoRR)*, abs/1704.03092.

Ibrahim, Shadi, Jin, Hai, Lu, Lu, He, Bingsheng, Antoniu, Gabriel and Wu, Song, 2012. Maestro: Replica-Aware Map Scheduling for MapReduce. In *International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, pages 435–442. IEEE/ACM.

Isard, Michael, Prabhakaran, Vijayan, Currey, Jon, Wieder, Udi, Talwar, Kunal and Goldberg, Andrew, 2009. Quincy: Fair Scheduling for Distributed Computing Clusters. In *Symposium on Operating Systems Principles (SOSP)*, pages 261–276. ACM.

Kalinov, Alexey and Lastovetsky, Alexey, 2001. Heterogeneous Distribution of Computations Solving Linear Algebra Problems on Networks of Heterogeneous Computers. *Journal of Parallel and Distributed Computing*, 61(4):520–535.

Kavulya, Soila, Tan, Jiaqi, Gandhi, Rajeev and Narasimhan, Priya, 2010. An Analysis of Traces from a Production MapReduce Cluster. In *International Conference on Cluster, Cloud and Grid Computing (CCGrid)*, pages 94–103. IEEE/ACM.

Kenyon, Richard, 1996. Tiling a Rectangle with the Fewest Squares. *Journal of Combinatorial Theory, Series A*, 76(2):272–291.

Langguth, Johannes, Manne, Fredrik and Sanders, Peter, 2010. Heuristic Initialization for Bipartite Matching Problems. *Journal of Experimental Algorithmics (JEA)*, 15:1.3:1.1–1.3:1.22.

Lima, Joao V. F., Gautier, Thierry, Maillard, Nicolas and Danjean, Vincent, 2012. Exploiting Concurrent GPU Operations for Efficient Work Stealing on Multi-GPUs. In *International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, pages 75–82.

Litke, Antonios, Skoutas, Dimitrios, Tserpes, Konstantinos and Varvarigou, Theodora, 2007. Efficient Task Replication and Management for Adaptive Fault Tolerance in Mobile Grid Environments. *Future Generation Computer Systems*, 23(2):163–178.

Lyons, Robert E and Vanderkulk, Wouter, 1962. The Use of Triple-Modular Redundancy to Improve Computer Reliability. *IBM Journal of Research and Development*, 6(2):200–209.

MANNE, Fredrik and SØREVIK, Tor, 1996. Partitioning an Array onto a Mesh of Processors. *Applied Parallel Computing Industrial Computation and Optimization*, pages 467–477.

MITZENMACHER, Michael, 2001. The Power of Two Choices in Randomized Load Balancing. *Transactions on Parallel and Distributed Systems (TPDS)*, 12(10):1094–1104.

NAGAMOCHI, Hiroshi and ABE, Yuusuke, 2007. An Approximation Algorithm for Dissecting a Rectangle into Rectangles with Specified Areas. *Discrete Applied Mathematics*, 155(4):523 – 537.

NAPOLI, Edoardo Di, FABREGAT-TRAVER, Diego, QUINTANA-ORTÃ, Gregorio and BIENTINESI, Paolo, 2014. Towards an Efficient Use of the BLAS Library for Multilinear Tensor Contractions. *Applied Mathematics and Computation*, 235:454 – 468.

NELSON, Thomas, RIVERA, Axel, BALAPRAKASH, Prasanna, HALL, Mary, HOVLAND, Paul D., JESSUP, Elizabeth and NORRIS, Boyanna, 2015. Generating Efficient Tensor Contractions for GPUs. In *International Conference on Parallel Processing (ICPP)*, pages 969–978.

OUELHADJ, Djamila and PETROVIC, Sanja, 2009. A Survey of Dynamic Scheduling in Manufacturing Systems. *Journal of Scheduling*, 12(4):417–431.

PEISE, Elmar, FABREGAT-TRAVER, Diego and BIENTINESI, Paolo, 2014. On the Performance Prediction of BLAS-based Tensor Contractions. In *International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS)*, pages 193–212. Springer.

PERES, Yuval, TALWAR, Kunal and WIEDER, Udi, 2010. The (1+ β)-Choice Process and Weighted Balls-into-Bins. In *Symposium on Discrete Algorithms (SODA)*, pages 1613–1619. SIAM.

PILKINGTON, John R., BADEN, Scott B. and BADEN, Scott B., 1994. Partitioning with Spacefilling Curves. Technical report, University of California, San Diego.

PlaFRIM2, 2009. Plateforme Fédérative pour la Recherche en Informatique et Mathématiques.
URL `https://www.plafrim.fr/fr/accueil/`

QIU, Zhen, STEIN, Cliff and ZHONG, Yuan, 2015. Minimizing the Total Weighted Completion Time of Coflows in Datacenter Networks. In *Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 294–303. ACM.

Raab, Martin and Steger, Angelika, 1998. Balls into Bins: A Simple and Tight Analysis. In *Randomization and Approximation Techniques in Computer Science (RANDOM)*, pages 159–170. Springer.

Ranganathan, Kavitha and Foster, Ian, 2001. Identifying dynamic replication strategies for a high-performance data grid. *Grid Computing—GRID*, pages 75–86.

Reed, Daniel A and Dongarra, Jack, 2015. Exascale Computing and Big Data. *Communications of the ACM*, 58(7):56–68.

Richa, Andrea W, Mitzenmacher, M and Sitaraman, R, 2001. The Power of Two Random Choices: A Survey of Techniques and Results. *Combinatorial Optimization*, 9:255–304.

Sanders, Peter, 2004. Algorithms for Scalable Storage Servers. In *International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM)*, pages 82–101. Springer.

Sanders, Peter, Egner, Sebastian and Korst, Jan, 2003. Fast Concurrent Access to Parallel Disks. *Algorithmica*, 35(1):21–55.

Semar Shahul, Ahmed Zaki and Sinnen, Oliver, 2010. Scheduling Task Graphs Optimally with A*. *The Journal of Supercomputing*, 51(3):310–332.

Sen, Tapan and Gupta, Sushil K, 1984. A State-of-Art Survey of Static Scheduling Research Involving Due Dates. *Omega*, 12(1):63–76.

Shalf, John, Dosanjh, Sudip and Morrison, John, 2011. Exascale Computing Technology Challenges. *High Performance Computing for Computational Science (VECPAR)*, pages 1–25.

Shams, Ramtin and Sadeghi, Parastoo, 2011. On Optimization of Finite-difference Time-domain (FDTD) Computation on Heterogeneous CPU and GPU clusters. *Journal of Parallel and Distributed Computing*, 71(4):584–593.

Shi, Yang, Niranjan, Uma Naresh., Anandkumar, Animashree and Cecka, Cris, 2016. Tensor Contractions with Extended BLAS Kernels on CPU and GPU. In *International Conference on High Performance Computing (HiPC 2016)*, pages 193–202.

Skilling, John, Erickson, Gary J and Zhai, Yuxiang, 2004. Programming the Hilbert Curve. In *AIP Conference Proceedings*, volume 707, pages 381–387. AIP.

Solomonik, Edgar and Demmel, James, 2011. Communication-optimal Parallel 2.5 D Matrix Multiplication and LU factorization Algorithms. In *International European Conference on Parallel and Distributed Computing (Euro-Par)*, pages 90–109. Springer.

Spark, Apache, 2016. Apache Spark: Lightning-Fast Cluster Computing. URL `http://spark.apache.org`

Stockinger, Heinz, Samar, Asad, Allcock, Bill, Foster, Ian, Holtman, Koen and Tierney, Brian, 2001. File and Object Replication in Data Grids. In *International Symposium on High Performance Distributed Computing (HPDC)*, pages 76–86. IEEE.

Strassen, Volker, 1969. Gaussian Elimination is Not Optimal. *Numerische mathematik*, 13(4):354–356.

Tan, Jian, Meng, Shicong, Meng, Xiaoqiao and Zhang, Li, 2013. Improving ReduceTask data locality for sequential MapReduce jobs. In *International Conference on Computer Communications (INFOCOM)*, pages 1627–1635. IEEE.

Topcuoglu, Haluk, Hariri, Salim and Wu, Min-you, 2002. Performance-Effective and Low-Complexity Task Scheduling for Heterogeneous Computing. *Transactions on Parallel and Distributed Systems (TPDS)*, 13(3):260–274.

Walkup, David W, 1980. Matchings in Random Regular Bipartite Digraphs. *Discrete Mathematics*, 31(1):59–64.

Walters, Mark, 2009. Rectangles as Sums of Squares. *Discrete Mathematics*, 309(9):2913 – 2921.

Wang, Fuxing, Ramamritham, Krithi and Stankovic, John A, 1995. Determining Redundancy Levels for Fault Tolerant Real-Time Systems. *Transactions on Computers*, 44(2):292–301.

Wang, Weina, Zhu, Kai, Ying, Lei, Tan, Jian and Zhang, Li, 2013. Map Task Scheduling in MapReduce with Data Locality: Throughput and Heavy-Traffic Optimality. In *International Conference on Computer Communications (INFOCOM)*, pages 1609–1617. IEEE.

West, Douglas Brent *et al.*, 2001. *Introduction to Graph Theory*, volume 2. Prentice Hall.

White, Tom, 2012. *Hadoop: The Definitive Guide*. O'Reilly Media.

Xie, Qiaomin and Lu, Yi, 2012. Degree-Guided Map-Reduce Task Assignment with Data Locality Constraint. In *International Symposium on Information Theory (ISIT)*, pages 985–989. IEEE.

Yekkehkhany, Ali, 2017. Near Data Scheduling for Data Centers with Multi Levels of Data Locality. *Computing Research Repository (CoRR)*, abs/1702.07802.

Zaharia, Matei, Borthakur, Dhruba, Sen Sarma, Joydeep, Elmeleegy, Khaled, Shenker, Scott and Stoica, Ion, 2010. Delay Scheduling: A Simple Technique for Achieving Locality and Fairness in Cluster Scheduling. In *European Conference on Computer Systems (EuroSys)*, pages 265–278. ACM.

Zaharia, Matei, Konwinski, Andy, Joseph, Anthony D, Katz, Randy H and Stoica, Ion, 2008. Improving MapReduce Performance in Heterogeneous Environments. In *Symposium on Operating Systems Design and Implementation (OSDI)*, pages 29–42.

Zarei Zefreh, Ebrahim, Lotfi, Shahriar, Mohammad Khanli, Leyli and Karimpour, Jaber, 2016. 3-D data partitioning for 3-level perfectly nested loops on heterogeneous distributed systems. *Concurrency and Computation: Practice and Experience*.