Equilibrage et régulation de charge dans les machines parallèles à mémoire distribuée

Mihaela Juganaru

ECOLE NATIONALE SUPERIEURE
des MINES de SAINT-ETIENNE

UNIVERSITE Jean MONNET
de SAINT-ETIENNE

N° d'ordre : 165 ID

THESE

présentée par

Mihaela JUGANARU

pour obtenir le titre de

DOCTEUR

spécialité : **Informatique**

Equilibrage et régulation de charge dans les machines parallèles à mémoire distribuée

le19 janvier 1999

Jury :
Hervé GUYENNET
Traian MUNTEAN
Jean AZEMA
Laurent CARRARO
Bertil FOLLIOT
Frank MONTHEILLET
Bernard PEROCHE
Jean-Louis ROCH
Ibrahima SAKHO

ECOLE NATIONALE SUPERIEURE
des MINES de SAINT-ETIENNE

UNIVERSITE Jean MONNET
de SAINT-ETIENNE

N° d'ordre : 165 ID

THESE

présentée par

Mihaela JUGANARU

pour obtenir le titre de

DOCTEUR

spécialité : **Informatique**

Equilibrage et régulation de charge dans les machines parallèles à mémoire distribuée

le19 janvier 1999

Jury :
Hervé GUYENNET
Traian MUNTEAN
Jean AZEMA
Laurent CARRARO
Bertil FOLLIOT
Frank MONTHEILLET
Bernard PEROCHE
Jean-Louis ROCH
Ibrahima SAKHO

A tous mes professeurs

de mathématiques et d'informatique

Je tiens à exprimer ici ma profonde gratitude envers les membres du jury :

- à Traian Muntean qui a été mon initiateur dans le chapitre merveilleux du parallélisme lors de mon arrivée en DEA à Grenoble. Il a été, naturellement, un des rapporteurs de cette thèse et ses conseils ont été très bénéfiques pour la forme finale du manuscrit.

- à Hervé Guyennet qui a accepté si gentilment d'être rapporteur et qui m'a fait des remarques justes et constructives et qui m'a suggéré la juste valeur du noyau de cette thèse.

- à Bertil Folliot, président du jury de thèse, qui connait bien mon activité de recherche.

- Jean-Louis Roch, mon professeur depuis le DEA, qui avait accepté depuis bien longtemps d'être membre du jury.

- à Jean Azéma qui a bien voulu être dans le jury et qui m'a donné une bonne idée de recherche.

- à Laurent Carrarro, qui a su m'écouter, m'expliquer, me guider, me soutenir dans mes démarches probabilistes; sans lui cette thèse n'aurait pas vu le jour.

- à Frank Montheillet, qui a m'a présenté une application, une vraie, à paralléliser et qui m'a fait connaitre un nouveau domaine de recheche. Un chapitre de cette thèse porte le fruit de cette collaboration.

- à Bernard Perroche, directeur de thèse, qui a tout fait, humainement et diplomatiquement, pour que la thèse, y compris la soutenance, se passe dans les meilleures conditions.

- à Ibrahima Sakho qui m'a donné la chance de faire cette thèse.

Cette thése a vu le jour au sein du Département d'Informatique de l'Ecole des Mines de Saint Etienne et tous les membres de ce département trouvent ici mes plus vifs remerciements :

- Annie Corbel-Bourgeat qui a été mon exemple;
- Hélène Sayet qui m'a toujours trouvé efficacement tous les nombreux livres et articles que j'ai pu lui demander;
- Marie-Line Barnéoud qui a été mon amie et mon ange gardien;
- Jean-Michel Moreau, Marc Rolens et Michel Beigbeder pour leur bons conseils sur LaTeX, Unix, `vi` ...
- toutes et tous mes collègues de bureau et, surtout, du coin café : Bich-Lien, Dominique, Graciela, Grégory, Gilles, Helymar, Jacques, Jean-Michel, Marc, Marion, Nicolas;
- Roland Jégou pour les nombreuses discussions sur les graphes;
- tous les autres pour leur gentillesse et leur accueil.

Je remercie également Claire Maurice du Département SMS pour notre fructueuse collaboration dans le domaine des matériaux et pour m'avoir fourni les logiciels dont j'ai eu besoin pour mes simulations mécaniques.

Je remercie toutes les personnes, elles sauront s'y reconnaitre, qui ont su m'appuyer moralement, avec la bonne parole, dans des noirs moments. Je tiens à remercier aussi toutes les personnes qui ont pris le temps de lire mon manuscrit et qui m'ont ainsi aidée à l'améliorer.

J'exprime aussi ma gratitude à Jean-Lucien Rascle, Chantal Ledoux et Dominique Testa pour m'avoir permis de finir ma thèse dans de bonnes conditions.

Merci à ma famille pour toute sa compréhension et tout son amour.

Table des matières

Chapitre 1

Introduction

1.1 Problématique du calcul parallèle

Même si des processeurs avec toujours plus de puissance de calcul sont conçus et développés, le besoin de calcul de certaines applications, telles que la prévision météorologique, le calcul mécanique ou le traitement d'images, augmente encore plus vite. La solution la plus intéressante à ce problème de puissance de calcul est le **parallélisme**[1].

Communément une **architecture parallèle** contient d'une dizaine à des milliers des processeurs, on parle alors de parallélisme massif [GRS91].

Les classifications possibles des architectures parallèles sont multiples[2] , en fonction de la nature du **réseau d'interconnexion**[3] entre processeurs ou de la présence / absence d'une mémoire commune. Nous nous intéresserons uniquement aux architectures parallèles à **réseau fixe d'interconnexion** des processeurs et à **mémoire distribuée.**

Concevoir des applications pour de telles machines impose soit un effort de la part du programmeur suivant une certaine méthodologie, soit des outils de parallélisation automatique. De plus, il faut pouvoir gérer les communications, la synchronisation, l'ordonnancement, l'allocation des ressources [BF96] : entrées /sorties, espace mémoire, charge de travail, fichiers [BCS92], [Had92].

Dans cette thèse, nous nous intéresserons uniquement à l'**allocation des charges.** On entend par **charges de travail** les composantes d'une application : des processus, des données ou des objets. Un cas particulier est celui des **charges régulières** où chaque composante peut s'exécuter sur n'importe quel processeur et exige toujours un même volume de traitement. Cette hypothèse qui semble être assez restrictive couvre en fait un vaste ensemble d'applications très exigeantes en puissance de calcul. C'est, par exemple, le cas pour les simulations moléculaires dans la physique des particules [BF90], [BBK91], pour le traitement d'images [Mig92], [GO93], pour la mécanique des fluides [Hei94], ou bien pour l'application de simulation mécanique implantée dans le cadre de cette thèse.

1. Voir [AG94] et [Wil95] comme introductions aux problèmes du parallélisme et à la programmation parallèle.
2. Le point de départ de ces classifications est le travail de Flynn [Fly72]
3. Voir [Lei93] et [dR95] comme monographies sur les réseaux d'interconnexion.

1.2 Place de l'allocation de charge dans les calculateurs parallèles

L'**allocation de charge** est une partie essentielle et incontournable lors de la mise en œuvre d'une application parallèle, car il s'agit de décider quelle composante de l'application sera exécutée par quel processeur. L'allocation de charge est primordiale pour tirer le meilleur parti de l'utilisation de la machine parallèle car elle vise à minimiser le temps total d'exécution de l'application ou à optimiser l'utilisation de ressources.

Le problème de l'allocation se pose et se résout de la même façon aussi bien pour le traitement (tâche ou processus), que pour les données qui composent l'application; nous appellerons **unité de charge** toute composante d'une application.

D'autre part, le problème quel processeur exécutera telle tâche est un problème difficile à résoudre.

Une application parallèle peut avoir une structure fixe et connue lors de l'étape de conception ou lors de la compilation et dans ce cas l'**allocation statique** est la solution la plus naturelle et la plus simple à mettre en œuvre.

Pour la classe d'applications dynamiques quand des nouvelles unités de charge sont créées pendant l'exécution ou dans les cas pour lesquels la dynamique de l'application ou de la machine n'est pas prévisible, l'**allocation dynamique** est l'unique solution. La difficulté de résolution du problème d'allocation dynamique est plus grande, car un plus du problème d'allocation en soi, on doit faire face à un problème de collecte d'information sur l'état de charge des processeurs, un processeur n'ayant qu'une vision sur lui-même et donc la mise en place, dans le cadre de l'allocation dynamique, d'une stratégie de collecte d'informations s'impose. D'un autre coté, en plus de la distribution de l'information on doit tenir compte de la distribution du calcul de l'allocation, car des solutions centralsés, où un unique processeur calcule la nouvelle allocation ne sont pas envisageables.

Afin d'accomplir cette tâche d'allocation il existe des méthodes génériques qui décrivent comment il faut procéder. Il y a aussi quelques outils complets qui assurent l'allocation de charge soit dans la cas statique [PR96] soit dans le cas dynamique comme Utopia [ZZWD93] ou GATOS [FS93] soit à la volée comme Athapascan [BGR96]. L'idée est toujours de faire en sorte que l'allocation de charge soit ou devienne la plus simple à la plus transparente possible à l'utilisateur avec des performances de plus en plus intéressantes.

Il est donc nécessaire de concevoir des outils génériques qui s'appuient sur des méthodes l'allocation (statique ou dynamique) de plus en plus performantes soit en terme de temps de calcul d'une allocation, soit en terme de la qualité de l'allocation obtenue. Les méthodes conçues se doivent être correctes, efficaces et robustes.

1.3 Contribution de la thèse

Dans ce travail, nous proposons d'abord une stratégie dynamique de régulation des charges régulières. Cette stratégie est, d'une part, correcte et exacte, aboutissant dans un temps insignifiant à l'état d'équilibre parfait, d'autre part, elle est extensible à toute topologie du réseau d'interconnexion. Utilisant cette stratégie, nous avons implanté un noyau générique de régulation de charges pour des machines parallèles.

Cette stratégie est analysée du point de vue de la complexité; une analyse probabiliste de sa complexité en temps est menée. Cette approche originale nous a conduit à la loi de probabilité qui est la borne supérieure du temps de régulation. Nous avons également estimé les paramètres de cette

loi, qui est une loi de Gumbel. Sur la base des résultats obtenus, un algorithme complet et stable d'allocation dynamique est déduit, car la loi de probabilité obtenue permet d'obtenir des bornes très fines pour l'initiation et le déclenchement de la stratégie de régulation.

Afin de vérifier en pratique le comportement de l'algorithme de régulation proposé nous avons développé entièrement sur une machine parallèle à base de transputers une application de déformation - recristallisation dynamique continue d'agrégats polycristallins. Cette application a exigé un choix judicieux des structures de données parallèles et de la mise en œuvre des processus de travail et des communications. Les résultats obtenus montrent la correction et l'efficacité réelle de l'algorithme de régulation proposé.

1.4 Organisation de la thèse

La suite de la thèse est organisée en trois grands chapitres. Le chapitre qui suit présente une synthèse du problème d'allocation de charge dans les architectures parallèles, l'accent est mis sur l'allocation dynamique et notamment sur les stratégies d'allocation de charges régulières, afin de montrer les avantages et les limitations des méthodes qui existent dans la littérature et de justifier l'approche de régulation proposée.

Le chapitre 3 est dédié à la partie théorique de la thèse, à savoir la description et l'analyse d'un algorithme complet, exact, stable, efficace et extensible d'allocation dynamique de charges régulières. Nous commençons par décrire entièrement la composante stratégie qui est basée sur un calcul de préfixe généralisé. Cette stratégie est proposée d'abord pour des architectures homogènes (même puissance de calcul pour chaque processeur) et ensuite pour des architectures hétérogènes (puissances différentes des processeurs). La stratégie est prouvée correcte: elle est capable d'assurer en temps fini la même charge de travail à chaque processeur. Son analyse de complexité comporte deux volets: une analyse algorithmique classique et une probabiliste qui vise à établir que la loi de probabilité qui modélise le temps maximal d'exécution de la régulation est une loi de Gumbel. Sur la base de ce résultat, des politiques d'initiation et de décision sont inférées et un algorithme complet d'allocation dynamique des charges régulières est décrit.

Le chapitre 4 concerne la validation de l'algorithme d'allocation dynamique dans le cadre d'une application dynamique réelle. Cette application concerne la simulation, sur une machine parallèle d'un modèle de recristallisation dynamique continue d'un agrégat polycristallin soumis à une déformation à chaud. Le modèle prend en compte aussi bien les aspects géométriques, que les aspects mécanique et métallurgie physique du matériau. L'implantation concrète de la déformation (compression) et de la recristallisation : structures de données, structures de processus, communications, placement, régulation, migration, accélération de temps obtenues est décrite en détails. Le chapitre est clos en montrant la signification de point de vue mécanique et informatique des résultats de simulation obtenus. L'algorithme proposé s'avère correct et efficace.

La dernière partie tire les conclusions de ce travail, montre les problèmes qui restent ouverts et indique quelques pistes pour les résoudre en partie.

Chapitre 2

Allocation de charge dans les machines parallèles à mémoire distribuée

Le but de ce chapitre est de modéliser le problème d'allocation de charge en général et de trouver une classification de ses solutions, en prêtant plus d'attention à l'allocation dynamique, plus particulièrement à l'allocation des charges régulières qui est au cœur de cette thèse.

Nous commencerons cette étude bibliographique par une modélisation du problème d'allocation de charges, en mettant l'accent sur la complexité du problème. Nous continuons avec une étude bibliographique du problème d'allocation statique.

La section suivante 2.3 traite de l'allocation dynamique et notamment des solutions distribuées. Les stratégies utilisées pour permettre aux processeurs de s'informer, d'initier et de décider du processus d'allocation, ainsi que les mécanismes de la mise en œuvre sont discutés.

La dernière section 2.4 est consacrée aux stratégies d'allocation pour le cas particulier des charges régulières. Les avantages et les limitations des méthodes qui existent dans la littérature sont dégagés.

2.1 Modélisation du problème d'allocation. Complexité

2.1.1 Modélisation du réseau d'interconnexion

En tant qu'ensemble de processeurs interconnectés par des liens de communication, une machine parallèle MIMD sans mémoire commune est, en général, modélisée par le graphe du réseau d'interconnexion de ses processeurs. Soit $G = (V, E)$ le graphe support d'interconnexion de la machine ou de la partie de la machine parallèle réservée à l'utilisateur [1]. V est l'ensemble des nœuds, chaque nœud étant associé à un processeur, et E est l'ensemble des arêtes, chaque arête est associée à un lien physique de communication entre processeurs. On notera $|V| = n$.

Dans une architecture distribuée sans mémoire commune, les processeurs communiquent à l'aide de messages. Les communications sont soit synchrones, soit asynchrones. Dans ce dernier cas il est

1. Une bonne partie des machines parallèles commercialisées sont multi-utilisateurs : VOLVOX d'Archipel, SP1 et SP2 d'IBM.

toujours possible de transformer les algorithmes asynchrones en algorithmes synchrones [Lav95], car l'analyse d'un algorithme synchrone est plus facile.

Il y a deux modèles différents [Cyb89, GLM+96] quant à la façon de réaliser les communications : port unique et multi-port. Dans le premier modèle, on suppose qu'un processeur peut faire une unique communication à la fois, le temps total de communication d'un processeur étant alors la somme des temps de communication sur tous ses liens comme dans [Nic92], tandis que dans le modèle multi-port tout processeur est capable de faire des communications simultanées sur tous ses liens physiques. Cette différence entre les modèles va engendrer des critères différents d'allocation statique (sous-section 2.1.5) et des complexités différentes en temps d'exécution pour les algorithmes d'allocation dynamique, plus particulièrement pour la régulation des charges régulières (voir la section 2.4).

2.1.2 Modélisation d'une application parallèle

Une application est, en général, perçue comme un ensemble T de tâches $T = \{t_1, t_2, \cdots, t_m\}$ qui sont reliées par des relations de communication et/ou de précédence. Une relation de communication entre deux tâches t_i et t_j signifie que la tâche t_i communique (dans le cas de la mémoire distribuée, envoie des messages) avec la tâche t_j ou l'inverse. Une relation de précédence entre t_i et t_j signifie que la tâche t_j ne peut commencer qu'après que t_i soit finie.

L'ensemble des tâches avec leurs relations se modélise par un graphe non-orienté (respectivement orienté) $G_T = (T, E_T)$ dit **graphe de communication** (respectivement **graphe de précédence**). Peu de restrictions sont faites, en général, sur les propriétés de ces graphes : pour le graphe de précédence on impose qu'il n'y ait pas de circuit; parfois pour le graphe de communication on impose lors de l'allocation dynamique qu'il soit un arbre [Tal93], [Rom97].

La **charge de travail** est constituée de toutes les tâches et les communications à accomplir.

A chaque tâche t_i on peut associer un certain **volume de calcul** $calcul(t_i)$ qui correspond au temps CPU nécessaire à l'exécution de cette tâche. A chaque couple (t_i, t_j) on peut associer aussi un certain **volume de communication** $comm(t_i, t_j)$ qui correspond à la taille totale des messages que les deux tâches échangent.

2.1.3 Modélisation d'une allocation

Une **allocation** établit pour chaque tâche le processeur qui sera chargé de l'exécuter. On peut donc modéliser une allocation par une application $A : T \rightarrow V$ qui à toute tâche t associe le processeur p qui l'exécutera :

$$A(t) = p$$

Le nombre d'allocations possibles d'un ensemble de n tâches sur m processeurs est de n^m, nombre trop important pour pouvoir lister et analyser toutes les allocations possibles. Par exemple, quand on veut placer si ce n'est que 7 tâches sur 4 processeurs, le nombre d'allocations possibles est de plus de 10000.

Quant une relation de communication existe entre deux tâches t et t', deux cas peuvent apparaître :

- les tâches t et t' sont allouées sur un même processeur $A(t) = A(t')$, ce qui fait que cette communication est une opération interne au processeur d'allocation de t et t';

- les tâches t et t' sont allouées sur des processeurs différents: $A(t) = p$, $A(t') = p'$ et $p \neq p'$, dans ce cas une communication physique entre les processeurs p et p' doit se faire par le biais d'un noyau de routage. Si on veut éviter l'utilisation d'un tel noyau on peut imposer de toujours placer les tâches communicantes sur un même processeur ou sur des processeurs physiquement voisins[2].

Un **ordonnancement** parallèle est un cas plus particulier de l'allocation, car pour chaque tâche il faut indiquer la date de commencement. On modélise donc un ordonnancement par une application $O : T \to V \times [0, \infty[$ où :

$$O(t) = (p, \tau)$$

indique que la tâche t sera exécutée par le processeur p et que τ est le moment où cette exécution commence ou peut commencer. Une contrainte naturelle de l'ordonnancement est que celui-ci doit être compatible avec les relations de précedence qui existent entre les tâches, c'est à dire que si t_1 précède t_2 et $O(t_1) = (p_1, \tau_1)$, $O(t_2) = (p_2, \tau_2)$, alors :

$$\tau_1 + calcul(t_1) \leq \tau_2$$

Nous traiterons uniquement le problème de la recherche d'une allocation pour une application donnée. Bien que de même nature, le problème d'ordonnancement est un problème à part entière qui ne sera pas traité ici; pour plus amples informations sur ce sujet voir [ERAL95].

2.1.4 Estimation du temps d'exécution d'une application parallèle

Il est évident que le temps d'exécution d'une application parallèle dépend de l'allocation des tâches qui la composent et que l'idéal serait de trouver une allocation qui minimise ce temps.

Pour une allocation donnée il est très difficile de calculer (estimer) a priori le temps global que l'application nécessitera. Il s'agit, d'une part, de connaître de façon quantifiée la structure de l'application et, d'autre part, de pouvoir modéliser convenablement à l'aide de ces valeurs le temps global d'exécution.

La connaissance de la structure de l'application se traduit par une connaissance de ses composantes (code et/ou données) et des relations qui existent entre les composantes : communication, dépendance, antériorité. Il est difficile de connaître exactement le temps d'exécution de chacune des composantes (voir [CS93]), on peut toutefois comme dans Athapascan ([RVV94]) avoir l'ordre de grandeur de ces temps. Quant aux relations de communication (ou d'antériorité) entre composantes, elles induisent un certain temps pour être accomplies, mais il est encore difficile d'estimer ce temps (voir [TP94] ou [SRG94] pour l'estimation des communications), car il dépend de l'allocation choisie, du noyau de routage, de l'ordre d'exécution des composantes sur un même processeur. Pour les applications dynamiques, il est encore plus compliqué de pouvoir estimer pendant l'exécution de l'application le coût de ses composantes.

Dans la suite nous donnerons une modélisation du temps global d'exécution d'une application en fonction d'une allocation A en supposant que la structure du réseau, la structure de l'application et le coût de chaque composante ($calcul(t)$ et $comm(t, t')$) sont connus. Nous avertissons dès le départ que c'est très rare que les valeurs $calcul(t)$ et $comm(t, t')$ soient connues, qu'on se contente très souvent

2. Cette restriction empêche ou même interdit l'équilibrage de charge.

des estimations ou, au mieux, de leurs bornes supérieures. Ces éléments connus ou supposés connus nous permettent de raisonner sur le temps global d'une application parallèle et de se rendre compte de la complexité du problème d'allocation.

En raison de la nature parallèle de l'application, le temps total d'exécution peut être exprimer comme dans [SE87a] et [CT93] par:

$$T_M(A) = \max_{p \text{ processeur}} T(p, A)$$

où $T(p, A)$ est le temps du processeur p et s'exprime comme la somme de son temps de calcul $T_{calcul}(p, A)$, son temps de communication $T_{comm}(p, A)$ et son temps d'inactivité $T_{sync}(p, A)$ qui correspond aux délais de synchronisation :

$$T(p, A) = T_{calcul}(p, A) + T_{comm}(p, A) + T_{sync}(p, A)$$

Le temps de calcul d'un processeur p $T_{calcul}(p, A)$ correspond à la somme des temps de calcul de chaque tâche allouée à ce processeur :

$$T_{calcul}(p, A) = \sum_{t, A(t)=p} calcul(t)$$

On peut introduire un coefficient de pondération α_p :

$$T_{calcul}(p, A) = \alpha_p * \sum_{t, A(t)=p} calcul(t)$$

afin de pouvoir prendre en compte le cas des architectures hétérogènes comme dans [SB78].

Le temps de communication $T_{comm}(p, A)$ correspond aussi à la somme des temps de communication des tâches allouées au processeur p:

$$T_{comm}(p, A) = \sum_{t, A(t)=p} t_{comm}(t)$$

où $t_{comm}(t)$ est le temps pris par la tâche t pour effectuer toutes ses communications avec les tâches qui ne sont pas situées sur le même processeur :

$$t_{comm}(t) = \sum_{t', A(t') \neq A(t)} t_{comm}(t, t')$$

Le temps $t_{comm}(t, t')$ dépend fortement de la fonction de routage de la machine parallèle, toutefois on peut l'estimer comme le produit entre le volume de l'information transmise et la distance physique entre les processeurs qui disposent des tâches t et t' :

$$t_{comm}(t, t') = comm(t, t') * dist(A(t), A(t'))$$

Donc

$$T_{comm}(p, A) = \sum_{t, A(t)=p} \sum_{t', A(t') \neq p} comm(t, t') * dist(p, A(t'))$$

Le temps de synchronisation (d'inactivité), même s'il devient important, est ignoré dans le calcul du temps total, car il est difficile à estimer étant dépendant surtout du séquencement des exécutions de tâches sur le processeur [KR93]. Dans [MR91], Marinescu et Rice analysent la perte de performance, due à la synchronisation des communications et de calculs.

Le temps du processeur p est alors :

$$T(p,A) = \sum_{t,A(t)=p} (calcul(t) + \sum_{t',A(t')\neq p} comm(t,t') * dist(p,A(t')))$$

Le temps total d'exécution d'une application sur une machine donnée et en utilisant une allocation A est :

$$T_M(A) = \max_{p \text{ processeur}} \sum_{t,A(t)=p} (calcul(t) + \sum_{t',A(t')\neq p} comm(t,t') * dist(p,A(t')))$$

2.1.5 Modélisation des critères et des contraintes d'allocation

A toute allocation A on peut associer une fonction coût $f(A)$. Un état de l'art des fonctions coût utilisées dans la littérature se trouve dans la synthèse faite par Muntean et Talbi [MT91a]. Nous donnerons quelques expressions de cette fonction ainsi que quelques contraintes qu'on peut imposer lors de la recherche d'une «bonne» allocation.

Une fonction coût naturelle [Lo85, KS93, KNN87, WM93, KB88] est le temps total d'exécution de l'application :

$$f(A) = T_M(A) = \max_{p \text{ processeur}} \sum_{t,A(t)=p} (calcul(t) + \sum_{t',A(t')\neq p} comm(t,t') * dist(p,A(t')))$$

Une fonction *max* n'est pas toujours simple à manipuler et elle a des propriétés de régularité (dérivabilité) moins bonnes qu'une fonction somme. On préfère parfois prendre [LLK92] la somme des temps d'exécution des processeurs :

$$f(A) = \sum_{p \text{ processeur}} T(p,A) = \sum_{t \text{ tâche}} calcul(t) + \sum_{\substack{t,t' \text{ tâches} \\ A(t)\neq A(t')}} comm(t,t') * dist(A(t),A(t'))$$

Dans cette expression le premier terme correspond à la somme des temps de calcul CPU de tous les processeurs et le deuxième à la somme globale des communications. Pour privilégier l'un ou l'autre de ces termes, on peut prendre une somme pondérée :

$$f(A) = K_{calcul} * \sum_{t \text{ tâche}} calcul(t) + K_{comm} * \sum_{\substack{t,t' \text{ tâches} \\ A(t)\neq A(t')}} comm(t,t') * dist(A(t),A(t'))$$

où K_{calcul}, K_{comm} sont des coefficients de pondération. Certains auteurs privilégient surtout les communications (donc $K_{calcul} = 0$), comme dans [CLK92] ou [BNG92] en ajoutant des bornes aux charges totales des processeurs.

Afin d'optimiser l'utilisation de la machine parallèle et/ou de tenir compte de ses limitations physiques ou des limitations de l'application, on peut imposer différentes contraintes:

1. Contraintes de non-oisiveté: suite à l'allocation, tous les processeurs reçoivent de la charge, ce qui signifie que l'application $A : T \rightarrow V$ est surjective.

2. Contraintes de charge: le nombre de tâches allouées est borné (inférieurement [KM87] ou supérieurement [JR92]).

3. Contraintes de taille mémoire: la somme des mémoires requises par chaque tâche allouée ne doit pas dépasser le montant de la mémoire libre du processeur[3].

4. Contraintes de routage: on impose que les tâches communicantes ne soient pas placées sur des processeurs trop éloignés. Certains, comme Bokhari [Bok81b, Bok79, Bok88], Ercal et Sadayappan [ERS90, SE87b, SE87a], Lamour [Lam91], imposent que le routage ne soit pas utilisé, i.e. que les tâches communicantes soient placées sur un même processeur ou sur des processeurs physiquement voisins.

5. Contraintes temporelles: pour les applications temps-réel, on impose que l'exécution de toutes ou certaines tâches n' excède pas une durée précise ou finisse avant des instants précis [Bac95].

2.1.6 Problème de recherche de l'allocation optimale

Dans le but d'optimiser les performances de l'application parallèle on se pose naturellement le problème de trouver une allocation A^* qui minimise la fonction coût choisie:

$$f(A^*) = \min_{A \text{ allocation}} f(A) \tag{2.1}$$

et, si besoin est, respecte les contraintes imposées.

Pour la fonction coût de type max l'équation (2.1) signifie qu'on cherche à minimiser le temps maximum d'exécution de tous les processeurs; tandis que pour l'expression somme l'équation signifie qu'on cherche à minimiser le temps moyen d'exécution des processeurs.

Les expressions des fonctions coût considérées se calculent en temps polynômiaux, une allocation aléatoire se construit toujours en temps polynômial et les contraintes se vérifient aussi en temps polynômial. On peut donc trouver un algorithme non-déterministe polynômial qui construit une allocation et vérifie si son coût est plus petit qu'une valeur C. Donc le problème de recherche d'une allocation optimale par rapport à une fonction coût est dans la classe NP.

Nous verrons dans la section suivante 2.2 des instances particulières de ce problème (topologies particulières du graphe des processeurs et/ou du graphe des tâches) qui admettent des solutions en temps polynômial.

Dans sa généralité, ce problème de recherche combinatoire est NP-complet [GJ79]. Pour la fonction coût de forme max, il est réductible au problème de recherche d'un chemin de poids maximum dans

3. Nous avons été confrontés à une contrainte de ce type lors de la mise en œuvre de l'application de simulation des phénomènes mécaniques (voir le chapitre 4): le nombre d'unités de charge ne pouvait pas dépasser 180 par processeur, car la mémoire d'un processeur de notre plateforme était limitée à 4Mbytes.

un graphe; pour la fonction coût de forme somme le problème est réductible au problème du voyageur de commerce [Lo84] ou au problème de la 3-satisfaisabilité [You96].

Parce que le problème de recherche d'une allocation optimale est NP-complet, on essaie rarement de trouver des solutions optimales, on se contente de solutions approchées, i.e. d'allocations qui ont des performances "acceptables".

2.2 Allocation statique

Une fois connue la structure d'une application parallèle, l'utilisateur se pose le problème de comment partager les composantes de l'application entre les différents processeurs de la machine parallèle qui sont à sa disposition, de sorte que l'exécution de l'application se passe dans les plus brefs délais. L'utilisateur a donc à résoudre un problème d'**allocation statique**, appelé aussi **placement** ayant comme objectif: minimiser le temps d'exécution de son application.

Les solutions qui existent pour le problème de placement dépendent du modèle choisi pour exprimer le temps d'exécution ou un autre critère de performances, de la topologie du réseau, du type de la solution recherchée (exacte ou approchée).

Nous n'avons nullement l'intention de faire une monographie complète du sujet. Les travaux de synthèse sur l'allocation statique sont nombreux: [AP88], [BCS89], [MT91a]; pour le problème particulier d'ordonnancement, il y a l'article de El-Rewini et al. [ERAL95].

Nous présenterons donc seulement les grandes classes des solutions qui existent dans la littérature. Le critère principal de classification que nous avons retenu est la qualité de la solution fournie: méthodes exactes, quand la solution trouvée vérifie la condition d'optimalité imposée, et méthodes approchées, quand la solution n'est pas forcément la meilleure, mais le temps dépensé pour l'obtenir est "raisonnablement" acceptable.

2.2.1 Méthodes exactes

Bien que NP-complet dans le cas général, il existe des cas particuliers où ce problème admet une résolution exacte en temps polynômial. C'est le cas lorsque le graphe de processeurs ou celui des taches présentent des caractéristiques particulières.

2.2.1.1 Topologies particulières

Plusieurs des méthodes qui fournissent pour des topologies particulières, telles que le graphe à deux nœuds, le réseau linéaire, l'anneau ou l'hypercube, des solutions exactes ont comme point de départ l'article de Stone [Sto77] qui donne un algorithme polynômial pour le cas de deux processeurs et fait une généralisation de son procédé pour le cas de n processeurs, sans pour autant que la solution soit, dans ce cas, optimale. Notons également que le problème de l'ordonnancement parallèle qui est NP-complet dans le cas général [VLL90] admet une solution polynomiale pour $n = 2$ processeurs [ERAL95].

Pour le cas de 2 processeurs Stone construit un réseau de transport en ajoutant au graphe des tâches deux nœuds (source et puits) qui correspondent aux processeurs. Les capacités des arcs sont données par les coûts de calcul des tâches et par les volumes de communication entre paires de tâches. Stone montre ensuite qu'une coupe dans ce réseau de transport correspond à une allocation et que la capacité de cette coupe est égale au coût de l'allocation. Alors trouver une allocation de coût minimal

revient à trouver une coupe minimale, ce qui signifie trouver un flot maximal, problème qui admet une solution en temps polynômial [GT86].

Bokhari [Bok79] donne une extension de cette méthode au cas dynamique où l'exécution de toutes les tâches se déroule en plusieurs phases; à la fin de chaque phase les tâches sont réallouées. Deux types de coût sont introduits: coût de résidence d'une tâche sur un processeur et coût de réallocation d'une tâche vers un processeur, autre qu'initial.

Dans le même article [Sto77], Stone donne aussi une idée d'extension de ce procédé de coupe dans le cas de n processeurs.

Deux autres extensions fournissent des allocations optimales: Cho, Lee et Kim [CLK92] pour les hypercubes et Lee, Lee et Kim [LLK92] pour les réseaux linéaires.

Dans son algorithme de coupes successives, Lo [Lo84] reprend la même idée; pour tout processeur p on applique le procédé de Stone en prenant p et $V-\{p\}$ pour les nœuds source et destination dans le réseau de flot. L'allocation A_p ainsi obtenue contient une restriction à l'ensemble $\{p\}$ d'une allocation optimale. Si à la fin toute tâche a été bien allouée (i.e. il existe $p \in V$ tel que $A_p(t) = p$), l'allocation est optimale, sinon on la complète.

Un autre cas pour lequel le problème d'allocation est polynômial est celui où le graphe des tâches est un arbre. Bokhari [Bok81a] propose la méthode suivante: on construit n copies du graphe des tâches, on ajoute encore $f+1$ nœuds (f étant le nombre des feuilles de l'arbre) et on forme un nouveau graphe valué. Dans ce graphe on recherche un arbre partiel qui relie les $f+1$ nœuds supplémentaires et qui indiquera l'allocation de coût minimum; l'algorithme est en $O(n^2 * m)$ temps.

Pour le graphe des processeurs en forme d'hypercube de dimension N et sous la condition de minimiser la somme des distances entre tâches, Antonio et Metzger [AM93] proposent une méthode tirée de la programmation non-linéaire qui aboutit à des solutions optimales; ils emploient la méthode du gradient projeté [BS79] pour obtenir une solution sur l'hypersphère de rayon 1 dans l'espace continu $\mathbb{R}^N$ qui est transformée ensuite en une solution discrète pour l'hypercube.

2.2.1.2 Topologies générales

Dans le cas général, l'espace des solutions au problème d'allocation optimale est exponentiel, d'où l'idée d'appliquer une méthode de recherche de type "branch-and-bound" ("séparation-évaluation partielle") afin de trouver une solution exacte.

L'idée générale de cette méthode est de structurer l'espace des solutions (y compris les solutions partielles) dans une arborescence qui sera parcourue d'une certaine façon (la première solution trouvée d'abord ou la meilleure etc.) en guidant ce parcours avec deux fonctions qui donnent une borne inférieure, respectivement supérieure, à la fonction coût de l'allocation finale; ceci permet d'élaguer certaines branches de l'arborescence.

Les premiers à avoir appliqué la méthode de "branch-and-bound" pour le problème d'allocation sont Ma, Lee et Tsuchiya [MLT82]. Leur allocateur est l'application directe de la méthode de recherche. La réduction de l'espace de recherche est faite à l'aide des restrictions qui s'imposent sur le placement des tâches sur certains processeurs et sur le placement de paire de tâches sur un même processeur.

Sinclair [Sin87] travaille dans le même cadre. Il considère des fonctions bornes inférieure et su-

périeure tout à fait naturelles sans se soucier de leur efficicacité. Par contre, il pense à réduire la profondeur de l'arbre de recherche en calculant d'abord le plus grand ensemble de tâches indépendantes (non-communicantes) et en faisant en premier l'allocation de ces tâches.

Magirou et Milis [MM89] utilisent une borne inférieure "raisonnable" en construisant pour toute allocation partielle une forêt recouvrante pour les tâches non encore allouées et en appliquant la méthode de Bokhari [Bok81a] qui admet une solution exacte devenant la nouvelle borne inférieure.

Il est évident que ce problème de placement est formulé en terme de programmation mathématique en variables bivalentes. Il est bien connu que les méthodes de résolution d'un tel problème sont en temps exponentiel. Billionet, Costa et Sutter [BCS92] ont résolu le problème primal à l'aide d'un algorithme "branch-and-bound" avec une politique de recherche sur la première branche. Les bornes inférieures pour la fonction objectif sont obtenues en considérant les solutions pour le problème dual Lagrangien qui est une relaxation du premier. Les auteurs signalent la qualité des solutions du problème dual.

Cosnard et Trystram [CT93] proposent une ébauche d'algorithme "branch-and-bound" pour le placement. Leur recherche est faite pour les nœuds qui ont la plus petite borne inférieure d'abord. Ils font l'observation que l'efficacité de cette méthode dépend de la finesse de cette fonction de sous–évaluation.

L'avantage des diverses formes d'algorithmes "branch-and-bound" est que l'on obtient à la fin une solution exacte, même si le temps peut devenir exponentiel. Des réductions de temps de calcul sont envisageables avec un bon choix des fonctions bornes et en tenant compte des spécificités du problème d'allocation.

2.2.2 Méthodes approchées

Dans la section concernant la modélisation du problème de recherche de l'allocation optimale nous avons mis en évidence combien il est difficile de trouver une solution exacte et combien l'espace des solutions possibles est grand. Pour ces raisons, on se contente parfois de solutions approchées dont on espère qu'elles sont proches de la solution optimale et, surtout, parce qu'e elles deviennent acceptables rapidement.

Il y a deux grandes classes d'approches permettant d'obtenir de telles solutions : les heuristiques et les méthodes de recherche combinatoire.

2.2.2.1 Heuristiques

Les heuristiques proposées pour résoudre le problème de placement sont nombreuses et elles ont toujours une fonction coût particulière à améliorer.

Bokhari propose dans [Bok81b] un algorithme d'allocation pour le cas $m < n$, i.e. moins de tâches que de processeurs. La qualité d'une allocation est donnée par sa cardinalité, qui est le nombre des tâches communicantes qui sont placées sur des processeurs voisins. On cherche toujours à améliorer une allocation en permutant deux tâches et, si ce n'est pas possible, on permute aléatoirement $[\sqrt{n}]$ tâches et on applique le même procédé d'amélioration. Le défaut de l'algorithme de Bokhari a été constaté par André et Pazat dans [AP88] : on ne prenait pas en compte la longueur des chemins entre des tâches communicantes non–voisines. André et Pazat corrigent ce défaut et font aussi l'extension pour le cas $m > n$ (plus de tâches que de processeurs).

Une classe à part d'heuristiques est constituée par les techniques de **groupement**; il s'agit de découper le graphe de tâches en n sous-graphes partiels connexes afin de minimiser la fonction coût et/ou le nombre des tâches communicantes qui ne sont pas situées sur le même processeur.

Dans [CE94], Calinescu et Evans proposent deux variantes d'une technique de groupement de processus sur les processeurs afin d'équilibrer la charge avant l'exécution. Les deux variantes s'appuient sur une recherche gloutonne : à chaque pas on cherche la tâche et le processeur qui vérifient une condition de minimalité, on fait l'allocation une fois pour toute ainsi que la mise à jour des charges estimées. La première variante cherche toujours le processus qui charge le moins possible un processeur, tandis que la deuxième fait cette recherche jusqu'à une valeur seuil en analysant après les communications induites par les processus qui vérifient cette condition de minimalité.

Sadayapan, Ercal et Ramanujan [ERS90], [SE87a], [SE87b] proposent des méthodes de groupement appliquées pour l'allocation sur l'hypercube et sur la grille. Pour l'hypercube le groupement se fait récursivement à partir de la grappe qui contient tout le graphe et à chaque pas on cherche à trouver une coupe en deux morceaux de toute grappe de l'itération précédente jusqu'on obtient un groupement complet.

Dans [Tal94] Talbi fait une démarche inverse : le point de départ est un groupement complet qu'il améliore ensuite par des méthodes générales de recherche tel que le recuit simulé ou les algorithmes génétiques.

Une technique de groupement à part est le **bi-partionnement** qui consiste récursivement à partitionner le graphe des tâches et le graphe de processeurs en deux parties et de faire le placement entre les deux. Une telle approche est décrite dans [BS94, Pel95, VR95]. Van Driessche et Roose dans [VR95] donnent aussi une caractérisation de la qualité du bipartionnement en fonction des propriétés spectrales des matrices associées aux graphes de départ.

2.2.2.2 Méthodes de recherche combinatoire

Pour appliquer ces méthodes il faut d'abord structurer l'espace des allocations en indiquant une relation de voisinage. Ceci permettra d'orienter la recherche selon les voisinages établis et la valeur de la fonction coût. Malheureusement cette fonction n'est pas monotone dans l'espace considéré et les minima locaux (dans tout son voisinage les coûts sont plus grands) ne sont pas toujours des minima globaux.

La **recherche itérative** est la plus simple à mettre en œuvre de ces méthodes. Elle consiste à parcourir l'espace des solutions en essayant toujours d'améliorer la fonction coût en passant de voisin en voisin. Dans [Tal93] on trouve une analyse de la façon dont le choix du voisin peut influencer la qualité de la solution finale.

Un algorithme glouton de recherche itérative est proposé dans [CG88]; au départ on prend la solution nulle et on ajoute chaque fois la paire (t', p') qui semble la mieux placée par rapport à l'allocation partielle déjà construite.

L'algorithme proposé dans [LA87] est une application en deux étapes de recherche itérative : dans un premier temps un algorithme glouton ajoute toujours à l'allocation partielle le nœud avec le volume le plus important de communications et ensuite on essaie d'améliorer cette allocation par échanges de paires de tâches situées sur des processeurs voisins.

Le plus souvent cette méthode simple est appliquée à titre d'exemple ou d'élément de comparaison avec d'autres techniques de placement [Nor93], [Tal93].

Même si les méthodes itératives permettent de trouver dans un temps raisonnable une solution, celle-ci n'est pas un optimum global, elle est plutôt un optimum local et toute possibilité de l'améliorer en passant par des solutions voisines est nulle, car il n'est pas permis d'augmenter la fonction coût. Afin de pallier ce manque les autres méthodes de recherche combinatoire autorisent de choisir une solution provisoire de moins bonne qualité. Le recuit simulé est de celles-là.

Le **recuit simulé** permet de choisir avec une certaine probabilité, convergente vers 0 dans le temps, des solutions "moins bonnes" en s'échappant du piège des optima locaux. Cette méthode est inspirée d'un procédé physique de traitement des métaux [AvL85] et a l'avantage d'être une méthode asymptotiquement convergente [AK89].

Pour le problème de placement, en dépit de sa très lente convergence, le recuit simulé offre des solutions de meilleure qualité [MT91a, Taw91, Nor93]. Elle est à cet effet utilisée comme méthode de référence pour diverses heuristiques [ERS90], [CL91].

Nombreux sont les algorithmes d'allocation statique fondés sur le recuit simulé. La méthode proposée par Wayne Bollinger et Midkiff [WM88] consiste en deux étapes: une pour faire l'allocation des processus et l'autre pour les liens de communication. Bultan et Aykanat [BA92] proposent une heuristique de placement basée sur une forme modifiée du recuit simulé où les conditions d'acceptation d'un nouvel état sont toujours constantes. Leurs comparaisons montrent que c'est une heuristique plus rapide que le recuit simulé classique, mais la qualité des solutions n'est pas meilleure.

Un autre avantage de la méthode du recuit simulé est sa simplicité de mise en œuvre. Il faut uniquement connaître l'organisation de l'espace de solutions et définir la probabilité d'acceptation des solutions moins bonnes. En marge de ces méthodes d'autres sont en train de se développer, soit en tirant parti des insuffisances des premières, soit en s'inspirant des modèles naturels d'amélioration des espèces. Ce sont respectivement les méthodes de recherche tabou et les algorithmes génétiques.

La **recherche tabou** est une technique des plus récentes [Glo89, Glo90]. Elle est plus délicate à mettre en œuvre [Tai93, GTdW93] car il faut redéfinir à chaque instant de la recherche la meilleure solution possible et un ensemble de points ou de déplacements interdits (tabou) afin d'éviter de repasser dans des états déjà visités. Il y a aussi des versions où le choix à faire est aléatoire [Glo90].

La recherche tabou est utilisée pour l'ordonnancement dans [PR93] et pour le placement dans [Taw91]. Tawbi [Taw91] a choisi de retenir comme ensemble tabou tous les derniers points visités et le choix d'une prochaine solution est fait dans le voisinage complet du point. Les résultats sont présentés comme comparables, à temps égal de recherche, à ceux du recuit simulé [Taw91].

Les méthodes vues jusqu'ici travaillent avec une seule solution qu'elles tentent d'améliorer. Les **algorithmes génétiques** [Gol88, Whi93] opèrent sur un ensemble des solutions qu'ils améliorent à chaque étape de recherche. Inspirés par la théorie évolutionniste, les algorithmes génétiques travaillent en probabilité en essayant d'améliorer globalement une population d'individus (i.e. l'ensemble de solutions). Une des étapes consiste dans une sélection des individus parmi les meilleurs de la population auxquels on applique les opérations de mutation, croisement et substitution afin d'obtenir des meilleurs

individus.

Lors de la mise en œuvre des algorithmes génétiques il faut codifier l'espace des solutions, définir les opérateurs génétiques et ensuite décider des valeurs des probabilités qui interviennent et de la taille de la population. La convergence dépend toujours de ces derniers paramètres [EAH90, NV92, Rud94], et cette convergence peut même se faire vers une solution autre que le minimum global [GS87, Cer93]. Il s'avère parfois difficile de définir les opérateurs génétiques, le croisement, plus particulièrement.

Il existe plusieurs travaux sur l'application des algorithmes génétiques au problème de placement [MSK87], [MT91b], [Law92], [Tal93].

Le grand intérêt pour ces algorithmes est dû au fait qu'ils sont facilement parallélisables de manière synchrone [Bal92], [Tal93] ou asynchrone [GS89], [Müh89].

2.3 Allocation dynamique

L'allocation statique exige une connaissance a priori de la structure de l'application et son usage est exclu pour les **applications dynamiques** dont les structures se modifient pendant l'exécution, quand de nouvelles charges se créent. Nous avons vu que l'allocation statique exige aussi une connaissance complète et détaillée de cette structure en termes de coût de calcul et de communication de chaque composante; parfois donc à défaut de ces informations, l'allocation statique décidée au démarrage de l'application peut s'avérer inefficace [BP92]. L'**allocation dynamique** est la solution adaptée pour répondre à de telles modifications de structure ou encore pour permettre la bonne continuation de l'application en cas de panne d'un processeur ou d'un lien physique.

Nous présenterons la problématique de l'allocation dynamique dans les systèmes parallèles et distribués faiblement couplés ainsi que les méthodes proposées dans la littérature sans être exhaustif (car il y a nombreux travaux de synthèse à ce sujet [CK88, BSS91, SKS92, JM93, Tal97, BF96] mis à jour périodiquement). Nous mettons cependant en évidence ses particularités par rapport à l'allocation statique ainsi que les éléments nécessaires à sa mise en œuvre.

Par souci de généricité, nous ne décrirons pas les algorithmes d'allocation dynamique spécialement conçus pour certaines applications : parallélisation de l'algorithme A* [DM94], branch and bound distribué [Dow95] ou programmation logique [LM92]; nous ne nous intéresserons pas non plus aux détails d'implantation des régulateurs de charge ou autres outils génériques. Quelques références bibliographiques pour cet aspect de l'allocation dynamique de charge sont : GATOS [BF92a], Utopia [ZZWD93], MOS [BP86], Dynamo [Tar94].

2.3.1 Définitions. Objectifs

Selon les notions introduites jusqu'ici on peut définir l'**allocation dynamique** comme le processus qui associe, pendant l'exécution d'une application, à chaque composante le processeur qui va l'exécuter afin d'améliorer les performances de l'application et/ou l'utilisation globale de la machine.

Nous avons défini des objectifs génériques d'une allocation, nous verrons plus loin ceux d'une allocation dynamique qui sont de type "minimiser" ou "maximiser" telle ou telle caractéristique globale de l'application et/ou de la machine.

Par rapport à un objectif fixé, une allocation dynamique est dite **optimale** si l'objectif est atteint.

Dans la section 2.1 nous avons montré que trouver une allocation optimale est un problème difficile dont la résolution exige la connaissance de la structure de l'application et du coût des composantes. Pendant l'exécution d'une application parallèle dynamique, acquérir de telles connaissances au niveau global peut s'avérer extrêmement difficile, voire impossible. D'une part il s'agit de prévoir les coûts de chaque nouvelle composante et, d'autre part, d'obtenir une information fiable en temps utile sur des éléments qui sont distribués sur les sites d'exécution de l'application.

Pour cette raison on ne rencontrera pas en général d'algorithmes optimaux d'allocation, sauf dans le cadre précis de la modélisation par files d'attente.

L'allocation dynamique est un processus qui est coûteux et qui peut dégrader les performances d'une application. Un algorithme d'allocation dynamique est dit **efficace** si le coût induit est couvert par le gain de temps obtenu en réalisant l'allocation dynamique. Le plus souvent l'efficacité prend en compte le temps total. La condition d'efficacité est donc :

$$T_{\text{application\_avec\_all\_dynamique}} + T_{\text{all\_dynamique}} < T_{\text{application\_sans\_all\_dynamique}}$$

On peut aussi utiliser un critère plus relaxé, celui d'effectivité [SKS92]. Un algorithme est **effectif** s'il améliore les performances du systèmes vis-à-vis d'un ensemble de mesures obtenues en l'absence de toute allocation. Par rapport à ces relations d'efficacité et d'effectivité, Wilkstrom et al. [WPG91] donnent des exemples d'applications pour lesquelles l'allocation dynamique se fait en temps raisonnable, sans que les conditions d'efficacité ou d'effectivité soient satisfaites, ni même qu'il y ait un gain de temps.

2.3.1.1 Objectifs

Minimiser le temps global d'une application parallèle est l'idéal de tout utilisateur; mais cette contrainte est très forte et elle est rarement utilisée [BF92b], car elle n'est par facile à exprimer dans un système distribué faiblement couplé; des conditions plus faibles sont alors exprimées en fonctions des processus (tâches) qui composent l'application. D'autre part l'objectif peut être fixé en rapport avec d'autres ressources que le temps global. Ainsi en pratique une condition objectif de l'allocation dynamique peut être parmi les suivantes :

- minimiser le temps total d'attente des processus [LO86]
- minimiser la longueur de la file d'attente CPU
- minimiser le taux d'occupation CPU
- minimiser le nombre de tâches actives sur un site
- minimiser le temps d'accès aux fichiers
- minimiser le temps moyen de réponse des tâches [HL86, SU87] (c'est la condition la plus utilisée)
- minimiser le facteur de non-balance qui est égal à l'écart-type des charges de chaque processeur [HL86]

- maximiser l'utilisation des ressources
- minimiser les communications
- minimiser le temps CPU restant des processeurs [LT94].

Parfois on impose deux ou plus de ces contraintes [Zat85], parfois complémentaires. Par exemple, Ni, Xu et Gendreau [NXG85] proposent de maximiser l'utilisation des ressources et de minimiser les communications et ils observent qu'elles sont complémentaires: réaliser l'une engendre la pénalisation de l'autre.

Hac et Johnson [HJ90] proposent de minimiser une fonction coût plus élaborée qui prend en compte la longueur de la file CPU, le taux d'occupation CPU, le nombre de tâches actives, l'accès distant ou local aux fichiers.

Selon les objectifs l'allocation dynamique sera qualifiée d'équilibrage de charge, de régulation ou de (re)distribution :
L'**équilibrage de charge** a pour objectif d'assurer à tous les processeurs la même charge de travail.
La **régulation** veut assurer un même niveau de charge de travail sur tous les processeurs.
La **distribution** se propose d'assurer du travail à tous les processeurs, pour qu'il y ait pas de processeurs oisifs (sans travail).

Dans les deux premiers cas, il apparaît un besoin de quantifier la charge de travail d'un processeur par un chiffre, l'**indicateur de charge.**

2.3.1.2 Indicateurs de charge

Comme indicateur de charge d'un processeur on peut prendre l'une des données suivantes :

- la longueur de la file d'attente CPU (i.e. le nombre de tâches allouées au processeur et pas encore exécutées)
- la différence entre le nombre de tâches arrivées et le nombre de tâches finies
- le temps total demandé par les tâches locales (temps de service)
- le travail estimé des charges non-finies [HL86].

voire une combinaison de 2 ou plus de ces valeurs [EB93].

On peut aussi considérer des paramètres synthétiques sur l'état global de charge du système: la valeur de la charge moyenne ou la variance à un moment donné, un facteur de variation globale de charge [XH91, XH93] calculé par rapport à un intervalle $[t, t']$:

$$r = \frac{|\sigma(charge^{(t')}) - \sigma(charge^{(t)})|}{\max(\sigma(charge^{(t')}), \sigma(charge^{(t)}))}$$

Un facteur local de variation [MDLT96] pour un processeur v est:

$$r(v) = \frac{|charge^{(t')}[v] - charge^{(t)}[v]|}{\max(charge^{(t')}[v], charge^{(t)}[v])}$$

où $charge^{(t)}[v]$ représente l'indicateur de charge du processeur v au moment t, $charge^{(t)}$ est le vecteur des indicateurs de charge du système, $\sigma(charge^{(t)})$ étant l'écart type de ce vecteur.

On peut imposer deux seuils par rapport auxquels les processeurs se définissent comme: peu chargés, normalement chargés et trop chargés.

L'indicateur de charge intervient à tous les niveaux de fonctionnement d'un algorithme d'allocation dynamique.

2.3.2 Composantes d'un algorithme d'allocation dynamique

Tout allocateur dynamique doit répondre à un ensemble de questions qui se synthétisent en deux questions essentielles : **quand** et **comment** procéder à l'allocation.

Ces deux questions regroupent d'une part: qui participe? quelles sont les informations nécessaires? comment les obtenir? combien faut-il transférer de charge? entre quels sites? C'est la partie **stratégie** d'un allocateur dynamique.

L'autre part correspond, en gros, à la question "quand faire l'allocation dynamique?" et aussi "quelles sont les unités de charges à transférer?". C'est la partie **mécanismes.**

Selon Jacqmot et Milgrom [JM93], la stratégie est liée globalement à la prise de décision. Les deux parties - stratégie et mécanismes - sont orthogonales, dans le sens qu'elles se conçoivent indépendamment l'une de l'autre. Toutefois Nicol [Nic92] remarque que l'efficacité d'un algorithme d'allocation dynamique, même si la stratégie est correcte, dépend entièrement des mécanismes.

Les performances globales d'un allocateur dynamique dépendent de la qualité de la stratégie et surtout des performances des mécanismes de mise en œuvre.

Sauf quelques exceptions, les stratégies sont *génériques*, elles sont conçues sans prendre en compte ni l'application pour laquelle se fera l'allocation dynamique, ni les performances de l'architecture support. Par contre, les paramètres des mécanismes dépendent fortement des caractéristiques de l'application et de l'architecture physique telles que la vitesse de calcul des processeurs, les temps de communication entre processeurs, le débit des liens physiques.

2.3.2.1 Stratégies

A la toute première question posée, à savoir "qui participe à l'allocation dynamique?" la réponse peut paraître quasi-évidente: tous les processeurs, car tous les processeurs ont de la charge à traiter. Mais il s'agit plutôt de savoir si tous les processeurs participent en même temps ou si chaque processeur participe uniquement s'il est sollicité. La première approche: tous les processeurs participent à l'allocation en même temps est caractéristique de l'allocation des charges régulières, sujet qui sera largement traité dans la section 2.4. Dans la littérature [FDM93] on parle donc des stratégies *locales* et *globales*. Dans cette partie notre attention sera concentrée sur les stratégies locales.

Une stratégie d'allocation dynamique a pour but de décider du transfert de charges entre sites. Pour cela les nœuds (processeurs) doivent, en principe, être informés[4]. Cette *information* acquise

4. Il existe toutefois des stratégies qui n'utilisent aucune autre information que l'indicateur local de charge

sur l'état de charge du système et nécessaire à l'allocation peut être *distribuée* sur chaque site ou *centralisée* sur un site privilégié; elle peut être aussi *locale*, concernant uniquement le voisinage proche du site, ou *globale*, concernant les indicateurs de charge de tous les processeurs.

Quant au choix du site vers lequel/duquel la charge est transférée (ce que dans la littérature on appelle la politique de localisation [SKS92]) il y a deux possibilités :

- le site se trouve dans le voisinage - c'est le cas le plus courant;
- le site se trouve dans tout le système, comme c'est le cas des stratégies à information globale ou la méthode du gradient [LK87] et ses dérivées.

Xu et Hwang [XH91] proposent 4 versions d'une politique de localisation : globale ou dans le voisinage, de manière circulaire (on regarde tour à tour tous les voisins) ou la politique du moins chargé d'abord; leurs simulations donnent de meilleurs résultats pour le choix local de la destination et pour la version circulaire.

La stratégie elle-même peut changer pendant l'exécution de l'application. De tels algorithmes d'allocation dynamiques sont dits **adaptatifs** [SKS92].

En raison de la nature distribuée des applications traitées, ainsi que des machines qui les traitent, il semble plus approprié qu'un allocateur soit distribué. On distingue cependant des allocateurs dont les stratégies sont centralisées ou hybrides.

Approches centralisées Cette classe de stratégies d'allocation suppose une vision globale de l'état de charge du système.

C'est typique des systèmes d'exploitation multiprocesseurs. Comme décrit dans [LO86], il existe un superviseur qui détient pour chaque processeur son état de charge et qui décide à lui tout seul quels sont les transferts à mettre en œuvre.

Une autre approche à information globale est celle par files d'attente[5] [Hac89]. Dans ce cas, chaque processeur est modélisé par un site formé d'une unité de service à laquelle sont attachées des files d'attente de processus permanents à traiter, créés sur le site ou venus d'autres sites. Le système qui nécessite l'allocation se modélise donc par un réseau d'interconnexion de sites. Tous les auteurs [HJ90, dG91, PTS88, TT85] considèrent que les transferts de charge entre sites sont immédiats et gratuits et que l'évolution du système est connue a priori et statique, c'est à dire tous les paramètres qui gouvernent le fonctionnement : taux d'arrivée dans les files, taux de service sont figés et connus. En fonction de ces paramètres se pose la question de trouver la meilleure façon de transférer : seuils de transfert, choix des sites destinations. Ces paramètres sont déterminés en résolvant des systèmes d'équations.

Ces deux approches centralisées sont assez loin de la réalité. Dans le premier cas, le site privilégié d'information–décision devient vite un goulot d'étranglement surtout si le nombre de processeurs est important; dans le deuxième cas la vision globale et stationnaire ne correspond pas aux applications réelles, car la dynamique n'est pas prise en compte. Malgré cela, l'approche reste intéressante parce qu'elle permet d'avoir un modèle d'analyse de la *stabilité* du point de vue de la théorie des files d'attente : l'arrivée globale des processus ne dépasse pas la capacité globale de traitement.

5. Voir [Kle75] pour une introduction à la théorie des files d'attente.

Approches semi-distribuées L'objection qu'on avait pour l'une des approches centralisées concernait l'existence d'un seul nœud "privilégié" qui concentrait toute l'information de charge du réseau et qui avait la tâche de prendre toutes les décisions de transfert, en contrepartie, la facilité de prise de décision était évidente.

Les approche semi-distribuées gardent l'idée du site privilégié, mais il n'est plus unique. Il s'agit des techniques de **groupement** [AG91, BE94, CE94, ZZWD93]; le réseau est partagé en groupes disjoints, chaque groupe dispose d'un site privilégié qui concentre l'information de charge du groupe et des informations synthétiques sur l'état de charge des autres groupes. La régulation de charge d'un processeur passe par le centre du groupe qui essaie de la faire à l'intérieur du groupe, si possible, sinon, un autre groupe est choisi pour faire le transfert de charge.

Peu d'auteurs se soucient de comment partager le graphe en groupes disjoints qui respectent des contraintes de connectivité et de distance. Ahmad et Ghafoor [AG91] et Boulouias et Gopal [BG89] montrent que ce problème de décomposition est NP-complet. Pour cette raison, Ahmad et Ghafoor [AG91] s'occupent de cette approche pour des graphes particuliers du réseau d'interconnexion (graphes distances-transitives) où la décomposition en groupes ("sphères") disjoints est facile.

Dans la plupart des cas, les groupements se font sur deux niveaux uniquement. Willebeek-LeMair et Reeves [WLR93] proposent une stratégie qui fonctionne sur plusieurs niveaux, le réseau des processeurs étant logiquement structuré en arbre binaire complet.

Evans et Butt [EB94] proposent une stratégie de groupement sur deux niveaux, mais pendant l'allocation dynamique la composition des groupes peut changer pour s'adapter à la variation de la charge du système.

Approches distribuées Les approches complètement distribuées sont de loin les plus utilisées pour l'allocation dynamique. Van Tilborg et Whittie [vTW84] remarquent que c'est la façon la plus naturelle de faire du moment que l'application est distribuée.

Au moment où la décision sur l'opportunité de l'allocation dynamique a été prise, à savoir si un processeur a peu ou trop de charge ou si une autre condition est satisfaite, il faut trouver un autre processeur de la part duquel ou vers lequel de la charge sera à transférer. Pour résoudre ce problème de base de toute stratégie d'allocation dynamique, il faut avoir obtenu un minimum d'information sur l'état de charge des autres processeurs (une partie ou tous) et prendre une décision sur le montant de charge qu'il faut transférer ou accepter.

Ça peut sembler paradoxal, mais il y a des **stratégies aveugles** qui n'utilisent aucune information distante lors de la prise de décision. Si la condition de déclenchement est satisfaite (trop de charge), un site destination est choisi aléatoirement parmi les voisins et un processus est envoyé vers ce site. Afin d'éviter de surcharger le site choisi (déjà surchargé) un protocole de réservation peut être mis en place : l'initiateur informe sur ses intentions d'envoyer de la charge et attend la réponse. En cas de rejet la procédure de choix est refaite jusqu'à un nombre fini de fois. Malgré ce manque d'information cette méthode aveugle et aléatoire se comporte très bien [ELZ86]. Barak et Shiloh [BS85] choisissent d'envoyer non pas un seul processus en cas de surcharge, mais la moitié de la charge du site.

Lors de la phase d'information les stratégies de régulation peuvent acquérir divers types d'informations : sur l'état de charge du plus proche voisinage ou sur l'état de charge global, soit complet (cas

très coûteux en temps) soit incomplet, juste des valeurs synthétiques sur la charge du système ou des informations sur l'état de charge d'autres nœuds particuliers plus éloignés.

Les **méthodes à information locale** sont faciles à mettre en œuvre car la phase d'information consiste en simples échanges des indicateurs de charge entre voisins. Le choix de la destination (de la source) du transfert se fait alors dans ce proche voisinage. Lorsque plusieurs sites sont candidats, un deuxième choix peut porter sur l'état de charge, par exemple, celui avec la meilleure charge (la plus petite s'il s'agit d'un envoi ou la plus grande pour une réception) [ELZ86].

La choix de la cible peut se faire aussi de façon **probabiliste**. Cela signifie trouver pour l'ensemble $\{u_1, u_2, \cdots u_k\}$ des cibles possibles d'un nœud v des probabilités $p_1, p_2, \cdots p_k$ telles que le transfert vers u_i se fasse avec la probabilité p_i. Dans [ELZ86] ces probabilités sont égales entre elles. Evans et Butt [EB93] prennent les probabilités $1 - p_i, i = 1, k$ directement proportionnelles aux indicateurs de charge des nœuds $u_i, i = 1, k$. Hsu et Liu [HL86] prennent les valeurs p_i selon l'indicateur de charge (la longueur de la file d'attente ou le temps moyen de service) et selon le nombre des tâches déjà transférées sur le processeur u_i.

D'autres stratégies comme la **méthode du gradient** et ses dérivées propagent de proche en proche des informations sur certains nœuds susceptibles de devenir des cibles de transfert. Ces méthodes essayent en même temps de réduire au minimum la taille des informations utiles. La méthode du gradient a été proposée par Lin et Keller [LK87], l'idée est de connaître non pas un nœud sous-chargé, mais la distance vers un tel nœud qui permettrait l'acheminement facile de la charge et, en cas de changement brusque de l'état de celui-ci retrouver un autre nœud sous-chargé. Chaque nœud v détient donc $d(v)$ la distance vers le plus proche nœud sous-chargé. Cette information est mise à jour lors des possibles changements des états de charge et propagée de proche en proche. Depuis un nœud sur-chargé v, le transfert est fait d'abord vers un proche voisin u situé à une plus petite distance d'un sous-chargé $d(u) = d(v) - 1$. Après (au moins) $d(v)$ transferts la charge va aboutir dans un nœud sous-chargé.

Muniz et Zaluska [MZ95] trouvent deux désavantages à la méthode du gradient dans sa forme initiale :

- l'information se propage de voisin en voisin et dans le pire des cas après une distance d entre le processeur source et destination l'information peut être périmée;
- s'il y a peu de processeurs sous-chargés et si les processeurs surchargés émettent beaucoup de processus, il y a un effet évident de surcharge.

L'extension qu'ils proposent est basée sur un système de réservation auprès des processeurs sous-chargés qui avant d'accepter des charges extérieures envoient un acquittement. Il y a un net surcoût de mise en œuvre qui est comblé dans certaines configurations de distribution de charge.

Kalé [Kal88] propose une version localisée de la méthode du gradient: les processus envoyés vers des sites sous-chargés ne s'arrêtent qu'après une distance minimum et ils ne dépassent pas une certaine distance maximum. Cette méthode améliore légèrement la méthode initiale en la rendant *stable*, car on empêche le vagabondage des processus dans le réseau.

Lüling, Monien et Ramme [LMR91] donnent une autre extension de la méthode du gradient. Ils construisent en plus pour chaque nœud v la distance $D(v)$ vers le plus proche surchargé. L'envoi de

charge est fait aussi quand la charge est normale vers un voisin plus éloigné d'un nœud sur-chargé. Ce double mécanisme assure une meilleure distribution de charge.

L'algorithme évolutif de Roman [Rom97] ne garde pas une seule valeur qui soit une distance, mais un vecteur des identificateurs de processeurs les plus proches qui ont un petit niveau de charge. La mise à jour de ce vecteur se fait par des opérateurs de croisement et fusion.

La collecte d'informations pendant la stratégie de régulation a pour but de trouver les bonnes destinations pour les envois de charge. La méthode du gradient utilisait un minimum d'information possible, d'autres stratégies sont encore plus élaborées et utilisent plus d'information.

Ni, Xu et Gendreau [NXG85] proposent que tous les processeurs gardent un état de charge d'autres processeurs (une partie ou tous) et que les processeurs faiblement chargés envoient des demandes de charge vers les processeurs fortement chargés jusqu'à ce qu'il deviennent normalement chargés.

Guyennet et Spies [GS93] forment un anneau virtuel des processeurs faiblement chargés, les processeurs fortement chargés sont liés virtuellement à un des processeurs de l'anneau et c'est à lui ou au moins chargé de l'anneau qu'il enverra sa surcharge.

Shamir et Upfal [SU87] font circuler dans le réseau de façon aléatoire un paquet qui contient les adresses des processeurs les plus chargés, si le paquet rencontre un surchargé son contenu peut être modifié, s'il rencontre un processeur faiblement chargé celui-là va se coupler avec le plus chargé du paquet afin de le délester de sa surcharge.

2.3.2.2 Mécanismes

Les **mécanismes** d'une allocation dynamique concernent :

- la politique d'initiation;
- la politique de décision;
- la politique de transfert.

Dans la présentation qui va suivre on mettra l'accent sur la politique d'initiation et sur la politique de transfert, car la politique de décision est toujours confondue avec la politique d'initiation, qu'on appelle d'ailleurs politique de déclenchement.

Politique d'initiation Il s'agit de détecter les situations où l'allocation dynamique serait bénéfique pour améliorer les performances de l'application concernée ou du système. Comme la plupart des allocateurs dynamiques sont distribués il faut pouvoir se rendre compte localement d'un possible déséquilibre entre les charges des nœuds.

Dans la littérature on distingue trois classes de politiques d'initiation, deux étant complètement symétriques [WLR93, SKS92] :

- receveur–initiateur, où les nœuds initient l'allocation dynamique quand ils sont (s'estiment) sous-chargés [RSAU91, NXG85];

- envoyeur–initiateur, où les nœuds qui initient sont (s'estiment) surchargés [LK87, GS93];

- mixte, où les deux catégories de nœuds initient ou même les nœuds normalement chargés [LMR91].

Un nœud s'estime sous-chargé ou surchagé par rapport à des seuils de charge [ELZ86, LK87]. Le processeur compare sa propre charge à ces seuils où à d'autres critères comme, par exemple, le plus long temps d'attente d'une tâche [BP92].

Les valeurs de ces seuils peuvent être des valeurs constantes [LK87] ou des fonctions de charge comme dans [XH93] où le seuil de surcharge est égal à $(1+\alpha) * \overline{charge}$, $\overline{charge}$ étant soit la charge moyenne du réseau, soit la charge moyenne du voisinage et α une constante positive. D'autres auteurs [MDLT96, FDM93] proposent des expression mathématiques plus laborieuses.

On peut faire la remarque simple qu'un seuil variable qui reflète l'évolution du système convient mieux qu'un seuil constant surtout dans le cas où les charges des nœuds varient toutes dans le même sens (par exemple vers la fin de l'application) sans pour autant avoir besoin de régulation.

Ces comparaisons ne se font pas systématiquement, à tout instant, mais juste en cas de variation importante de la charge ou périodiquement, en fonction d'une période T [RSAU91]. Calinescu et Evans [CE94] analysent l'influence de cette valeur de la période en mettant en évidence par simulation que le facteur de non-équilibre est plus faible quand l'intervalle de temps est plus réduit. Dans des systèmes de files d'attente l'intervalle T dépend du taux d'occupation de tous les sites [HJ90] ou il est mis à jour en fonction des taux d'arrivée et de service sur les autres sites [PTS88].

Politique de décision La décision sur l'opportunité de l'allocation de charge intervient après qu'un minimum d'informations sur l'état de charge général ou du voisinage soit obtenu.

Dans [WLR93], on compare l'écart entre la charge du nœud qui a initié la collecte d'information et la charge moyenne du voisinage avec un seuil établi sur la base d'une relation de profitabilité de la régulation.

Pour les stratégies centralisées la décision est directement prise par le processeur qui concentre l'information de charge [LO86].

Le chapitre 3 de cette thèse présentera un algorithme d'équilibrage pour charges régulières avec une politique de décision fondée sur l'efficacité de l'allocation dynamique.

Politique de transfert La politique de transfert repose sur les informations obtenues et les décisions prises concernant le tranfert de charge. Elle décide au niveau du processeur **"quoi"** (quelle charge de travail) transférer.

Ce choix sur la charge à transférer peut se faire de plusieurs façons:

- le processus le plus récent (c'est le cas de transfert le plus courant)

- le plus vieux processus qui tourne parce qu'il a les plus fortes chances de tourner encore [GS93]

- les processus permanents et plus grands consommateurs de temps CPU [BP86]

- les processus pour lesquels les opérations d'entrée/sortie coûtent moins cher sur le site distant [BP86].

Parfois, dans la littérature on rencontre l'hypothèse que le transfert des processus d'un site à l'autre est immédiat, il ne coûte rien. Dans ce cas, les algorithmes sont simples et les fonctions mathématiques qui modélisent le coût de l'allocation ont de bonnes propriétés; il est facile de trouver des algorithmes optimaux : par exemple, tous les algorithmes basés sur le modèle des files d'attente ou l'algorithme en spirale de Leland et Ott [LO86].

La charge à transférer peut avoir déjà commencé à être exécutée auquel cas il faut l'interrompre en vue d'une reprise ultérieure ou d'une re-exécution sur un autre site. On parle alors de **migration.**

Ce mécanisme de **préemption** suppose, d'une part, la définition des points où l'exécution peut être interrompue et reprise (des points de reprise) et, d'autre part, l'existence d'un mécanisme de sauvegarde du contexte d'exécution (état des registres, de la mémoire, des communications).

Lors d'une migration aux problèmes de préemption s'ajoute celui du transfert du contexte d'exécution et de l'espace de nommage [Ell94]; ceci suppose la garantie que les messages destinés à une tâche migrée arriveront à la nouvelle destination. A ce sujet, Elleuch et Muntean [EM94] analysent des protocoles possibles : geler les communications et envoyer un message avec la nouvelle destination à tous les correspondants possibles, rediriger les messages, rejeter les messages et envoyer aux émetteurs un message d'avertissement.

Bernard et Folliot [BF96] font la remarque que, intuitivement, si le coût de la migration était petit, la migration ne serait que bénéfique. Lennerstad et Lundnberg [LL94] vont plus loin et prouvent que dans l'hypothèse où la migration/ préemption a un coût nul, l'efficacité maximale obtenue est dépendante uniquement du nombre des tâches et des processeurs, comprise entre 2 et 2.5. Eager et al. [ELZ88] analysent aussi le coût de la migration en montrant que le gain obtenu n'est jamais supérieur à 40%.

2.4 Stratégies d'allocation dynamique de charges régulières

Les **charges régulières** sont des charges de même type dont les unités de charge sont indépendantes et requièrent le même traitement. L'absence de dépendances entre les unités engendre le fait que toute unité de charge peut être traitée sur n'importe quel processeur et dans n'importe quel ordre[6]. Cette hypothèse qui semble être assez restrictive couvre en fait un vaste ensemble d'applications très exigeantes en puissance de calcul. C'est, par exemple, le cas pour les simulations moléculaires dans la physique des particules [BF90], [BBK91], pour le traitement d'images [Mig92], [GO93], pour la mécanique des fluides [Hei94], etc. Pour de telles applications le processus d'allocation de charges consiste plutôt en une **régulation de charges**, i.e. le maintien du même nombre d'unités de charges (données ou processus) sur chaque processeur.

Vu les spécificités des charges régulières, on a intérêt à faire la régulation en même temps sur tous les sites de la machine parallèle. Cette approche rend ces stratégies fiables et plus faciles à mettre en œuvre que les algorithmes classiques d'allocation dynamique (voir la partie précédente).

6. Dans la thèse de Fonlupt [Fon94] les algorithmes qui traitent des charges régulières sont appelés algorithme à pile

Nous allons nous concentrer seulement sur la composante "stratégie" de la régulation, c'est à dire "**comment**" aboutir à l'état d'équilibre, en laissant de côté les aspects liés à l'implantation concrète de telles stratégies: qui décide et comment décide t-on d'initier la régulation, qui décide de son opportunité lors de son initialisation, quelle politique de transfert de charge adopter (qui transférer, de quelle façon, de la migration ou de la préemption, comment transférer le contexte) tant de problèmes qui sont résolus dans le cas général.

Nous allons d'abord donner une modélisation du problème de régulation puis une classification des solutions basée sur la façon d'acquérir l'information nécessaire pour mettre en œuvre la régulation. Ensuite nous présenterons les deux grandes classes de solutions: avec informations locales et avec information globales.

2.4.1 Modélisation. Complexité. Classification

2.4.1.1 Modélisation du problème de régulation

Etant donné un réseau d'interconnexion $G = (V, E)$ de processeurs, le problème de **régulation de charge** est le suivant: sachant que chaque nœud $v \in V$ détient un certain nombre d'unités de charge $charge[v]$, comment réaliser des transferts de charge le plus efficacement possible de sorte que chaque processeur détienne le même nombre d'unité de charge (plus ou moins 1), c'est à dire

$$\left\lfloor \frac{1}{n} \sum_{v \in V} charge[v] \right\rfloor \quad \text{ou} \quad \left\lceil \frac{1}{n} \sum_{v \in V} charge[v] \right\rceil \tag{2.2}$$

Le nombre d'unités de charge détenu par un nœud est quelconque, mais parfois on ajoute une hypothèse [JR92] qui borne le nombre d'unités de charge qu'un nœud peut détenir au début

$$charge[v] \leq M, v \in V$$

Il est parfois difficile d'obtenir l'état d'équilibre tel que défini par (2.2), cette condition peut se relâcher de la façon suivante (comme dans [GLM+96])

$$|charge[v] - \overline{charge}| < M$$

où $\overline{charge} = \frac{1}{n} * \sum_{v \in V} charge[v]$. Dans le même article il est proposé une définition de l'**état d'équilibre local**:

$$|charge[v] - charge[u]| < M', \forall (u, v) \in E.$$

Imposer de garder à tout instant une condition d'équilibre local des voisinages est une façon d'aboutir à l'état d'équilibre, comme dans les méthodes dites diffusives (voir la sous-section 2.4.3.1) où on essaie à chaque pas de réduire l'écart entre voisins.

Comme mesure de déséquilibre dans le système, on fait appel à la variance de charge [Boi90, Cyb89]:

$$E^2(charge) = \frac{1}{n} \sum_{v \in V} (\overline{charge} - charge[v])^2$$

ou l'écart par rapport à la valeur moyenne [GLM+96]:

$$\Delta(charge) = \max_{v \in V} (charge[v] - \overline{charge})$$

2.4.1.2 Coût de la régulation

Dans la définition nous avons utilisé les mots "plus efficacement possible", ce qui signifie l'existence d'un ou de plusieurs critères pour évaluer le coût de la régulation. L'environnement pour lequel nous avons posé le problème de la régulation est distribué, donc toute solution de régulation est un algorithme distribué.

On peut faire une équivalence entre le coût de la régulation et la complexité de cet algorithme distribué en terme de calcul, temps de communication, nombre total de messages, plus long chemin dans le graphe des communication induit, temps total, complexité de Kolmogorov [7].

On peut aussi imposer comme coût de la régulation un autre critère comme, par exemple, le goulot maximal d'etranglement formé.

Dans tous les cas l'idéal serait de trouver la régulation optimale, i.e. de coût minimum.

2.4.1.3 Modélisation par des problèmes de flot. Complexité du calcul dans l'hypothèse séquentielle et centralisée

Résoudre le problème de la régulation c'est d'abord trouver la quantité de charge que chaque nœud doit échanger (envoyer / recevoir) avec ses voisins directs puis trouver une politique correcte et efficace afin de réaliser les transferts de charge.

Plus formellement, il s'agit d'abord de trouver une fonction $transfert : E \rightarrow I\!R$ telle que :

- si $transfert(u,v) > 0$, le nœud u envoie au nœud v une quantité de charge de $transfert(u,v)$;
- si $transfert(u,v) < 0$, l'envoi est de v vers u;
- pour tout nœud u : $charge[u] + \sum_{(u,v)\in E} transfert(u,v) = \overline{charge}$.

Les deux premières assertions indiquent le sens du transfert de charge entre deux nœuds, alors que la dernière assure à la fin de la régulation à chaque processeur une quantité de charge égale à $\overline{charge}$ unités.

Ceci revient à trouver un flot maximal dans le réseau $S = (V', E', c)$, où $V' = V \cup \{s,t\}$, $E' = E \cup \{(s,u)|charge[u] > \overline{charge}\} \cup \{(v,t)|charge[v] < \overline{charge}\}$, $c : I\!R \rightarrow E$ est la fonction de capacité :
$c(u,v) = \infty$, $\forall (u,v) \in E$;
$c(s,u) = charge[u] - \overline{charge}$;
$c(v,t) = \overline{charge} - charge[v]$.

Le problème du flot maximal sans imposer aucune restriction est polynômial (la meilleure solution connue pour l'instant est en $O(n * m * \log(n^2/m))$ [GT86]) [8].

Lié aux définitions possibles du coût de la régulation on peut ajouter des restrictions à ce problème de recherche du flot maximal :

- Minimisation du flux maximal d'un arc [Ben96] – problème polynômial. Ceci correspond à prendre comme coût de la régulation la valeur du transfert de charge le plus important;

7. Voir [Lav95] pour la définition de ces complexités.
8. Rappelons-nous que n représente le nombre des nœuds et m le nombre des arêtes.

- Minimisation du coût du flot – problème NP-complet [GJ79]. Ce qui signifie que la somme des transferts soit minimale;

- Minimisation du flux maximal des chemins entre les nœuds – problème NP-complet car réductible au problème du chemin de coût maximal. Cette restriction correspond à prendre le plus long chemin comme coût de la régulation.

Même dans le cas centralisé, la résolution du problème de régulation de charge est coûteuse, voire très coûteuse. Il faut donc trouver un compromis entre le coût du calcul de la régulation et le coût de la régulation elle-même. Ainsi dans le cadre réel distribué du problème de la régulation nous n'allons pas chercher la solution optimale par rapport au coût de la régulation, mais des solutions approchées.

2.4.1.4 Problème de la redistribution des jetons

D'un point de vue théorique, le problème de régulation de charge est une généralisation du problème de la redistribution des jetons [PU86]; un jeton peut être assimilé à une unité de charge, à la différence qu'un jeton est toujours envoyé tout seul, jamais groupé avec d'autres jetons. Le problème est donc le suivant: étant donné un réseau $G = (V, E)$ avec $|V| = n$ et n jetons distribués dans le réseau tels qu'un processeur n'ait pas plus de K jetons, comment redistribuer les jetons de telle sorte que chaque nœud ait exactement 1 jeton. Peleg et Upfal dans [PU86] ont analysé la complexité de ce problème de la (n, K)-redistribution trouvant une borne de $\Omega(K + logn)$ pas d'envois et ont proposé un algorithme distribué qui atteint cette borne pour certaines topologies particulières.

Ghosh et al. [GLM+96] reprennent le problème de la redistribution dans un cas plus général où le nombre total de jetons est quelconque. Ils trouvent pour ce problème des bornes supérieures de la complexité en temps des algorithmes proposés mais aussi du problème dans une approche séquentielle, en fonction de la taille et du type du réseau (port unique ou multi-port), l'écart maximum et aussi de l'expansion[9] du réseau.

2.4.2 Classification des méthodes de régulation

Dans notre cadre, i.e. machine parallèle à mémoire distribuée, il manque d'abord une vue globale et complète de l'état du réseau et, ensuite, si on a cette vue globale complète, il faut qu'elle soit dans tous les nœuds et que chaque nœud soit capable de décider lui–même les démarches à faire en terme de communications avec les voisins.

Mais il serait très coûteux de collecter toutes les informations sur l'état du système, c'est à dire que tout nœud connaisse le vecteur $charge[v]$, $v \in V$. (En terme de communication il faudra assurer l'**échange total**, voir le chapitre 4 de [dR95].) Se pose aussi, dans l'hypothèse d'un échange total le problème de la correction des informations obtenues, car elles risquent de devenir obsolètes durant cette opération d'échange qui est loin d'être instantanée.

Les informations obtenues le plus simplement possible sont les valeurs $charge[w]$, pour des nœuds w voisins. Certaines méthodes utilisant seulement ce type d'informations collectées à des instants bien

9. Voir annexe A pour la définition de l'expansion d'un graphe.

précis qui leur permet d'aboutir de manière exacte ou approchée dans l'état équilibré. Ces méthodes sont dites **avec informations locales**. Nous allons développer ces méthodes dans la section 2.4.3. Un autre type de méthodes est celui des méthodes **avec informations globales** qui ne se contentent plus d'informations sur les voisins. Pour les machines parallèles avec un noyau de communication extrêmement fiable les solutions qui prennent en compte les informations complètes de charge se font rares. Nous en verrons un exemple [Nic92]. Dans la plupart des algorithmes, les informations globales acquises ne sont pas complètes, elles sont synthétiques: la charge moyenne du réseau, la charge totale d'un ou plusieurs sous-réseaux, qui est une solution mieux adaptée pour les machines massivement parallèles. Ces méthodes sont exactes, mais il faut prendre en compte le coût d'acquisition des informations qui dépassent le cadre de leur voisinage. La section 2.4.4 sera consacrée aux méthodes avec informations globales.

La figure 2.1 dresse une classification de ces méthodes.

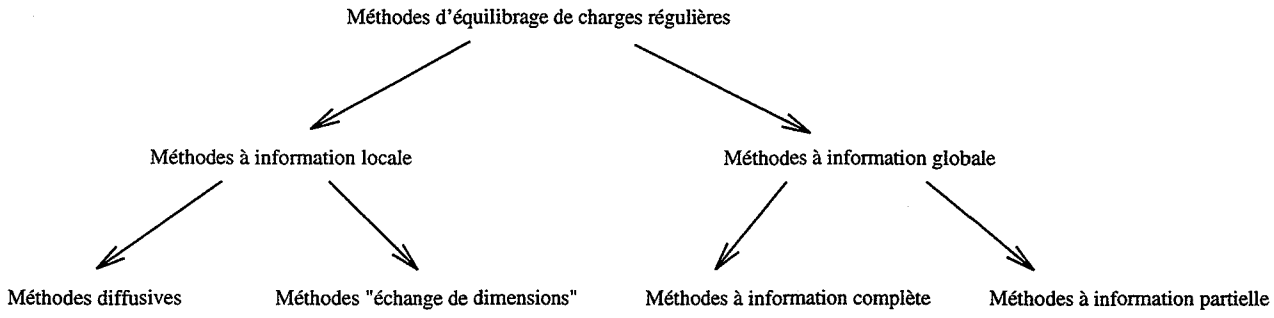


FIG. 2.1 – *Une classification des stratégies d'équilibrage de charges régulières*

2.4.3 Méthodes avec informations locales

Les méthodes de régulation de la charge qui n'utilisent que des informations sur le proche voisinage d'un nœud seront appelées méthodes avec informations locales. Elles sont très intéressantes parce que leur champ d'utilisation est complet: toute topologie d'interconnexion des processeurs peut les utiliser sans demander des fonctions spéciales de communications, autres que la communication point-à-point avec ses voisins. Leur défaut principal est qu'en général ce ne sont pas des méthodes exactes: la convergence vers un état d'équilibre est asymptotique. Il faut aussi dire que cette convergence suppose une manipulation de transfert des quantités des charges parfois non-entières, d'où l'exigence de plus grandes précautions lors de leur implantation.

2.4.3.1 Méthodes diffusives

Les méthodes de cette classe ont comme point de départ les travaux de Cybenko [Cyb89] et de Boillat [Boi90]. Nous allons présenter le principe de base de la méthode diffusive suivi d'une modélisation mathématique afin de justifier les études de convergence qui ont été faites. Nous allons montrer aussi comment résoudre le problème d'échange des unités de données non-unitaires qui apparaîssait

dans la solution initiale. Le paragraphe de la fin sera consacré aux extensions qui existent dans la littérature pour cette méthode.

Principe de base L'algorithme de la méthode initiale a une forme simple et naturelle, chaque nœud $u \in V$ exécute itérativement de façon synchrone le code suivant :

```
/* Etape d'information */
pour tout v voisin
    reçoit(charge[v]);
à tout voisin v
    envoie(charge[u]);

/* Etape de calcul */
ma_charge := charge[u];
pour tout v voisin
    si charge[u] > charge[v]    envoi[u,v] = (charge[u] - charge[v]) * poids(u,v);
       sinon envoi[u,v] = 0;
    ma_charge := ma_charge - envoi[u,v];

/*Etape d'échange */
pour tout v voisin
    si envoi[u,v] ≠ 0 envoie au voisin v une quantité de charge de envoi[u,v];
fait toutes les réceptions de charge des voisins;
```

où $charge[x]$ indique la quantité de charge, exprimée en nombre d'unités, détenue dans un nœud x et $poids(x, y)$ est un facteur de pondération, qui est défini par l'utilisateur, en général symétrique. Nous verrons dans les sections suivantes quelques propositions pour le choix de ce paramètre.

Ce code est itéré jusqu'à ce qu'une condition d'arrêt soit atteinte.

Modélisation mathématique Nous allons exprimer la charge de chaque nœud $u \in V$, $charge^{(t)}[u]$, après t itérations en fonction de la configuration initiale $charge^{(0)}[v]$, $v \in V$.

A la fin d'une itération $t+1$ on peut exprimer $charge^{(t+1)}[u]$ de manière suivante :

$$\begin{aligned}
charge^{(t+1)}[u] &= charge^{(t)}[u] - \sum_{v \in N(u)} envoi[u,v] + \sum_{w \in N(u)} envoi[w,u] \\
&= charge^{(t)}[u] - \sum_{\substack{v \in N(u) \\ charge^{(t)}[u] > charge^{(t)}[v]}} poids(u,v) * (charge^{(t)}[u] - charge^{(t)}[v]) \\
&\quad + \sum_{\substack{w \in N(u) \\ charge^{(t)}[w] > charge^{(t)}[u]}} poids(w,u) * (charge^{(t)}[w] - charge^{(t)}[u]) \\
&= charge^{(t)}[u] + \sum_{v \in N(u)} poids(u,v) * (charge^{(t)}[v] - charge^{(t)}[u])
\end{aligned}$$

$$= (1 - \sum_{v \in N(u)} poids(u,v)) * charge^{(t)}[u] + \sum_{v \in N(u)} poids(u,v) * charge^{(t)}[v]$$

Si on considère la matrice $P = (p_{xy})_{x,y \in V}$, définie de la façon suivante :

$$p_{xy} = \begin{cases} 1 - \sum_{z \in N(x)} poids(x,z), & \text{si } x = y \\ poids(x,y), & \text{si } y \in N(x) \\ 0, & \text{sinon.} \end{cases}$$

$charge^{(t+1)}$ peut s'exprimer comme :

$$charge^{(t+1)} = P * charge^{(t)}$$

Ceci permet donc d'avoir la relation suivante entre $charge^{(t+1)}$ et $charge^{(0)}$:

$$charge^{(t+1)} = P^{t+1} * charge^{(0)} \tag{2.3}$$

Etudes de convergence

En nombre de pas d'itération Selon l'équation (2.3), la convergence du vecteur $(charge^{(t)})$ est gouvernée par la convergence de la matrice P^t, P étant symétrique et $\sum_{y \in V} p_{xy} = 1$.

Si on prend $poids(x,y) \in [0,1), \forall (x,y) \in E$ et $1 - \sum_{y \in V} poids(x,y) > 0$, la matrice P est doublement stochastique et P^t est convergente vers la matrice $\frac{1}{n} I_n$[10], où $n = |V|$. La vitesse de convergence est donnée par $|\lambda_2|$, où $1 > |\lambda_2| > |\lambda_3| \geq \cdots \geq |\lambda_n|$ sont les valeurs propres de la matrice P, voir [Cyb89, Boi88, Boi90] pour la preuve et [Fie75] et [Gan66] pour une théorie générale.

Pour analyser le comportement de ce type de matrices quand la matrice P est connue, on peut faire appel à l'analyse de la chaîne de Markov induite par cette matrice P, voir [Ros95].

Si on choisit :

$$poids(x,y) = \frac{1}{1 + max(d_G(x), d_G(y))}, \forall (x,y) \in E$$

la convergence est en $O(n^2)$ pas d'itérations pour des graphes quelconque [Boi88, Boi90]. Il est évident que l'équilibre de charge est atteint dans un seul pas de diffusion pour le graphe complet K_n. Boillat et Kropf [Boi90, BK90] font remarquer que pour des tores $T_{n_1 \times n_2 \times \cdots n_m}$ la convergence est en $O(\max(n_1, n_2, \cdots, n_m)^2)$. Pour l'hypercube H_d la convergence en nombre de pas d'itérations est en $O(d)$ [Cyb89, Boi90].

10. I_k est la matrice unité de dimension k.

En temps total Le temps de diffusion comporte deux composantes: le temps nécessaire pour acquérir des informations sur le voisinage et le temps pour accomplir les échanges de charge. Les estimations de ce temps sont différentes selon le modèle utilisé: port unique ou multi port.

Subramanian et Scherson [SS94] ont donné des bornes inférieures et supérieures du temps total pour le modèle port unique. A chaque itération t le temps d'information reste constant et le temps pris par un processeur x pour faire les envois est:

$$\sum_{y \in N(v)} p_{xy} * |charge^{(t)}[x] - charge^{(t)}[y]|$$

Le temps total T après un nombre suffisamment grand d'itérations peut être borné par:

$$\Omega\left(\frac{\log \sigma}{\Gamma}\right) \leq T \leq O\left(\frac{N * \sigma}{\Gamma}\right)$$

et

$$\Omega\left(\frac{\log \sigma}{\Phi}\right) \leq T \leq O\left(\frac{\sigma}{\Phi^2}\right)$$

où σ est l'écart type du vecteur de charge initiale $charge^{(0)}$, Φ est la conductance de fluide du graphe G et de la matrice P et Γ est la conductance électrique[11]. Ces calculs confirment aussi le fait que la convergence de la méthode de diffusion est plus forte pour les tores à plusieurs dimensions que pour les réseaux linéaires ou tores bi-dimensionnels.

Pour le modèle multi-port le temps des envois d'un processeur x est:

$$\max_{y \in N(v)} p_{xy} * |charge^{(t)}[x] - charge^{(t)}[y]|$$

et donc on obtient pour le temps total T les mêmes bornes divisées par $\frac{1}{1 + \min_{x \in V} d_G(x)}$.

Extensions et variantes Le but de cette partie est de montrer brièvement les méthodes dérivées des méthodes de diffusion.

Toutes ces méthodes ont été conçues dans le but d'accélérer la convergence vers l'état d'équilibre ou de simplifier leur mise en œuvre. Il s'agit soit de modifier la politique des envois de charges, soit de proposer d'autres coefficients ou équations qui gouvernent la diffusion.

Méthode diffusive post-mortem Les simulations que nous avons faites [LS96] avec la méthode diffusive montrent qu'elle peut présenter deux inconvénients. Sur un même lien physique, les envois de charge se font en plusieurs étapes, d'où un certain surcoût du temps d'initialisation de chaque communication. Ainsi il est même possible qu'à une itération t l'envoi sur ce lien soit fait dans un sens et à une autre itération $t' > t$ l'envoi soit fait dans l'autre sens. Pour palier ces inconvénients qui dégradent les performances de la méthode, on propose que pendant les itérations seulement la quantité de charge que les processeurs auraient dû avoir échangée soit évaluée. Les envois seront effectivement réalisés après satisfaction du critère de convergence.

11. Voir annexe A

Les simulations que nous avons faites [AA97] ont montré que cette stratégie est bénéfique pour des topologies denses, i.e. de degré important, comme par exemple des tores en 3 dimensions, et plutôt pénalisante pour les réseaux de faible connectivité, comme les réseaux linéaires ou les grilles bidimensionnelles.

Diffusion à unique coefficient Cette méthode [XL93] consiste à prendre pour toutes les arêtes du graphe du réseau d'interconnexion un même poids :

$$poids(x, y) = \alpha \tag{2.4}$$

tel que $\alpha \in (0, 1)$ et $1 - d_G(x) * \alpha > 0$ ce qui assure la convergence de la méthode [Cyb89]. Les auteurs, Xu et Lau se posent ensuite la question de trouver la valeur de ce paramètre (paramètre de diffusion) pour le réseau particulier d'un tore de dimension k tel que la convergence soit la plus rapide possible. L'ordre de convergence reste le même que pour la méthode générale.

Ce type de diffusion est utilisé dans les applications réelles [Hor93, Koh95] avec $\alpha = 1/2$.

Méthode parabolique Cette méthode dérivée de la diffusion a été proposée par Heirich [Hei94]. Elle a un point commun avec la diffusion à unique coefficient : les quantités de charge à échanger entre voisins sont calculées avec des poids donnés par l'équation (2.4), mais dans ces calculs $charge[u]$ n'intervient plus, sa place est prise par des valeurs $ch_u^{(m)}$ calculées de manière récursive.

La méthode est toujours itérative, les communications faites à chaque itération sont plus nombreuses, car il faut un certain nombre de pas successifs d'échange des valeurs entre voisins et de calcul de nouvelles valeurs et d'un unique échange effectif de charge entre voisins. Pour un nœud u l'étape de calcul des envois de charge lors d'une itération se décrit comme suit :

$ch_u^{(0)} \leftarrow charge[u]$;
for $i = 1, m$ do
    envoi $ch_u^{(i-1)}$ aux voisins $v \in N(u)$;
    reçoit $ch_v^{(i-1)}$ de tous voisins $v \in N(u)$;
    $ch_u^{(i)} \leftarrow F(ch_u^{(0)}, ch_u^{(i-1)}, ch_{v_1}^{(i-1)}, \cdots, ch_{v_k}^{(i-1)})$;

envoi $ch_u^{(m)}$ aux voisins v;
reçoit $ch_v^{(m)}$ de tous voisins v;

Pour tout voisin $v \in N(u)$ faire
    $envoi[u, v] \leftarrow \alpha * (ch_u^{(m)} - ch_v^{(m)})$;

où $v_1, \cdots, v_k$ sont les nœuds voisins et F la relation de récurrence pour laquelle Heirich propose et analyse plusieurs formules[12].

Dans [HT95] la méthode est appliquée pour un tore en trois dimensions. La convergence a été améliorée par rapport à la méthode diffusive classique pour $m = 3$ et F fonction linéaire.

12. Le nom de parabolique est dû à la forme de la fonction F qui privilégie la valeur initiale de charge du nœud lui-même.

Méthode des envois proportionnels Dans la description de la méthode de diffusion la matrice P qui fait la transformation du vecteur $(charge[u], u \in V)$ reste immuable pendant toutes les itérations. Une idée simple apparaît: est-ce que on peut prendre une matrice $P(t, charge(t))$ adaptative par rapport à la distribution de charge?

Dans cet esprit une version de la diffusion a été proposée dans [WLR93] puis dans [LRCM95]. Après l'étape d'information chaque nœud est capable de calculer la charge moyenne du voisinage:

$$\overline{Charge_u} = \frac{1}{k+1} * \left(charge[u] + \sum_{v \in N(v)} charge[v] \right)$$

les envois de charge se font uniquement si le nœud est sur-chargé par rapport à son voisinage:

$$charge[u] > \overline{Charge_u}$$

et vers les nœuds v déficitaires:

$$charge[v] < \overline{Charge_u}$$

$$deficit[v] = \overline{Charge_u} - charge[v].$$

La quantité de charge envoyée vers un nœud est une partie de l'excédent de charge proportionnelle au déficit de ce nœud:

$$envoi[u,v] = \left(charge[u] - \overline{Charge_u}\right) * \frac{deficit[v]}{\sum_{charge[z]<\overline{Charge_u}} deficit[z]}.$$

Diffusions à envois unitaires Ces méthodes sont des cas particuliers des méthodes de diffusion, les envois de charge faits contiennent toujours une seule unité de charge.

Cette approche est due soit aux contraintes qui peuvent exister entre les unités de charge, par exemple dans un arbre des processus communicants dans [Sal90], soit à l'application concrète mise en œuvre, par exemple dans [Koh95] une simulation des particules où le domaine de simulation est décomposé en bandes et chaque bande représente en faite une unité de charge.

Luque et al. [LRCM95] proposent aussi des envois unitaires de charge dans le but d'éviter les envois "fractionnés" de charge (voir sous-section 2.4.3.4).

2.4.3.2 Méthode "échange de dimensions"

Cybenko dans [Cyb89] l'a définie de façon synthétique pour l'hypercube dans le souci d'avoir une base de comparaison avec la méthode diffusive. La méthode avait déjà été utilisée auparavant [DG89] et [FKW87] pour des applications concrètes.

Les développements qui ont suivi ont eu comme objectif de rendre la méthode applicable à toute topologie d'interconnexion. Les sous-sections 2.4.3.2, 2.4.3.2 et 2.4.3.2 présenteront ces extensions.

La méthode "échange de dimensions" pour l'hypercube Cette méthode converge en temps fini avec d pas d'itérations pour l'hypercube H_d, vers l'état d'équilibre global [Cyb89]. L'idée est d'assurer après chaque pas k que les nœuds dont les étiquettes diffèrent dans la position k ont exactement la même quantité de charge. Ceci signifie que chaque nœud u prend connaissance de la quantité de charge de son voisin de la dimension k, u_k, et réalise avec celui-ci un échange de charge de montant $envoi[u, u_k] = \frac{1}{2} * |charge[u] - charge[u_k]|$. Le code exécuté de manière synchrone par chaque nœud est le suivant :

pour tout $k = 1, d$
  soit u_k le voisin de u dans la dimension k;
  envoie à u_k $charge[u]$;
  reçoit de u_k $charge[u_k]$;
  si $charge[u] > charge[u_k]$
    envoie à u_k la quantité de charge $\frac{1}{2} * (charge[u] - charge[u_k])$;
  si $charge[u] < charge[u_k]$
    reçoit de u_k la quantité de charge $\frac{1}{2} * (charge[u_k] - charge[u])$;

Il est simple de vérifier qu'après chaque pas i, pour toute dimension k, $k \leq i$ la propriété "u a la même charge que u_k" est conservée.

Donc à la fin de l'algorithme après d pas d'itérations tous les nœuds ont la même charge égale à $\frac{1}{2^d} * \sum_{u \in H_d} charge[u]$.

Une extension pour les topologies générales Si on associe d couleurs aux arêtes de l'hypercube H_d telles que les arêtes de la dimension j $(1 \leq j \leq d)$ aient la couleur j, la méthode échange de dimension peut s'exprimer par :

pour toute couleur $j = 1, d$
  pour toute arête (u, v) de couleur j
    si $charge[u] > charge[v]$ alors $envoi[u, v] = \frac{1}{2} * (charge[u] - charge[v])$;
    si $charge[v] > charge[u]$ alors $envoi[v, u] = \frac{1}{2} * (charge[v] - charge[u])$;

Hosseini et al. proposent dans [HLM+90] une extension naturelle de cette méthode à tout graphe connexe : on trouve un d-coloriage des arêtes de G et on applique la méthode décrite ci-dessus itérativement[13]. Pour un graphe quelconque on peut toujours trouver un coloriage des arêtes en temps polynomial avec $d(G) + 1$ couleurs.

Comme pour la méthode diffusive, le vecteur de charge après $t+1$ itérations peut s'exprimer par :

$$charge^{(t+1)} = M * charge^{(t)}$$

13. Pour l'hypercube une seule itération suffit.

où M est un produit de d matrices symétriques et doublement stochastiques, la condition de convergence $\lim_{t\to\infty} M^t = \frac{1}{n} * I_n$ étant satisfaite. L'ordre de cette convergence en nombre de pas est assez grand : $O(log(L))$ où $L = \max(charge^{(0)})$.

La méthode généralisée Dans [XL92], Xu et Lau proposent la méthode "échange de dimension généralisé" (EDG) où les transferts ne sont plus calculés avec la pondération $\frac{1}{2}$, mais une pondération λ qui doit être choisie afin d'assurer une convergence plus rapide vers l'état d'équilibre. Ce paramètre s'exprime en fonction du graphe et du coloriage des arêtes. Par contre, malgré les formules qui le caractérisent, ce paramètre λ reste difficile à calculer.

Ce calcul est mené jusqu'au bout dans [XL95] par les mêmes auteurs pour les topologies en tores multi-dimensionnels. Ils comparent aussi cette méthode avec la méthode diffusive à unique coefficient, en remarquant que pour un tore de dimension n (qui se colorie avec $2n$ couleurs) une itération de la méthode EDG correspond à $2 * n$ itérations de la méthode de diffusion, et donc EDG est plus rapide.

Une version randomisée de la méthode Pour des graphes quelconques, le coloriage des arêtes est parfois lourd à calculer ($O(n * m)$ en temps, voir [NC88], chapitre 6) pour les topologies de grande taille. Ghost et Muthkrishnan dans [GM94] proposent une version randomisée de la méthode "échange de dimension". Elle consiste à faire un tirage aléatoire parmi toutes les arêtes du graphe, à éliminer ensuite certaines d'entre elles telles que l'ensemble restant soit un couplage et de réaliser l'égalisation des charges entre les nœuds directement couplés, même si le couplage déterminé n'est pas maximal.

Ceci évite de calculer a priori un coloriage pour le graphe du réseau et rend la méthode très souple vis-à-vis d'une possible défaillance d'arêtes.

2.4.3.3 Problème des envois fractionnés

Lors de la mise en œuvre de ces méthodes, on se rend compte qu'on ne peut pas les implanter telles quelles, car tandis que les calculs des envois de charge ne considèrent que des charges divisibles, les charges envoyées ne le sont pas. Il est donc impératif de trouver un moyen sûr pour réaliser les transferts de charge en tant qu'entités indivisibles.

Luque et al. [LRCM95] proposent d'envoyer la charge unité par unité, mais ils ne prouvent pas que cette stratégie aboutit à l'état d'équilibre (la méthode est celle des envois proportionnés).

Il est facile à remarquer que la stratégie simple (comme dans [Hor93]) d'envoyer toujours $\lfloor envoi[x,y] \rfloor$ (ou $\lceil envoi[x,y] \rceil$) est instable, on risque de ne jamais aboutir à un état d'équilibre. Par contre, pour l'échange de dimension randomisé [GM94], cette stratégie fonctionne bien. Ghost et Muthukrishnan prouvent qu'on aboutit toujours à un état d'équilibre local.

Une solution intuitive à ce problème des envois fractionnés consiste à choisir avec une certaine probabilité p (par exemple $p = poids(x,y)$) si on envoie $\lfloor envoi[x,y] \rfloor$ ou $\lceil envoi[x,y] \rceil$ unités de charge. Nous avons fait des simulations qui semblent bien s'orienter vers l'état d'équilibre pour une probabilité $p = envoi[x,y] - \lfloor envoi[x,y] \rfloor$ (la partie fractionnée de l'envoi de charge à faire).

On peut aussi envisager de choisir a priori pour les arêtes (x,y) du graphe une orientation et de faire les envois prenant la partie inférieure ou supérieure selon cette orientation comme dans [HLM+90]. Subramanian et Scherson remarquent [SS94] que ce choix doit se faire de manière précise pour éviter l'instabilité des transferts et ils proposent une technique de décomposition du graphe qui assure

une orientation des arêtes telle qu'elles ne forment pas des circuits et que certaines conditions de connectivité soient satisfaites.

2.4.3.4 Avantages et limitations

Le principal avantage de ces deux méthodes générales est l'extensibilité, elles peuvent s'appliquer pour toute topologie du réseau d'interconnexion tout en assurant une convergence asymptotique vers l'état d'équilibre. Un autre avantage est leur fonctionnement en mode SPMD [Boi90] sur n'importe quelle topologie réelle de la machine parallèle sans recourir à des bibliothèques spécialisées de communication, seule la communication entre des processeurs directement connectés est utilisée, d'où un degré maximum d'extensibilité [KG94, KS94] et une certaine facilité d'implantation.

La méthode "échange de dimensions" a aussi l'avantage de fonctionner pour tous les types de réseau d'interconnexion, car les nœuds ne font plus d'envois multiples.

En contre-partie, la convergence vers l'état d'équilibre est asymptotique. Ces méthodes s'avèrent exactes uniquement pour des topologies particulières, graphe complet pour la diffusion et hypercube pour l'échange de dimension.

Il y a toutefois des inconvénients liés à la mise en œuvre des envois de charge et à la mise en place des conditions d'arrêt dans les cas de convergence asymptotique. Notons aussi le problème du choix judicieux des paramètres de diffusion ou d'échange de dimensions qui garantissent la convergence (par exemple, la méthode diffusive à unique coefficient $\alpha = \frac{1}{4}$ pour un tore $2k \times 2k$ n'est pas convergente).

2.4.4 Méthodes avec informations globales

Dans la sous-section 2.4.1.1 nous avons vu une modélisation du problème qui suppose que l'information de charge des nœuds est globale et complète. Dans les machines massivement parallèles sans mémoire commune, avoir cette approche est irréaliste.

Dans la sous-section suivante nous présenterons deux solutions centralisées qui requièrent des informations globales complètes et ensuite nous ferons l'état de l'art des méthodes distribuées de régulation qui travaillent avec des information globales incomplètes.

2.4.4.1 Méthodes centralisées

Blelloch dans [Ble89], en étudiant l'implantation du calcul de préfixe sur les PRAM propose la méthode de régulation la plus naturelle : mettre toutes les unités de charge dans un vecteur contigu et puis faire le partage de celui-ci en parts égales. Si $x_1, x_2, ...x_k$ constitue la liste des unités de charge qui sont à traiter, étant donné que tous les n processeurs ont accès à cette liste, chaque processeur sera servi dans l'ordre avec une tranche de $\left\lfloor \frac{k}{n} \right\rfloor$ (+1) unités de charge. Cette méthode est difficilement envisageable pour une machine parallèle réelle pour deux raisons : premièrement on peut se heruter à une limitation mémoire sur le site qui acquiert les k unités de charge, puis, si cela était possible, ce processeur deviendrait vite un goulot d'etramglement lors de la concentration et la distribution des unités de charge.

Nicol [Nic92] reprend la même idée de compactage des unités de charge à distribuer et la méthode proposée évite les déplacements de charge pour les processeurs qui en ont déjà. Elle consiste à calculer

d'abord la valeur de la charge moyenne $\overline{charge}$, à aligner les excès des nœuds qui vérifient :

$$charge[x] > \overline{charge}$$

$$exces[x] = charge[x] - \overline{charge}$$

dans un vecteur, à aligner les déficits des nœuds :

$$charge[y] < \overline{charge}$$

$$deficit[y] = \overline{charge} - charge[y]$$

dans un autre vecteur et ensuite mettre en correspondance bijective ces deux vecteurs. Par rapport à la méthode précedente, cette méthode a le mérit de reduire la taille des envois d'unités de charge; toutefois les deux désavantages restent valables.

2.4.4.2 Méthodes distribuées

L'information concernant la charge moyenne du système $\overline{charge}$ est très importante pour toutes les méthodes de cette classe, car elle permet d'identifier les nœuds comme sur-chargé ou sous-chargé. Un autre point commun de ces méthodes est leur champ d'application : topologies qui sont ou qui contiennent des anneaux ou des réseaux linéaires. Le réseau linéaire rend possible l'obtention de la charge globale des processeurs situés à gauche d'un processeur à l'aide d'un calcul de préfixe (le même avec lequel on calcule $\overline{charge}$).

L'algorithme Râteau proposé par Fonlupt dans sa thèse [Fon94] est une méthode de régulation sur des anneaux. On calcule d'abord

$$q = \left\lfloor \frac{1}{n} \sum_{v \in V} charge[v] \right\rfloor$$

$$r = \left(\sum_{v \in V} charge[v] \right) \mathrm{mod} n.$$

Pendant $n - r$ itérations tout nœud x qui a :

$$charge[x] > q$$

envoie à son voisin de droite la quantité de charge de $charge[x] - q$ unités. Pendant les r itérations suivantes la comparaison est faite avec $q+1$ et l'envoi de charge est de $charge[x] - q - 1$. A la fin $n - r$ processeurs ont q unités de charge et r autres $q + 1$ unités.

La méthode de régulation de Gerogiannis et Orphanoudakis [GO93] travaille sur un réseau linéaire suivant le même schéma : une étape où les nœuds sur-chargés se débarrassent de leur excès de charge et une deuxième étape qui met au même niveau de charge tous les processeurs. Pour identifier les destinations des envois la méthode fait appel à la diffusion (envois d'une même information vers tous les sites) et au calcul de préfixe qui sont supposés être des routines de base de communication. Les auteurs supposent aussi que les envois de charge puissent se faire entre n'importe quels nœuds.

Pour des processeurs $i, i = 1, n$ situés sur un réseau linéaire le calcul de préfixe appliqué aux valeurs $charge[i]$ donne :

$$W[i] = \sum_{j=1}^{i-1} charge[j]$$

$$\overline{charge} = \frac{W[n]}{n}$$

Jàjà et Ryu font remarquer dans [JR92] qu'en calculant :

$$l[i] = \left\lfloor \frac{W[i]}{\overline{charge}} \right\rfloor$$

et

$$r[i] = \left\lceil \frac{W[i+1]}{\overline{charge}} \right\rceil$$

on détermine pour un processeur i les numéros des processeurs où sa charge doit être envoyée : $l[i], l[i]+1, \cdots, r[i]-1, r[i]$, si $charge[i] > 0$. Les processeurs $l[i]$ et $r[i]$ reçoivent $(l[i]+1) * \overline{charge} - W[i]$ et, respectivement, $W[i+1] - r[i] * \overline{charge}$ et les autres processeurs de $l[i]+1$ à $r[i]-1$ reçoivent $\overline{charge}$ unités de charge.

La mise en œuvre de ces échanges est différente selon les suppositions sur la topologie de départ et les routines de communication. Pour l'hypercube, Jàjà et Ryu [JR92] proposent d'envoyer d'abord ces quantités de charge dans les nœuds intermédiaires et ensuite à leur destination afin d'éviter l'encombrement des liens de communication. Pour le réseau linéaire, Mehortra et al. [MRW93] appliquent les envois de charge tels qu'ils ont été calculés et ils donnent une estimation en temps de cette politique de transfert.

Miguet dans [Mig92] s'attaque aussi à la distribution sur un réseau linéaire, mais sous la restriction que les unités de charge (des données) respectent une condition de voisinage selon un réseau linéaire et les données voisines doivent être placées soit sur le même processeur soit sur deux processeurs voisins. Cette méthode est reprise dans des nombreuses applications telles que la machine dictionnaire [Gas93] ou la distribution des cartes trapèze [Dub96].

2.4.4.3 Avantages et limitations

Le principal avantage des méthodes de cette classe est qu'elles sont exactes, en assurant en temps fini l'état d'équilibre global et aussi qu'elles ne prennent en compte que des charges discrètes dès le départ (ou elles sont facilement adaptables). Le degré de synchronisation est plus faible que pour les méthodes diffusives et "échange de dimensions", à l'exception des algorithmes à information complète et de l'algorithme Râteau. Ces avantages rendent l'implantation de ces méthodes plus aisée; par exemple il n'y a plus à se soucier de conditions d'arrêt (de convergence) incontournables pour les méthodes à information locale.

Sous la condition d'avoir des routines efficaces de communication, les performances de ces méthodes sont bonnes.

Les méthodes à information globale imposent une topologie réseau linéaire recouvrante ce qui n'est pas toujours possible, pour les arbres, par exemple. Le calcul de préfixe exigé par ces méthodes demande lors de sa mise en œuvre $O(n)$ étapes de communication (sauf pour l'hypercube), ce qui devient extrêmement coûteux pour les topologies de très grande taille.

2.4.5 Conclusions

Nous avons présenté un état de l'art des stratégies de régulation pour les charges régulières. Ces stratégies sont très variées, et pourtant on ne peut pas dire qu'il y ait une méthode ou une classe de méthodes meilleure qu'une autre, car elles sont exactes pour des topologies et/ou communications particulières et approchées dans le cas général.

Les méthodes à informations locales fournissent rarement des résultats exacts, mais elles sont applicables à tout réseau d'interconnexion des processeurs. Les méthodes à informations globales ont l'avantage de fournir de telles solutions exactes, mais soit le calcul est complétement centralisé, soit le réseau d'interconnexion est particulier, soit des routines spéciales de communication sont demandées.

La méthode que nous allons décrire dans le chapitre suivant se situe dans cette classe de stratégies d'équilibrage de charge régulières. Plus précisément, elle sera une méthode exacte à information globale et indépendante de la topologie d'interconnexion des processeurs. Par ailleurs elle n'exige pas de fonctions de communication particulières autres que que les communications point-à-point entre processeurs physiquement voisins.

Chapitre 3

Allocation dynamique des charges régulières par calcul de préfixe généralisé

Le but de ce chapitre est de proposer un algorithme efficace de régulation de charges régulières. Nous avons vu lors du chapitre précédent que la résolution optimale du problème d'allocation (y compris ce type de régulation) est un problème extrêmement difficile. Nous avons donc préféré mettre en place une heuristique de régulation qui soit correcte, qui ait l'avantage de travailler pour tout réseau d'interconnexion des processeurs et qui soit facile à mettre en œuvre. La stratégie proposée est basée sur un calcul de préfixe généralisé sur un arbre recouvrant de profondeur minimale et elle assure la convergence en temps fini dans l'état d'équilibre global.

Pour avoir un algorithme de régulation efficace, il faut s'assurer que le temps pris par la régulation sera compensé par le gain de temps obtenu suite à la régulation. Il faut donc avoir une idée précise du coût en temps de la régulation. Dans ce but, nous avons développé un modèle probabiliste et une analyse statistique des simulations faites sur ce modèle qui nous ont permis de trouver la loi de probabilité qui modélise le temps maximal de régulation. L'efficacité de l'algorithme de régulation s'appuie sur une politique d'initiation et une politique de décision inférées à partir de ce résultat probabiliste.

Ce travail a fait l'objet de plusiers publication dont [JS96] sur la stratégie de régulation et [JCS96] sur son analyse en complexité.

3.1 Notations et définitions

Nous allons garder les mêmes notations que dans la partie 2.4 de l'état de l'art des stratégies de régulation de charges régulières: $G = (V, E)$ pour le graphe du réseau d'interconnexion, $charge[v]$ la quantité de charge du nœud $v \in V$.

Nous faisons une extension de la fonction *charge* pour tout sous-graphe $H = (V', E')$ de G:

$$charge(H) = \sum_{v \in V'} charge[v]$$

Si $T = (V, E')$ est un arbre recouvrant de G, nous pouvons mettre tous les nœuds de G sur des niveaux qui indiquent la distance des nœuds par rapport à la racine. Pour tout nœud $v \in V$, nous adopterons les notations et les définitions suivantes :

- $v^{(0)}$ pour le prédécesseur du nœud v;
- T_v pour le sous-arbre de T enraciné en v; si v est une feuille de T, T_v est alors réduit au nœud v;
- $taille(T_v)$ pour le nombre de nœuds du sous-arbre T_v.

Soit l'ensemble $\{x_1, x_2, ...x_n\}$ muni d'une relation d'ordre linéaire : $x_1 \prec x_2 \prec ... \prec x_n$. Soient $\{val(x_1), val(x_2), ...val(x_n)\}$ un ensemble de valeurs associé à l'ensemble de départ et $*$ une opération binaire entre ces valeurs. Nous appelons préfixe d'ordre i la valeur :

$$P(i) = val(x_1) * val(x_2) * ... * val(x_i)$$

Par ailleurs $P(i)$ peut s'exprimer comme :

$$P(i) = *_{x_k \preceq x_i} val(x_k)$$

La notion de préfixe se généralise facilement pour toute relation d'ordre $\prec$. Le préfixe généralisé d'un élément x se définit comme :

$$P(x) = *_{y \preceq x} val(y)$$

Si on considère un arbre T avec une racine r, cette structure induit une relation d'ordre. Alors dans le cadre du problème de régulation $val(x) = charge[x]$, le préfixe généralisé calculé dans un nœud x :

$$P(x) = \sum_{y \preceq x} charge[y]$$

correspond à la charge à la charge du sous-arbre T_x enraciné dans x. Egalement $P(r) = \sum_{x \preceq r} charge(x)$ correspond à la charge totale de l'arbre T.

Si $val(x) = 1$, pour tout nœud x de l'arbre, le préfixe généralisé indique la taille du sous-arbre T_x.

3.2 Principe de l'allocation dynamique par calcul de préfixe généralisé

Nous avons vu dans l'état de l'art des stratégies de régulation une classe entière de méthodes dédiées aux réseaux linéaires et basées sur une connaissance globale de l'état de charge : la valeur de la charge moyenne $\overline{charge}$. Le fait que le réseau de base soit linéaire permettait de connaître par un calcul de préfixe la charge d'un sous-réseau et en la comparant avec la taille du sous-réseau et $\overline{charge}$, on arrivait à déterminer les échanges de charge à effectuer pour aboutir à l'état d'équilibre.

Utiliser un sous réseau qui soit linéaire pour faire de la régulation de charges peut s'avérer inefficace pour les réseaux de grande taille (car il faut, par exemple, $O(n)$ pas de communication pour calculer $\overline{charge}$, n étant la taille du réseau), d'une part, et pour les réseaux qui n'admettent pas de sous-réseaux linéaires recouvrants, comme les arbres, d'autre part.

Pour pallier ces désavantages, nous proposons une généralisation de cette méthode qui soit applicable à tout réseau d'interconnexion, sans demander de fonctions particulières de communication, et qui améliore sensiblement le temps de régulation pour les réseaux de grande taille. Cette stratégie est basée sur un calcul de préfixe généralisé et utilise comme structure de communication un arbre recouvrant de profondeur minimale.

L'arbre recouvrant construit, notons-le T, permet de définir un ordre partiel entre les nœuds du graphe et, ainsi, d'obtenir, par le calcul de préfixe généralisé selon cet ordre, la charge des sous-arbres de T et leur taille et par la suite la charge de chaque sous-arbre dans un état d'équilibre. Le principe de la régulation de charge est alors de procéder à des échanges de charge entre nœuds de l'arbre de sorte que chaque sous-arbre ait une charge égale à sa charge dans l'état d'équilibre.

3.3 Stratégie de régulation pour des architectures homogènes

La stratégie de régulation de charge qui sera présentée a été d'abord conçue pour des machines parallèles homogènes. Elle sera ensuite étendue dans la sous-section 3.4 aux cas des machines hétérogènes et servira de base pour un algorithme complet de régulation de charge, dans la sous-section 3.6.

3.3.1 Description

La stratégie de régulation que nous proposons est distribuée et censée travailler sur une architecture homogène. Le but de la régulation est donc d'aboutir à un état d'équilibre de charge, c'est à dire un état où tous les processeurs ont à une unité près le même nombre d'unités de charge : *charge*. Elle consiste en trois phases :

Phase 1 : Calcul d'un arbre recouvrant de profondeur minimale;
Phase 2 : Collecte d'informations sur l'état de charge par un calcul de préfixe généralisé selon l'ordre induit par l'arbre recouvrant de la phase 1;
Calcul des échanges de charges;
Phase 3 : Echanges effectifs des charges.

La première phase de l'algorithme calcule une fois pour toutes à partir d'un centre v^* de G un arbre recouvrant par un parcours en largeur d'abord. Nous ne développerons pas ici un tel calcul qui est très amplement décrit dans la littérature : voir par exemple, [KRS84] pour le calcul distribué du centre d'un graphe, [GHS83] pour le calcul d'un arbre recouvrant de coût minimum, [Lav95] pour le calcul simultané du centre et de l'arbre recouvrant. L'idée d'utiliser un arbre recouvrant pour réaliser différents calculs comme, par exemple, un tri par fusion est présente dans la littérature [KK93], mais Kale et Krishnan remontent toute l'information au niveau de la racine de l'arbre pour la traiter. Dans la cas de la régulation, on pourrait penser pareillement à une stratégie qui concentre toutes les charges au niveau de la racine pour les distribuer facilement ensuite, mais ce n'est pas envisageable pour les topologies de grande taille et/ou avec un nombre important d'unités de charge, car la racine deviendrait sûrement un goulot d'étranglement.

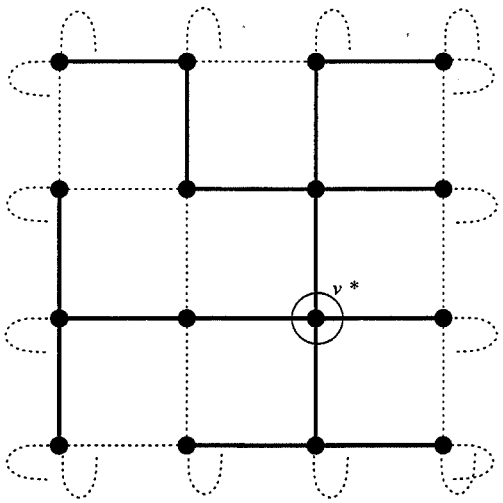


FIG. 3.1 – *Un arbre recouvrant de la grille torique* 4×4.

Soit alors $T = (V, E')$ cet arbre recouvrant, enraciné en v^*. La figure 3.1 donne un tel arbre pour la grille torique 4×4; à noter que, dans ce cas, n'importe quel nœud peut être choisi comme centre car les nœuds sont indifférenciés.

La deuxième phase commence par le calcul de préfixe généralisé initialisé en chaque nœud. A la suite d'un tel calcul, chaque nœud v connaît $charge(T_v)$ et $taille(T_v)$, $charge(T_u)$ et $taille(T_u)$ pour tout nœud fils u. En particulier, la racine v^* est à même d'évaluer la charge moyenne $\overline{charge}$ de tout le réseau et de la diffuser vers tous les nœuds.

L'étape suivante consiste, pour tout nœud v à déterminer, par rapport à la charge moyenne, l'état de la charge de T_v, ainsi que T_u pour chacun de ses fils u.

Un des trois cas suivants peut apparaître :

1. $charge(T_v) > taille(T_v) * \overline{charge}$, alors T_v est surchargé. Il doit se délester de son excédent de charges en envoyant celui-ci à l'extérieur du sous-arbre, c'est à dire à $v^{(0)}$ à travers la racine v.
   Par exemple, pour la distribution des charges indiquée dans la figure 3.2, le sous-arbre enraciné dans $(3,3)$ est surchargé, il va envoyer son excédent (98) à son prédécesseur le nœud $(2,3)$.

2. $charge(T_v) < taille(T_v) * \overline{charge}$, alors T_v est sous–chargé. Il doit combler son déficit grâce à un sous-arbre excédentaire par le biais de $v^{(0)}$.
   Dans la même configuration, le nœud $(2,2)$ est sous-chargé, il recevra son déficit de charge par le biais du nœud $(2,3)$.

3. $charge(T_v) = taille(T_v) * \overline{charge}$, alors T_v est globalement équilibré. La régulation de la charge de ses nœuds est une opération interne à ce sous-arbre. Cette opération est récursivement réalisée selon l'état de la charge de chaque sous-arbre enraciné dans les nœuds fils comme décrit dans les deux premiers cas.

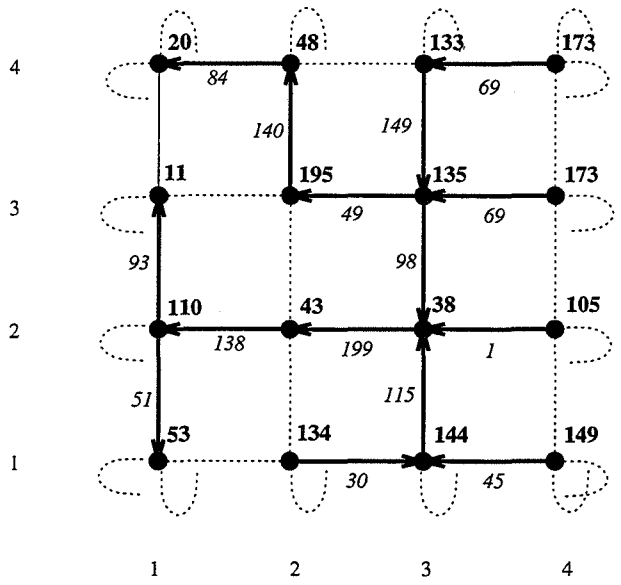


FIG. 3.2 – *Schéma de distribution de charges dans une grille torique* 4×4. *La distribution initiale de la charge est représentée par les nombres en gras* ($\overline{charge} = 104$). *Les nombres en italique sur les arcs indiquent la quantité de données que le nœud origine doit envoyer au nœud extrémité.*

En fait les deux premiers cas orientent chaque arête de T_{v^*} et lui attachent la quantité de données que le nœud origine devra envoyer au nœud extrémité. Chaque nœud dispose ainsi d'un ensemble de nœuds qui émettent vers lui et d'un ensemble de nœuds vers lesquels il émet. La figure 3.2 donne l'illustration de tous ces échanges pour l'arbre recouvrant de la figure 3.1 extrait de la grille torique 4×4.

Une question importante est alors comment réaliser correctement ces échanges de charges sous la contrainte naturelle qu'un nœud v ne peut envoyer plus que la charge qu'il détient. Plusieurs protocoles peuvent être utilisés, par exemple :

- en commençant par les feuilles de l'arbre qui sont surchargées ou par des nœuds qui ont l'ensemble des voisins émetteurs nul;
- en commençant par n'importe quel nœud;
- on peut envoyer de la charge unité par unité ou des paquets de plusieurs unités
- s'il y a plusieurs récepteurs, on peut choisir en premier celui de déficit minimal ou maximal.

Le protocole que nous proposons est basé sur le principe suivant : la charge qu'un nœud v enverra à un voisin u devra délester T_v de sa surcharge, si $u = v^{(0)}$, ou combler le déficit de T_u, si u est un fils. Autrement dit, l'échange de données entre deux voisins se fait en une unique communication.

Ceci se traduit par le fait que tout nœud émetteur qui détient une charge quelconque doit d'abord s'assurer que celle-ci couvre au moins les besoins d'un de ses voisins récepteurs. Dans la négative il se

met en attente de données de la part de ses voisins émetteurs avant d'essayer à nouveau de couvrir les besoins d'un voisin récepteur.

Pour décrire plus formellement l'activité d'un nœud v pendant le processus de régulation de charges, soit S, (respectivement R) l'ensemble de ses voisins qui émettent vers lui (respectivement vers lesquels il émet) et soit $echange(x, y)$ le montant de la charge que x doit envoyer à y. L'algorithme s'écrit comme suit :

/* 1-ère phase */
Calculer l'arbre recouvrant T_{v^*} enraciné au centre v^* du graphe d'interconnexion.

/* 2-ème phase : calcul de la charge moyenne $\overline{charge}$ */
/* à l'aide du calcul de préfixe généralisé sur T_{v^*} */
(1) $taille(T_v) \leftarrow 1$;
$charge(T_v) \leftarrow charge(v)$;
(2) pour tout u fils de v
Recevoir $taille(T_u)$ et $charge(T_u)$ de u;
$taille(T_v) \leftarrow taille(T_v) + taille(T_u)$;
$charge(T_v) \leftarrow charge(T_v) + charge(T_u)$;
(3) Envoyer $taille(T_v)$ et $charge(T_v)$ à $v^{(0)}$;
/* 2-ème phase : diffusion $\overline{charge}$ */
(4) Si $v \neq v^*$ Recevoir $\overline{charge}$ de $v^{(0)}$;
(5) pour tout u fils de v
Envoyer $\overline{charge}$ à u;
/* 2-ème phase : calcul des échanges de charges */
(6) $R, S \leftarrow \emptyset$;
(7) Si $charge(T_v) > taille(T_v) * \overline{charge}$ alors
$R \leftarrow R \cup \{v^{(0)}\}$;
$echange(v, v^{(0)}) \leftarrow charge(T_v) - taille(T_v) * \overline{charge}$;
Si $charge(T_v) < taille(T_v) * \overline{charge}$ alors
$S \leftarrow S \cup \{v^{(0)}\}$;
$echange(v^{(0)}, v) \leftarrow taille(T_v) * \overline{charge} - charge(T_v)$;
(8) pour tout u fils de v
Si $charge(T_u) > taille(T_u) * \overline{charge}$ alors
$S \leftarrow S \cup \{u\}$;
$echange(u, v) \leftarrow charge(T_u) - taille(T_u) * \overline{charge}$;
Si $charge(T_u) < taille(T_u) * \overline{charge}$ alors
$R \leftarrow R \cup \{u\}$;
$echange(v, u) \leftarrow taille(T_u) * \overline{charge} - charge(T_u)$;

/* 3-ème phase : échanges de charges entre voisins */
(9) Tant que $R, S \neq \emptyset$ faire
Tant que $R \neq \emptyset$ et $\exists x \in R : charge(v) \geq echange(v, x)$
Envoyer la quantité $echange(v, x)$ de charges à x;
$charge(v) \leftarrow charge(v) - echange(v, x)$;
$R \leftarrow R - \{x\}$;
Si $S \neq \emptyset$ alors
Recevoir la quantité $echange(x, v)$ de charges de x;
$charge(v) \leftarrow charge(v) + echange(x, v)$;
$S \leftarrow S - \{x\}$;

3.3.2 Preuve de correction

Nous avons vu précédemment que la politique des envois de charge était valide. Le fait que le sous-réseau choisi comme support de communication soit un arbre, donc sans cycles, exclut toute possibilité d'interblocage relatif au séquencement de ces envois.

Dans cette sous-section nous allons prouver que la stratégie présentée est correcte et exacte, c'est à dire qu'à la fin tous les processeurs ont la même quantité de travail : $\overline{charge}$.

Considérons le digraphe $\overrightarrow{T_{v^*}}$ construit à partir de T_{v^*} en orientant les arêtes selon le sens des échanges de données entre nœuds voisins; pour tout triplet (v, R, S) et pour tout nœud x l'arête définie par v et x est orientée de v vers x ou de x vers v, selon que, respectivement, $x \in R$ ou $x \in S$. Pour tout sommet v tel que T_v est excédentaire, on a, d'après les points (7) et (8) de l'algorithme :

$$R = \{v^{(0)}\} \cup \{u : T_u \text{ est déficitaire}\}$$
$$S = \{u : T_u \text{ est excédentaire}\}.$$

Considérons alors la grandeur

$$\sum_{u \in S} echange(u, v) - \sum_{u \in R} echange(v, u).$$

Par définition de R on a :

$$\sum_{u \in R} echange(v, u) = (charge(T_v) - taille(T_v) * \overline{charge}) + \sum_{u:T_u \text{déficitaire}} (taille(T_u) * \overline{charge} - charge(T_u))$$

avec $charge(T_v) = charge(v) + \sum_{u:T_u \text{déficitaire}} charge(u) + \sum_{u:T_u \text{excédentaire}} charge(u)$, et

$$\sum_{u \in S} echange(u, v) = \sum_{u:T_u \text{excédentaire}} (charge(T_u) - taille(T_u) * \overline{charge})$$

Tous calculs faits on obtient :

$$\sum_{u \in S} echange(u, v) - \sum_{u \in R} echange(v, u) = \overline{charge} * (taille(T_v) - \sum_u taille(T_u)) - charge(v).$$

Mais comme $taille(T_v) = 1 + \sum_u taille(T_u)$ il s'ensuit que :

$$charge(v) + \sum_{x \in S} echange(u, v) - \sum_{u \in R} echange(v, u) = \overline{charge}.$$

C'est à dire qu'après que v ait réalisé ses échanges avec ses voisins (père et fils) sa charge courante est égale à la charge moyenne. Par le même raisonnement, on établit un résultat identique pour tout sommet v tel que T_v est déficitaire.

3.4 Stratégie de régulation pour des architectures hétérogènes

La stratégie présentée dans la précédente section fonctionne pour les machines parallèles homogènes, où tous les processeurs sont supposés avoir la même puissance de calcul. Nous allons adapter celle-ci pour les machines hétérogènes dont les processeurs ont une puissance variable de calcul. Cette solution se prête aussi à des systèmes distribués [BF96].

3.4.1 Objectif de la régulation sur architectures hétérogènes

Nous gardons toujours les mêmes notations qu'auparavant et nous introduisons le concept de vitesse afin d'exprimer l'hétérogénéité de la machine. On appellera *vitesse* l'application $vitesse : V \rightarrow I\!R$, qui à chaque nœud v associe $vitesse[v]$ égale à la vitesse relative de calcul du processeur v, c'est à dire que $\forall v_1, v_2 \in V$ toute quantité de charge sera exécutée $\frac{vitesse[v_1]}{vitesse[v2]}$ plus vite sur $v1$ que sur v_2.

Rappelons que $charge(G) = \sum_{v \in V} charge[v]$ représente le nombre total d'unités de charge qui existent dans le système au moment du déclenchement de la régulation de charge.

L'objectif de la régulation de charge, dans ce cas, est d'assurer à chaque processeur une quantité de travail (nombre d'unités de charge) telle que le temps d'exécution soit le même pour tous les processeurs.

Soit $final[v]$ cette charge pour $v \in V$. La condition que les processeurs ont le même temps d'occupation signifie :

$$\frac{final[v]}{vitesse[v]} = T, \forall v \in V \tag{3.1}$$

telle que la conservation de la charge du système soit garantie, c'est à dire :

$$\sum_{v \in V} final[v] = charge(G) \tag{3.2}$$

L'équation (3.1) s'exprime :

$$final[v] = T * vitesse[v], \forall v \in V$$

En faisant la somme de toutes ces relations on obtient :

$$\sum_{v \in v} final[v] = T * \sum_{v \in v} vitesse[v]$$

Utilisons maintenant l'équation de conservation des charges (3.2) :

$$charge(G) = T * \sum_{v \in V} vitesse[v]$$

d'où :

$$T = \frac{charge(G)}{\sum_{v \in V} vitesse[v]}$$

et donc :

$$final[v] = \frac{vitesse[v]}{\sum_{u \in V} vitesse[u]} * charge(G) \tag{3.3}$$

On peut noter pour tout sous-graphe H de G :

$$vitesse(H) = \sum_{x \in H} vitesse[x]$$

3.4.2 Stratégie adaptée de régulation de charge pour architectures hétérogènes

Remarquons d'abord que $vitesse(T_x)$ demande aussi un calcul de préfixe généralisé, tout comme $charge(T_x)$.

Soit x un nœud quelconque. Après équilibrage la charge globale du T_x sera :

$$\sum_{v \in T_x} final[v] = \sum_{v \in T_x} vitesse[v] * \frac{charge(G)}{vitesse(G)}$$

Les comparaisons se font alors entre $vitesse(T_x) * \frac{charge(G)}{vitesse(G)}$ et $charge(T_x)$:

(1) Si $vitesse(T_x) * \frac{charge(G)}{vitesse(G)} = charge(T_x)$, alors le sous-arbre T_x est équilibré, les envois de charge vont se faire uniquement à l'intérieur de ce sous-arbre;

(2) Si $vitesse(T_x) * \frac{charge(G)}{vitesse(G)} > charge(T_x)$, alors le sous-arbre T_x est sous-équilibré, et donc il a besoin d'un apport de charge depuis l'extérieur égale à la différence de ces valeurs;

(3) Si $vitesse(T_x) * \frac{charge(G)}{vitesse(G)} < charge(T_x)$, alors le sous-arbre T_x est surchargé, il va envoyer son excédent de charge par le biais de l'arc $(x, x^{(0)})$.

Le code de cette stratégie est très semblable au code de la stratégie pour milieux homogènes, la première et la dernière phases restent identiques. Tout nœud v exécute lors de la phase de calcul des envois :

/* 2-ème phase : calcul de la valeur $\frac{charge(G)}{vitesse(G)}$ */
/* à l'aide du calcul de préfixe généralisé sur T_{v^*}*/
(1) $vitesse(T_v) \leftarrow vitesse[v]$;
$charge(T_v) \leftarrow charge(v)$;
(2) pour tout u fils de v
Recevoir $vitesse(T_u)$ et $charge(T_u)$ de u;
$vitesse(T_v) \leftarrow vitesse(T_v) + vitesse(T_u)$;
$charge(T_v) \leftarrow charge(T_v) + charge(T_u)$;
(3) Envoyer $vitesse(T_v)$ et $charge(T_v)$ à $v^{(0)}$;
/* 2-ème phase : diffusion $\frac{charge(G)}{vitesse(G)}$ */
(4) Si $v \neq v^*$ Recevoir $\frac{charge(G)}{vitesse(G)}$ de $v^{(0)}$;
(5) pour tout u fils de v
Envoyer $\frac{charge(G)}{vitesse(G)}$ à u;
/* 2-ème phase : calcul des échanges de charges */
(6) $R, S \leftarrow \emptyset$;
(7) Si $charge(T_v) > vitesse(T_v) * \frac{charge(G)}{vitesse(G)}$ alors
$R \leftarrow R \cup \{v^{(0)}\}$;
$echange(v, v^{(0)}) \leftarrow charge(T_v) - vitesse(T_v) * \frac{charge(G)}{vitesse(G)}$;
Si $charge(T_v) < vitesse(T_v) * \frac{charge(G)}{vitesse(G)}$ alors
$S \leftarrow S \cup \{v^{(0)}\}$;
$echange(v^{(0)}, v) \leftarrow vitesse(T_v) * \frac{charge(G)}{vitesse(G)} - charge(T_v)$;
(8) pour tout u fils de v
Si $charge(T_u) > vitesse(T_u) * \frac{charge(G)}{vitesse(G)}$ alors
$S \leftarrow S \cup \{u\}$;
$echange(u, v) \leftarrow charge(T_u) - vitesse(T_u) * \frac{charge(G)}{vitesse(G)}$;
Si $charge(T_u) < vitesse(T_u) * \frac{charge}{vitesse(G)}$ alors
$R \leftarrow R \cup \{u\}$;
$echange(v, u) \leftarrow vitesse(T_u) * \frac{charge(G)}{vitesse(G)} - charge(T_u)$;

3.5 Analyse de complexité

Cette analyse concerne essentiellement la troisième phase, la plus coûteuse des trois que compose la stratégie. Etant donné que la stratégie proposée fonctionne en mode distribué, nous allons nous intéresser à sa complexité en temps, nombre total et taille des messages. Cette analyse sera faite uniquement pour les architectures homogènes. Dans le cas hétérogène, l'analyse est identique. Nous faisons dans une première partie le point sur les résultats d'une implantation de cette stratégie de régulation et ensuite une analyse "classique" de complexité, mais celle-ci va s'avérer insuffisante pour avoir des estimations assez fines de la complexité en temps. Pour pallier cette insuffisance, nous procéderons à une analyse probabiliste de ce temps dans la sous-section 3.5.3.

3.5.1 Implantation

Nous avons implanté cette stratégie comme un noyau de base de régulation. Ce noyau sera englobé dans un allocateur dynamique complet et il est effectivement utilisé pour l'application de simulation numérique de recristallisation dynamique continue (voir la partie 4.3.5).

L'implantation a été faite sur une VOLVOX-TS306, machine MIMD homogène reconfigurable, à mémoire distribuée, disposant de 24 processeurs de travail de type transputer T805. Le language utilisé est C-Inmos, le code est parfaitement extensible à n'importe quelle taille de machine. Aussi, un effort de portage sur un autre type de machine serait minime, car seules les routines de communication sont à adapter.

A cause du nombre des processeurs et de la limite à 4 des liens physiques de communication, tous les tores considérés sont bidimensionnels et de taille 3×3, 3×4, 4×4, 4×5 ou 4×6.

La limitation à 4 du nombre de liens de communication d'un processeur rend impossible la connexion directe entre la racine de l'arbre, qui est de degré 4, et le processeur hôte de la VOLVOX. Nous avons donc dû insérer le processeur hôte entre deux processeurs de travail pour pouvoir construire des grilles toriques bidimensionnelles, comme indiqué dans la figure 3.3.

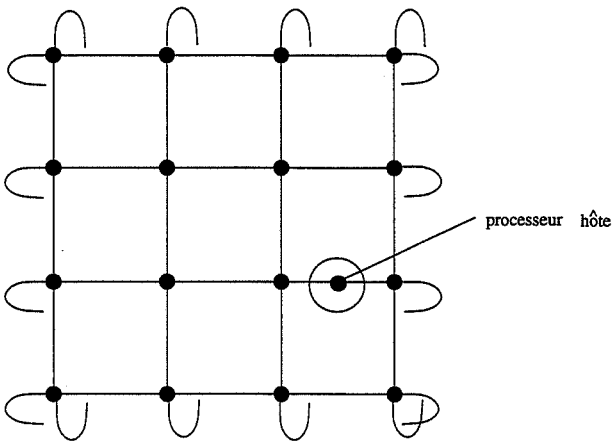


FIG. 3.3 – *Configuration de la VOLVOX en grille torique bidimensionnelle de taille* 4×4.

La charge de chaque processeur a été aléatoirement générée selon une loi de distribution uniforme $\mathcal{U}[0, x[$. Donc, les résultats obtenus dépendent de trois paramètres: la borne supérieure x de l'intervalle de distribution, la taille d'une unité de données et la dimension de la grille torique.

Il est évident que le temps des étapes (1) - (8) de l'algorithme (la deuxième phase) ne dépend que de la profondeur de l'arbre. La figure 3.4 donne le temps moyen de ces étapes pour un échantillon de 12 tests.

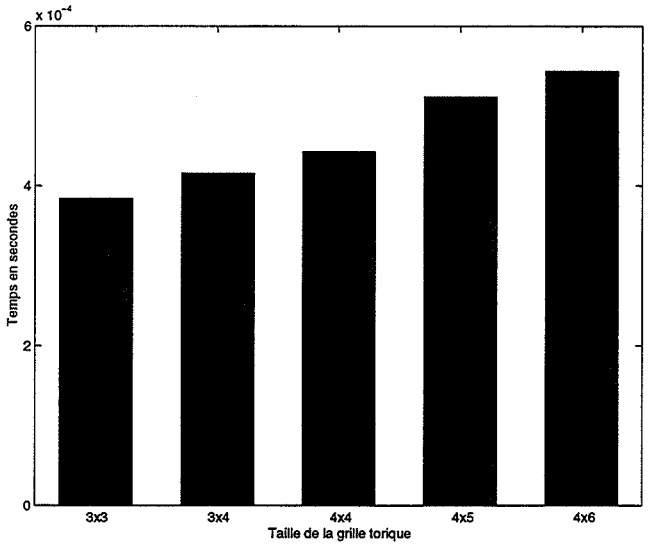


FIG. 3.4 – *Le temps nécessaire pour la deuxième phase (il ne dépend qûe de la topologie).*

Le temps du pas (9), dédié aux échanges de charges, est fortement dépendant de la taille des données à échanger, c'est à dire la taille de l'unité de charge et la taille des paquets de données. La figure 3.5 donne la durée moyenne de cette étape pour un échantillon de 12 tests pour une loi de distribution $\mathcal{U}[0, 200[$ et pour des données de taille unitaire 1, 10 et 100 bytes.

La figure 3.6 donne le temps de répartition obtenu en faisant varier l' intervalle de génération des charges pour des charges de taille unitaire égale à 32 bytes.

3.5.2 Analyse classique de complexité

La première phase est dominée par le calcul du centre v^* du graphe et par le calcul d'un arbre recouvrant minimal de racine v^*. On peut, comme indiqué précédemment, utiliser pour ces calculs un unique algorithme distribué comme celui décrit dans [Lav95] et dont la complexité est $O(n^2 \log n)$ en temps et $O(n^2 \log n)$ en nombre de messages de taille $O(\log n)$ bits. Il faut toutefois noter que quand bien même l'allocateur peut être sollicité plusieurs fois lors d'une même application, ces calculs ne sont effectués qu'une seule fois.

La deuxième phase est plutôt dominée par la collecte de l'information sur la distribution initiale de la charge du réseau. En raison de son caractère distribué (vague d'informations remontant depuis les feuilles jusqu'à la racine de l'arbre recouvrant et inversement), son coût est $O(h)$ en temps et $2*(n-1)$

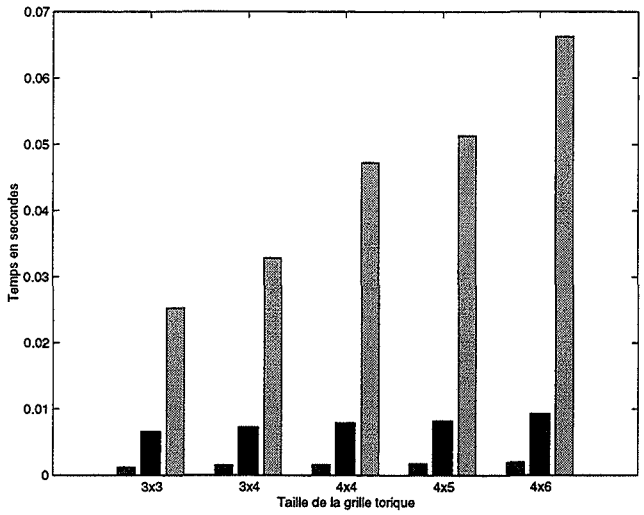


FIG. 3.5 – *Variation du temps effectif de répartition en fonction de la taille des unités de charges (1, 10, 100 bytes) pour une distribution de charges* $\mathcal{U}[0, 200[$.

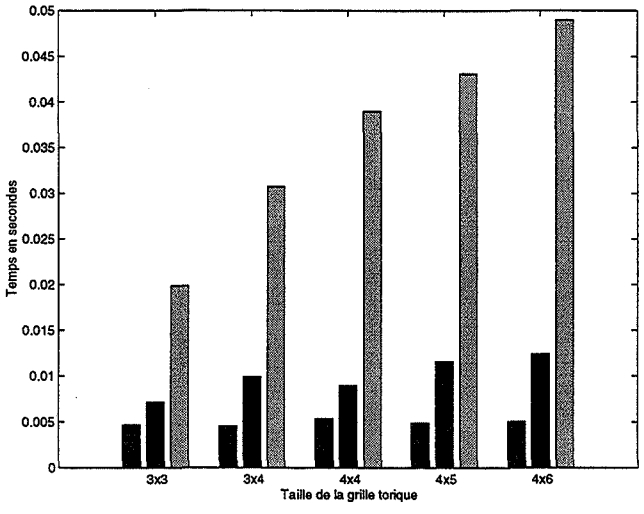


FIG. 3.6 – *Variation du temps de répartition en fonction de l'intervalle de distribution (*$\mathcal{U}[0, 10[$, $\mathcal{U}[0, 100[$, $\mathcal{U}[0, 500[$*) des charges de données de taille unitaire égale à 32 bytes.*

en nombre de messages qui sont de taille constante, h étant la profondeur de l'arbre. Les calculs qui se font sont des préfixes généralisés selon l'ordre induit par l'arbre et sont en $O(h * k)$ temps, donc négligeable par rapport aux communications, k étant le degré maximal du graphe.

La troisième phase exige au plus $n-1$ messages qui sont de taille très variable. Soit τ le temps de transfert d'une unité de charge. Le plus grand temps d'exécution de cette phase est atteint dans le cas extrême où toute la charge est concentrée en un seul nœud situé à la plus grande distance par rapport à la racine. Si on connaissait la taille moyenne d'un envoi sur ce chemin, notons-le L, on pourrait exprimer le coût de cette phase en $O(\tau * h * L)$. Mais parce que dépendant de beaucoup de paramètres dont certains non quantifiables, L n'est pas toujours facile à déterminer.

Considérons le cas où les nœuds doivent attendre les réceptions de la part de leur voisins avant d'entamer les envois. Ainsi si nous désignons par $date(v)$ le temps attendu jusqu'à finir dans l'état d'équilibre, on a:

$$date(v) = \begin{cases} 0 & \text{si } S = \emptyset \\ \tau * \max\limits_{u \in S}(date(u) + echange(u,v)) & \text{sinon.} \end{cases}$$

En considérant $echange(x,y)$ comme le poids de l'arc (x,y), il suit que la date t^* à laquelle tous les sommets ont leur charge équilibrée

$$t^* = \max_{v \in V} date(v)$$

est au plus égale à τ multiplié par le poids d'un chemin de poids maximum du digraphe $\overrightarrow{T_{v^*}}$, d'origine u telle que $S = \emptyset$ et d'extrémité v telle que $R = \emptyset$. La figure 3.7 donne une illustration d'un tel chemin pour le digraphe de la figure 3.2.

Soit alors P^* un tel chemin, $\|P^*\|$ sa longueur, et $K(P^*) = \max\limits_{(x,y) \in P^*} echange(x,y)$. On a:

$$t^* \leq \sum_{(x,y) \in P^*} echange(x,y) \leq \sum_{(x,y) \in P^*} K(P^*) = \|P^*\| * K(P^*)$$

d'où:

$$t^* = O(\tau * \min_{P^*}(\|P^*\| * K(P^*))).$$

Cette formule de complexité est plus fine que celle donnée auparavant, mais c'est une formule qu'on ne peut évaluer qu'a posteriori, après avoir calculé les échanges de charge. Elle est donc inexploitable dans le cadre d'une régulation de charge efficace. D'où la nécessité d'une autre analyse qui permettrait une analyse a priori du coût de la régulation. Nous proposons une analyse probabiliste.

3.5.3 Analyse probabiliste

Le but de cette analyse probabiliste est de déterminer une borne T_p pour pouvoir exprimer qu'avec une certaine probabilité p le temps de régulation ne dépassera pas T_p.

3.5.3.1 Modélisation

Nous allons construire un modèle du temps de régulation en terme de variables aléatoires en exprimant le poids maximum, notons-le W^*, des chemins de l'arbre orienté $\vec{T}_{v^*}$. Le temps qu'on cherche à analyser est un produit du temps de transfert d'une unité de charge τ et de ce poids maximum.

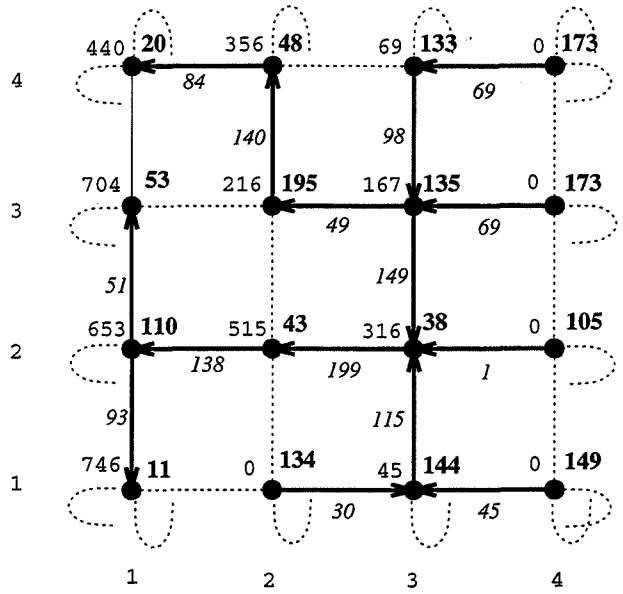


FIG. 3.7 – *Schéma d'exécution des échanges de charges. Les nombres en "courrier" correspondent aux dates au plus tard auxquelles les nœuds auront reçu la charge qu'ils attendent de leurs prédécesseurs dans* $\overrightarrow{T_{v^*}}$. *Le chemin de poids maximum est donné par la suite de nœuds ((4,4) (3,4) (3,3) (3,2) (2,2) (1,2) (1,1)) de poids 746.*

Soient $(charge[v], v \in V)$ les variables aléatoires (en abrégé v.a.) qui représentent le montant des charges des nœuds. Supposons que $(charge[v], v \in V)$ sont stochastiquement indépendantes et suivent une même loi de probabilité. Soit $\overline{charge}$ la moyenne arithmétique de ces v.a. :

$$\overline{charge} = \frac{1}{n} * \sum_{v \in V} charge[v]$$

Pour $v \in V$, soit :

$$Z_v = \sum_{x \in T_v} charge[x] - taille(T_v) * \overline{charge}$$

On a :

$$Z_v = \frac{taille(T_v)}{n} * \sum_{u \notin T_v} charge[u] + \frac{-n + taille(T_v)}{n} * \sum_{w \in T_v} charge[w] \qquad (3.4)$$

Ces variables ont la signification suivante :

- si $Z_v = 0$, il n'y a pas d'échange entre v et son prédécesseur $v^{(0)}$ (dans la hiérarchie T_{v^*}),
- si $Z_v > 0$, le prédécesseur $v^{(0)}$ envoie la quantité Z_v de charges à v,
- sinon v envoie à son père $v^{(0)}$ la quantité $|Z_v|$ de charges.

Notons $u \rightarrow w$ le chemin d'origine u et d'extrêmité w dans $\vec{T}$ et $W_{u\rightarrow w}$ son poids :

$$W_{u\rightarrow w} = \begin{cases} 0 & \text{si le chemin n'existe pas} \\ \sum_{v\in u\rightarrow w} |Z_v| & \text{sinon.} \end{cases}$$

Alors le poids maximum des chemins de $\vec{T}$ est :

$$W^* = \max_{(u,w)\in V\times V} W_{u\rightarrow w}.$$

Remarquons d'abord que $Z_{v^*} = 0$ et que les v.a. $Z_v, v \in V$ sont des combinaisons linéaires des v.a. $(charge[v], v \in V)$, mais qui ne sont pas linéairement indépendantes, car la matrice de transformation est singulière; par suite, les v.a. Z_v ne sont pas stochastiquement indépendantes.

Une autre remarque que nous pouvons faire est que les v.a. Z_v sont centrées $(E(Z_v) = 0)$ et, donc, l'espérance de la loi de départ ne va pas intervenir dans le calcul de W^*; en d'autres termes, les charges $charge[v]$ et $charge[v] + \epsilon$ donnent la même quantité W^*. Il faut aussi dire que si les v.a. Z_v ont toutes la même espérance, leurs variances sont différentes, dépendantes de la position du nœud v dans l'arbre T_{v^*} :

$$V(Z_v) = \frac{taille(T_v)(n - taille(T_v))}{n} * \sigma^2$$

σ^2 étant la variance des v.a. $charge[v]$.

Pour certaines lois de la charge initiale il est possible de trouver la loi d'une v.a. Z_v, par exemple, si $(charge[v], v \in V)$ suivent une loi normale, $(Z_v, v \in V)$ suivent aussi des loi normales. Même si on arrive à déterminer les lois des $Z_v, v \in V$, il est très difficile de trouver de façon analytique les lois des $W_{u\rightarrow v}$, car elles sont des sommes, sous certaines conditions, de v.a. dépendantes, et encore plus difficile de trouver la loi de W^* qui est un maximum de ces v.a.. Toutefois il existe un théorème des valeurs extrêmes[1] [Sap90] :

Théorème : Soient $X_1, X_2, ..., X_n$ des variables aléatoires indépendantes suivant une même loi avec la fonction de répartition F et $Y_n = \max(X_1, X_2, ..., X_n)$. Alors il existe a_n, b_n tel que $a_n * Y_n + b_n$ tend vers $G(y)$, où $G(y)$ est :

- soit une loi de Weibull (ou de Frechet), $G(y) = e^{-y^\alpha}$, si $1 - F(x)$ converge vers 0 comme x^{-k} quand $x \rightarrow \infty$;
- soit une loi de Gumbel, $G(y) = e^{-e^{-y}}$, si $1 - F(x)$ converge vers 0 comme e^{-x} quand $x \rightarrow \infty$.

Il y a des cas où les conditions de ce théorème peuvent être relaxées sans trop modifier la conclusion [FHR94]. Mais dans notre cas, les variables aléatoires $W_{u\rightarrow v}$ sont bien loin des conditions de régularité qui permettraient la relaxation des hypothèses du théorème :

- les v.a. $W_{u\rightarrow v}$ ne sont pas indépendantes et la matrice de corrélation est inconnue;
- les v.a. $W_{u\rightarrow v}$ ne suivent pas une même loi de probabilité;

1. Ce résultat est de même type que le théorème central-limite : si $X_1, X_2, ..., X_n$ sont des v.a. indépendantes de même loi et $S_n = \frac{X_1 + X_2 + \cdots X_n}{n}$, alors il existe a_n, b_n tel que $\frac{S_n - b_n}{a_n}$ tend vers une loi normale quand n tend vers ∞.

– les v.a. $W_{u \to v}$ ne sont pas des v.a. centrées.

Il semble donc difficile de prouver formellement un résultat similaire. Nous procéderons donc par une analyse statistique des simulations du modèle que nous venons de définir.

3.5.3.2 Simulations

Afin de trouver de manière empirique la distribution de la v.a. W^*, nous avons fait des simulations en obtenant des échantillons de taille suffisamment grande: 400 et 1000. Ensuite, sur la base de ces échantillons, nous avons formulé des hypothèses probabilistes quant au comportement de W^* et appliqué des tests d'adéquation pour les vérifier.

3.5.3.2.1 Conditions de simulation Nous avons pris comme loi initiale de distribution des v.a. $charge[v], v \in V$ une loi normale (comme loi continue) et une loi de Poisson (comme loi discrète). Le choix de ces deux lois a été dicté par le fait que la charge d'un processeur à un moment donné est la somme des charges unitaires qui se trouvent dans la file d'attente du processeur [Kle75] et qui sont générées une à une à des instants précis:

$$charge[v] = \sum_{i=1}^{n} X_v^{(i)}$$

où $X_v^{(i)}$ sont v.a. discrètes de Bernoulli $\mathcal{B}(p_n)$. Le comportement à la limite est le suivant, si:

– $p_n \geq p > 0$, alors il existe a_n, b_n tel que $\dfrac{charge[v] - a_n}{b_n}$ tend vers une loi normale;

– $n * p_n \to \lambda$ pour $n \to \infty$, alors $charge[v] = \sum_{i=1}^{n} X_v^{(i)}$ tend vers la loi de Poisson $\mathcal{P}(\lambda)$.

Les paramètres de ces loi $\mathcal{P}(\lambda)$ et $\mathcal{N}(400, \sigma)$ [2] ont été pris égaux à:

· 1, 10, 15, 20, 30, 50, 100, 1000 pour λ;

· 20, 40, 50, 80, 100 pour σ.

Les topologies machines choisies ont été les tores bidimensionnels $k \times k$, $k = 4, 5, 6, 7, 8, 9, 10, 15$.

Les simulations ont été réalisées dans l'environnement MATLAB. A chaque fois, nous avons calculé le poids maximum des chemins de l'arbre de régulation.

3.5.3.2.2 Résultats Le premier pas après avoir réalisé les simulations a été de dresser la distribution empirique de W^* obtenue pour différents jeux de paramètres. La figure 3.8 donne les distributions de deux échantillons lissées selon [Sap90], pages 119-121.

Nous avons pu constater que plus la topologie est grande, plus les chemin W^* sont coûteux; ces coûts augmentent aussi avec la variance de la loi initiale de génération des charges.

2. Nous avons fait remarquer que l'espérance $M = E(charge[v])$ n'intervient pas dans le calcul de W^*.

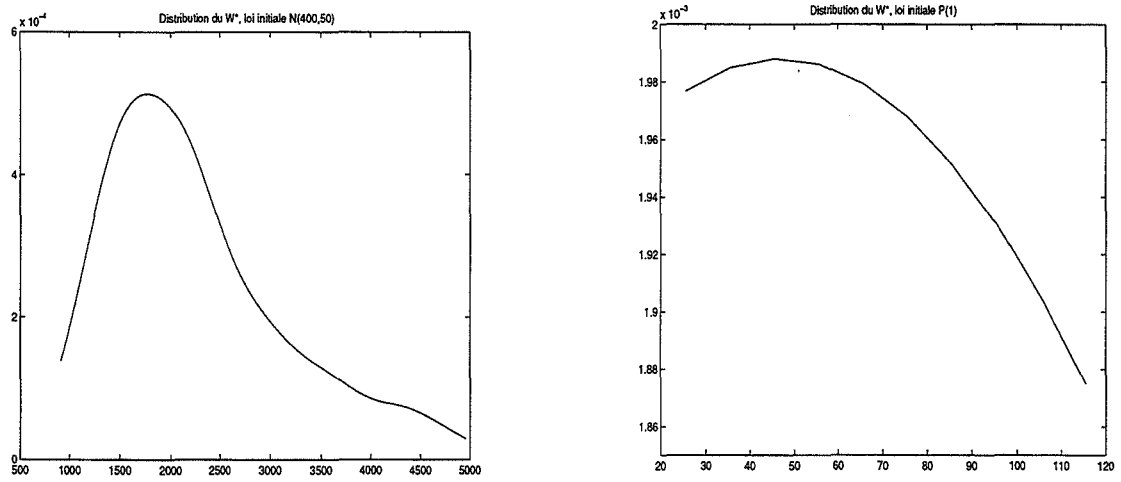


FIG. 3.8 – *Les distributions de deux échantillons de taille 400 pour l'arbre extrait du tore* 10×10*; la loi des charges initiales est* $\mathcal{N}(400, 50)$ *à gauche et* $\mathcal{P}(1)$ *à droite.*

3.5.3.3 Tests d'adéquation

Au premier regard des distributions obtenues, on n'a pas une idée précise sur la nature de la loi de W^*. On ne sait pas si il y a une loi connue et répertoriée qui se cache derrière la figure 3.8, ou même si on va arriver à trouver une formule quelconque pour exprimer cette loi.

Nous avons d'abord pensé que la loi de W^* pourrait être la loi lognormale, vu l'allure de ce graphique. Pour se faire une idée de la validité de cette hypothèse, nous avons tracé les papiers fonctionnels[3] de la loi lognormale[4], voir la figure 3.9. Ces papiers fonctionnels suggèrent que la loi recherchée n'est pas la loi lognormale. D'autre part, nous avons vu qu'essayer tour à tour toutes les lois répertoriées n'est pas du tout une approche raisonnable.

Même si les conditions du théorème des événements extrêmes sont loin d'être satisfaites, nous avons essayé de comparer les distributions obtenues avec ces deux lois. Nous avons d'abord pensé à faire les papiers fonctionnels de Weibull et de Gumbel, comme dans la figure 3.10. On peut constater que la loi de Weibull ne convient pas et que pour la loi de Gumbel il est nécessaire de faire des tests statistiques plus fins.

Selon ces résultats, où le graphique est plus proche d'une droite dans les papiers de Gumbel, nous avons fait des tests statistiques plus fins pour vérifier l'hypothèse:

($\mathcal{H}$) " W^* suit une loi de Gumbel"

La loi de Gumbel est une loi paramétrée, la fonction de répartition d'une v.a. X qui suit une telle loi est:

$$F(x) = \mathrm{e}^{-\mathrm{e}^{-(a*x+b)}}$$

a est un paramètre d'échelle et b un paramètre de centrage.

3. Voir l'annexe B, on cherche donc à obtenir un graphique proche d'une droite.
4. Voir annexe 2.

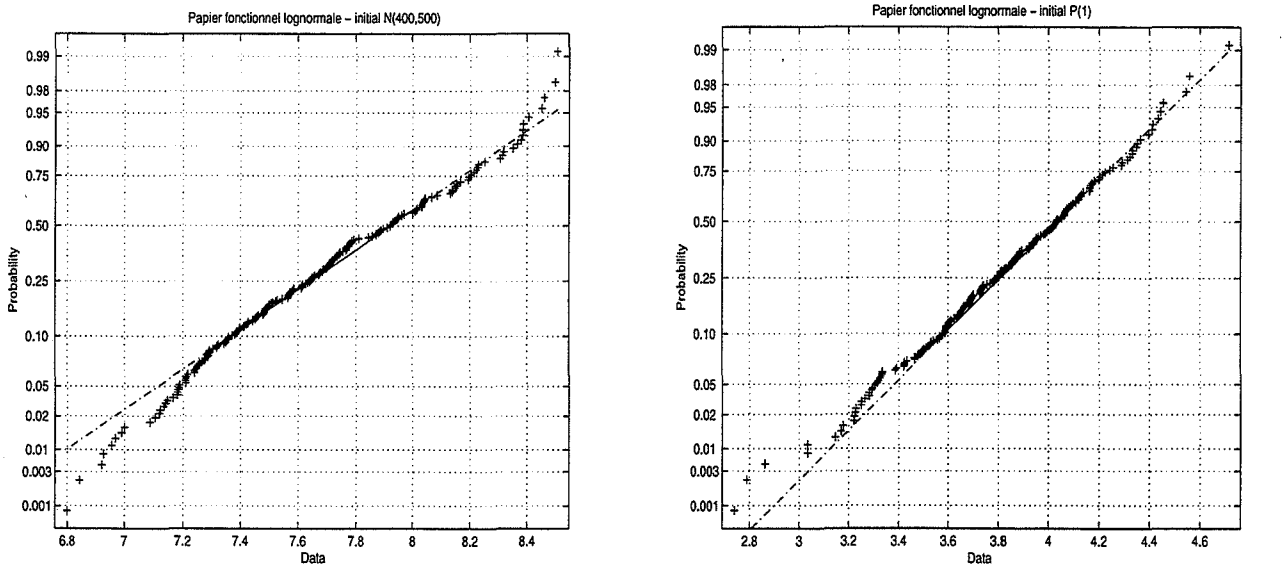


FIG. 3.9 – *Les papiers fonctionnels de la loi lognormale pour les échantillons représentés dans la figure 3.8.*

Avant d'appliquer les tests, nous avons estimé les paramètres a et b par la méthode du maximum de vraisemblance en prenant comme loi de départ pour la résolution numérique itérative du système les paramètres $(\tilde{a}, \tilde{b})$ estimés par la méthode des moments. L'annexe B.3 donne en détail les calculs nécessaires à la recherche de ces paramètres.

Les tests choisis pour valider l'hypothèse $(\mathcal{H})$ ont été le test de χ^2 et le test de Kolmogorov. Les deux tests ont confirmé l'hypothèse probabiliste avec un niveau de confiance $\alpha = 0.95$ et, respectivement, $\alpha = 0.8$. Le niveau de confiance très élevé du test de χ^2 est dû à la perte d'information engendrée par le regroupement des valeurs en intervalles.

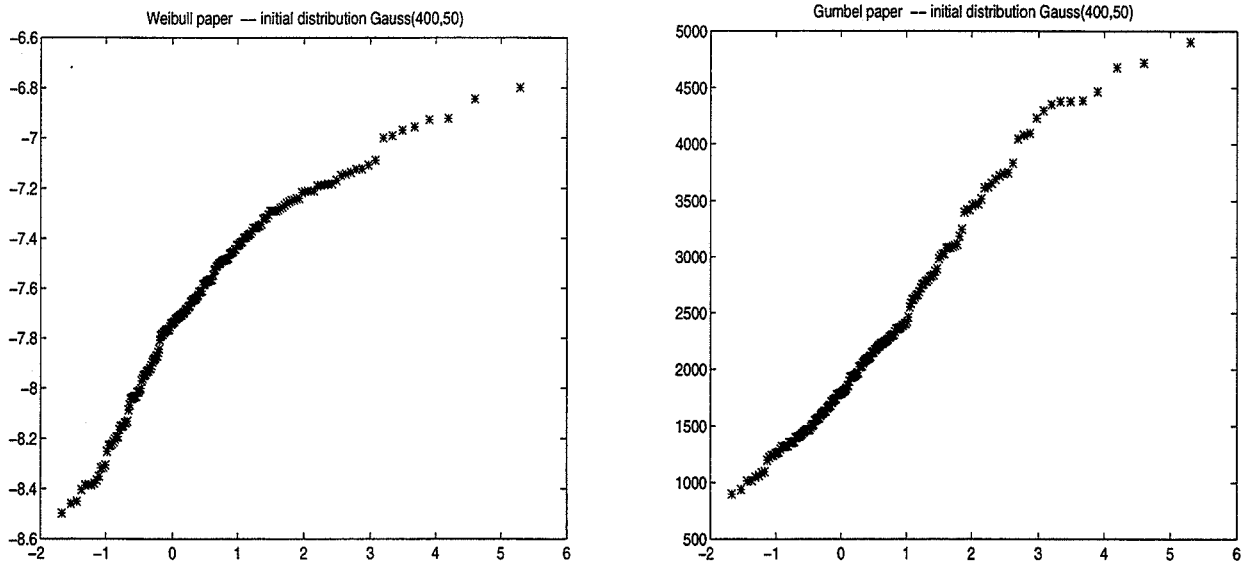
3.5.3.4 Conjecture

Nous avons fait aussi quelques simulations avec d'autres lois de génération de charge initiale, la loi uniforme et la loi exponentielle, et nous avons toujours confirmé l'hypothèse que W^* suit une loi de Gumbel. Selon le théorème des valeurs extrêmes, nous pouvons formuler :

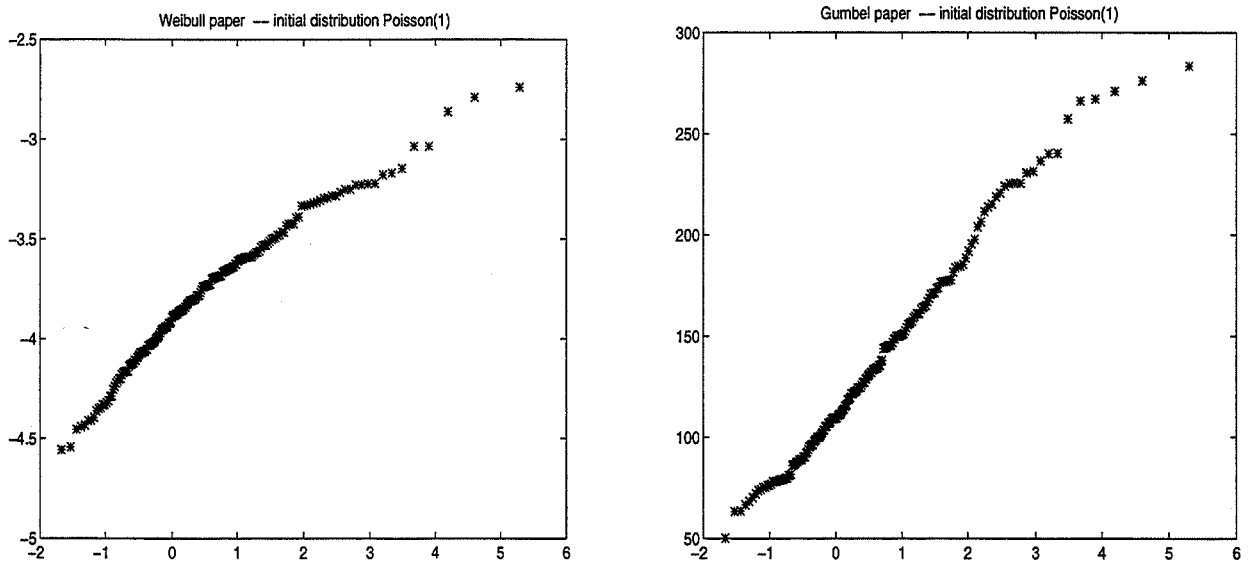
Conjecture : Pour tout réseau d'interconnexion "de grande taille" et pour toute loi de distribution de charges initiales, le temps de régulation de charge suit une loi de Gumbel.

La taille "grande" du réseau d'interconnexion des processeurs signifie au moins une dizaine de nœuds et assure la véridicité de la conjecture.

Le corollaire, qui découle naturellement de cette conjecture et qui est d'une importance capitale pour la stratégie de régulation de charge proposée est le suivant :



3.10.a



3.10.b

FIG. 3.10 – *Les papiers de Weibull et de Gumbel pour des échantillons de 400 valeurs du coût du chemin maximal dans le tore* 10×10 *quand la distribution initiale de charge est* $\mathcal{N}(400, 50)$ *(3.10.a) et* $\mathcal{P}(1)$ *(3.10.b).*

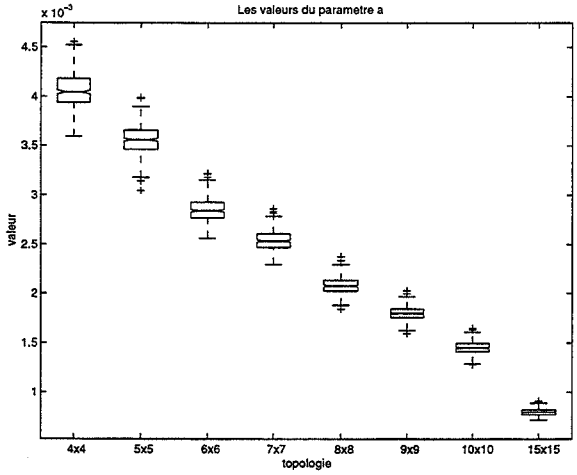


FIG. 3.11 – *Les valeurs du paramètre a quand $V(charge[v]) = 50$ pour différents arbres extraits des tores*

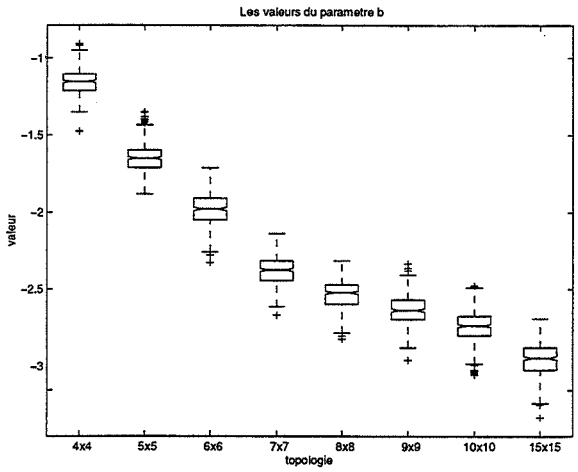


FIG. 3.12 – *Les valeurs du paramètre b en fonction de la topologie machine*

Corollaire : Si $p \in (0,1)$, alors le temps maximal de régulation est borné avec la probabilité p par :

$$\tau * \frac{-\ln(-\ln p) - b}{a}. \tag{3.5}$$

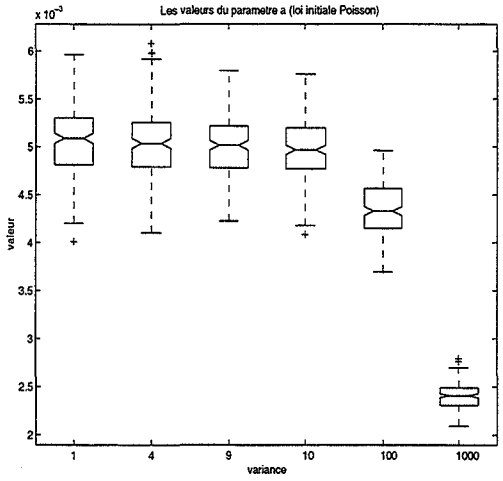
Preuve: Selon la conjecture énoncée, la fonction de répartition de la variable W^* est $F(x) = e^{-e^{-(a*x+b)}}$.

Soit T_p la borne du temps maximal de régulation acceptée avec la probabilité p. Alors $p = P(\tau * W^* \leq T_p)$, et donc $T_p = \tau * F^{-1}(p)$, d'où $T_p = \tau * \frac{-\ln(-\ln p) - b}{a}$. □

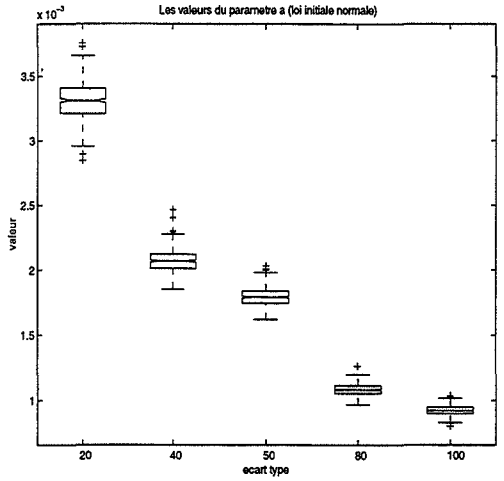
Quant aux paramètres de la loi de Gumbel nous avons pu constater par simulation qu'ils ne sont pas sensibles à la nature de la loi initiale (Poisson, normale ou autres). Le paramètre b est fonction uniquement de l'arbre recouvrant, tandis que le paramètre a dépend, en plus, de la variance de la loi initiale : σ^2 pour la loi normale et λ pour la loi de Poisson. Les simulations laissent voir que le paramètre a est inversement proportionnel à l'écart-type de charges initiales. La figure 3.12 donne les valeurs du paramètre b pour les arbres extraits des tores. La figure 3.11 donne les valeurs de a pour les mêmes arbres et quand $V(charge[v]) = 50$. La figure 3.13 montre les variations du paramètre a selon les paramètres de la loi de départ. Les paramètres ont été estimés avec des intervalles de confiance, en partant de l'échantillon de 400 valeurs par la méthode bootstrap [Efr82] et les graphiques représentent ces intervalles [Tuk77].

En s'appuyant sur la conjecture énoncée, nous pouvons effectivement montrer que le paramètre b reste inchangé par rapport à la loi de génération des charges initiales et que a est inversement proportionnel à l'écart type de cette loi.

Proposition 1 : Soit une topologie assez "grande" (telle que la conjecture soit satisfaite) et deux lois de génération de charges initiales qui ont le même écart-type. Alors les paramètres de la loi de Gumbel



Distribution initiale – $\mathcal{P}(\lambda)$



Distribution initiale – $\mathcal{N}(50, \sigma)$

FIG. 3.13 – *La variation du paramètre a en fonction de la variance(l'écart-type) de la loi initiale.*

pour la v.a. W^* sont les mêmes.

Preuve: Soient $Ch_v^{(1)}$, $v \in V$ et $Ch_v^{(2)}$, $v \in V$ des charges initiales générées selon deux lois différentes et $V(Ch_v^{(1)}) = V(Ch_v^{(2)}) = \sigma$.

Selon l'équation de définition (3.4): $E(Z_v^{(1)}) = E(Z_v^{(2)}) = 0$ et $V(Z_v^{(1)}) = V(Z_v^{(2)}) = \alpha_v * \sigma$, où α_v est une constante associée à la position de v dans l'arbre T. De même

$$Cor(Z_v^{(1)}, Z_u^{(1)}) = Cor(Z_v^{(2)}, Z_u^{(2)}) = \beta_{v,u} * \sigma$$

On peut en déduire que:

$$E(W^{(1)*}) = E(W^{(2)*})$$
$$V(W^{(1)*}) = V(W^{(2)*})$$

Conformément à la conjecture énoncée, $W^{(1)*}$ et $W^{(2)*}$ sont des v.a. de Gumbel et selon les formules (B.1, B.2) on obtient:

$$\frac{\gamma - b^{(1)}}{a^{(1)}} = \frac{\gamma - b^{(2)}}{a^{(2)}}$$
$$\frac{\frac{\pi^2}{6}}{a^{(1)^2}} = \frac{\frac{\pi^2}{6}}{a^{(2)^2}}$$

d'où on obtient que $a^{(1)} = a^{(2)}$ et $b^{(1)} = b^{(2)}$. □

Proposition 2: Pour une topologie assez "grande", le paramètre b est invariable par rapport à la loi de génération des charges initiales et le produit entre l'écart-type de la loi de génération des charges initiales et le paramètre a de la v.a. W^* est constant.

Preuve: Selon l'assertion précédente il suffit de prouver que si $Ch'_v = \alpha * charge[v]$, alors W'^* admet le même paramètre b que W^* et que $\alpha * a' = a$, car si $V(charge[v]) = \sigma$, $V(Ch'_v) = \alpha * \sigma$.

Si $Ch'_v = \alpha * charge[v]$, $\forall v \in V$, alors $Z'_v = \alpha * Z_v$, $\forall v \in V$. Et donc $W'_{u\to v} = \alpha * W_{u\to v}$, ce qui enduit que $W'^* = \alpha * W^*$. En appliquant l'espérance et la variance à cette relation :

$$\begin{aligned} E(W'^*) &= \alpha * E(W^*) \\ V(W'^*) &= \alpha^2 * V(W^*) \end{aligned}$$

on obtient toujours que :

$$\begin{aligned} \frac{\gamma - b'}{a'} &= \alpha * \frac{\gamma - b}{a} \\ \frac{\frac{\pi^2}{6}}{a'^2} &= \alpha^2 * \frac{\frac{\pi^2}{6}}{a^2} \end{aligned}$$

La résolution de ce système mène aux relations suivantes :

$$\begin{aligned} \alpha * a' &= a \\ b' &= b \end{aligned}$$

□

Donc le paramètre b est figé pour un arbre donné et $a * \sigma = C$ où σ est l'écart-type de charges initiales et C est une constante liée aussi à la topologie. Alors la formule (3.5) de la borne supérieure du temps de régulation est de l'ordre $O(\tau * \sigma * C')$, où C' est une nouvelle constante.

Le paramètre b et la constante C peuvent s'estimer à l'aide de simulations sur la topologie donnée. Nous ne sommes pas arrivé à trouver des formules pour exprimer ces paramètres.

3.6 Un algorithme complet de régulation de charge

Jusqu'ici nous avons présenté une stratégie de régulation de charge ainsi que son analyse de complexité répondant ainsi à la question : comment faire la régulation?. Si on veut utiliser effectivement cette stratégie, il faudra lui adjoindre une politique de décision et une politique d'initiation.

Donc il reste à répondre à deux questions :

- quand initier la régulation?
- comment décider si la régulation sera bénéfique?

La stratégie proposée est générique, elle ne dépend pas de l'application, alors que les réponses à ces deux questions risquent de dépendre fortement de la nature de l'application, i.e. du taux de création des nouvelles unités de charge et de leur temps de traitement. Par le même souci de généricité nous proposons une politique de décision dans la sous-section 3.6.2 et une politique d'initiation dans la sous-section 3.6.3 qui ne prennent pas en compte ces spécificités. Avant d'entamer le développement de ces politiques, nous montrons quelle est la signification du résultat précis de complexité obtenu auparavant, dans la sous-section 3.5.3. En fin de section 3.6.4, l'algorithme résultant qui englobe la stratégie et les politiques de décision-initiation sera détaillé.

Le problème d'une politique de transfert qui décidera quelles unités de charge choisir pour les transférer ne se pose pas, car les charges sont supposées être régulières, donc avec un même temps de transfert / migration.

3.6.1 Préliminaires. Conditions suffisantes de régulation

Prenons d'abord les notations suivantes :

· $T_{regulation}$ le temps nécessaire pour faire la régulation;

· $T_{//\_avec}$ le temps pris par l'application jusqu'à sa fin si la régulation est mise en œuvre;

· $T_{//\_sans}$ le temps pris si la régulation n'est pas utilisée.

La condition nécessaire et suffisante de régulation est :

$$T_{//\_avec} + T_{regulation} < T_{//\_sans} \tag{3.6}$$

Si la régulation n'est pas faite, le temps de l'application est :

$$T_{//\_sans} = \theta * \max_{v \in V} charge[v] \tag{3.7}$$

θ étant le temps d'exécution d'une unité de charge. Suite à la régulation, le temps d'exécution devient :

$$T_{//\_avec} = \theta * \overline{charge} \tag{3.8}$$

Donc selon (3.6) et les relations (3.7) et (3.8) la condition de régulation est :

$$T_{regulation} < \theta * (\max_{v \in V} charge[v] - \overline{charge}) \tag{3.9}$$

Dans cette formule on fait intervenir le facteur de déséquilibre du système : $\max_{v \in V} (charge[v] - \overline{charge})$, comme dans [WLR93].

Si on tient compte de la borne supérieure T_p acceptée avec la probabilité p pour le temps de régulation :

$$T_{regulation} \leq T_p$$

$$T_p = \tau * \frac{-\ln(-\ln p) - b}{a}$$

une condition suffisante de régulation est :

$$T_p < \theta * (\max_{v \in V} charge[v] - \overline{charge}) \tag{3.10}$$

C'est à partir de cette condition que nous allons inférer les conditions à vérifier pour les politiques d'initiation et de décision du processus de régulation.

3.6.2 Une politique de décision

Dans la stratégie de régulation, la racine de l'arbre recouvrant joue un rôle primordial, car c'est le premier nœud capable de calculer la charge moyenne et c'est le nœud qui diffuse cette information dans le réseau. Pour cette raison, la racine est le nœud le mieux indiqué pour décider de l'opportunité de la régulation, c'est à dire vérifier si :

$$\tau * \frac{-\ln(-\ln p) - b}{a} < \theta * (\max_{v \in V} charge[v] - \overline{charge}) \tag{3.11}$$

Parce que $\tau * \frac{-\ln(-\ln p) - b}{a}$ est la borne acceptée du temps maximal de la régulation, on peut renforcer l'inégalité (3.11) en imposant que :

$$\frac{\tau * \frac{-\ln(-\ln p) - b}{a}}{\theta * (\max_{v \in V} charge[v] - \overline{charge})} < \alpha \tag{3.12}$$

où $\alpha \in (0, 1)$ est constant.

Le paramètre b est supposé connu; quant au paramètre a, il dépend du radical de la variance des charges initiales :

$$a * \sqrt{V(charge)} = C$$

Pour ce radical, l'écart type des valeurs $charge[v], v \in V$ est un estimateur consistant :

$$\mathsf{std}(charge) = \sqrt{\frac{1}{n-1} * (\sum_{v \in V} charge[v]^2 - \overline{charge}^2)}$$

Donc, avant de permettre le commencement du processus de régulation, la racine devra s'assurer que :

$$\frac{\frac{\tau * (-\ln(-\ln p) - b)}{C} * \mathsf{std}(charge)}{\theta * (\max_{v \in V} charge[v] - \overline{charge})} < \alpha \tag{3.13}$$

Les valeurs qui interviennent dans cette formule : $\max_{v \in V} charge[v]$, $\sum_{v \in V} charge[v]^2$ se calculent toujours par calcul de préfixe généralisé de même type que pour $\sum_{v \in V} charge[v]$ lors de la remontée des informations vers la racine.

3.6.3 Une politique d'initiation

Le but d'une politique d'initiation est de détecter les situations où la régulation serait capable d'améliorer les performances en temps de l'application. Ainsi étant donné les coûts d'information, il est souhaitable que, lorsqu'un nœud sollicite une régulation, le déséquilibre du système vu de ce nœud soit sûr. La politique de décision qui survient après, basée sur une vue globale, plus large et plus précise sur l'état de charge, renforcera ou infirmera cette initiation.

Comme signalé dans 2.3, une demande trop fréquente de régulation engendre une perte de performances par le surcoût induit; d'autre part, les pertes de performances peuvent également se produire si une régulation n'est pas faite tandis qu'un nombre important des processeurs est sous-chargé.

Si la politique de décision peut être centralisée, la politique d'initiation doit être distribuée car le fonctionnement de la machine parallèle est distribué et tous les nœuds sont potentiellement capables d'engendrer des déséquilibres de charge.

Les politiques d'initiation sont variées et dépendent essentiellement de l'état de charge des nœuds. Il y a donc selon [SKS92] :

- des politiques où les nœuds sous-chargés initient;
- des politiques où les nœuds surchargés initient;
- des politiques mixtes.

Dans tous les cas, il faut définir quand un nœud est considéré comme sous-chargé ou surchargé.

L'hypothèse de régularité sous laquelle nous travaillons stipulant que les unités de charge exigent le même temps de traitement, l'unique source de déséquilibre est la création des nouvelles unités de charge. Donc dans ce cas uniquement les nœuds surchargés vont engendrer le besoin de régulation.

Pour préciser ce qu'est un nœud surchargé, nous introduisons comme dans [WLR93] une borne supérieure. Afin de refléter en permanence l'état global de charge, nous la prendrons variable pendant l'exécution de l'application et de même valeur pour tous les processeurs.

Soit Sup cette valeur. Si :

$$charge[v] > Sup$$

il faudra initier la régulation. Sans perte de généralité, on peut supposer que

$$charge[v] = \max_{x \in V} charge[x]$$

(sinon $charge[v] < charge[v_0] = \max_{x \in V} charge[x]$ induit que $charge[v_0] > Sup$).

La condition (3.10) de la nécessité de la régulation devient :

$$\theta * \overline{charge} + T_{regulation} < \theta * charge[v]$$

d'où :

$$charge[v] > \overline{charge} + \frac{1}{\theta} * T_{regulation}$$

Alors cette borne supérieure de charge s'exprime naturellement comme :

$$\begin{aligned} Sup &= \overline{charge} + \frac{\tau}{\theta} * \frac{-\ln(-\ln p) - b}{a} \\ &= \overline{charge} + \frac{\tau}{\theta} * \mathbf{std}(charge) * \frac{-\ln(-\ln p) - b}{C} \end{aligned} \tag{3.14}$$

Remarquons que dans cette formule les valeurs $\overline{charge}$ et $\mathbf{std}(charge)$ sont globales et localement indisponibles pour chaque processeur, en particulier celui qui a sollicité une régulation. Elles ne peuvent donc être qu'appochées. On peut pour cela recourir soit à des méthodes d'intelligence artificielle, soit

à des méthodes probabilistes (simulation de Monte-Carlo, méthodes statistiques). Dans ce travail, nous recourons à une approche naturelle, à savoir considérer pour $\overline{charge}$ la dernière valeur calculée disponible et pour $\mathsf{std}(charge)$ une approche statistique, par exemple la valeur moyenne totale ou partielle.

Du fait que la valeur de Sup comporte des valeurs approchées, nous la prendrons légèrement relâchée par le biais d'un constante C_S choisie autour de 1 :

$$Sup = C_S * \left(\overline{charge} + \frac{\tau}{\theta} * \mathsf{std}(charge) * \frac{-\ln(-\ln p) - b}{C} \right) \tag{3.15}$$

3.6.4 Algorithme de régulation de charges basé sur des décisions probabilistes

Nous avons donc décrit dans 3.3 une stratégie de régulation, dans 3.6.2 et respectivement, dans 3.6.3 une politique de décision et d'initiation. Dans cette partie nous allons présenter un algorithme complet de régulation basé sur ces politiques et la stratégie discutée précédemment.

Nous l'avons déjà remarqué : la borne supérieure d'initiation Sup dépend des valeurs statistiques sur l'état de charge : $\overline{charge}$ et $\mathsf{std}(charge)$ qui doivent être mises à jour pour avoir des résultats acceptables. D'autre part, il faut s'assurer de l'opportunité des demandes de régulation de charge; en l'absence de toute information sur la dynamique de l'application, un appel périodique selon une période T nous semble une solution convenable. Cette période ne dépend pas seulement de l'application (temps d'exécution d'une unité de charge, nombre d'unités de charge, dynamique de génération), mais aussi des caractéristiques physiques de la machine : temps de communication, rapport entre temps CPU et temps communication.

Donc sur chaque processeur v un module de régulation fonctionne de façon concurrente avec l'application selon le code suivant :

(étiquette) Attendre T unités de temps;
Si $charge[v] \leq Sup$ aller à *(étiquette)*;
Invoquer l'étape d'information;
Attendre la décision de la racine;
Recevoir $\overline{charge}$ et $\mathsf{std}(charge)$;
Mettre à jour Sup;
Si décision = "Oui" alors exécuter les échanges de charge;
Aller à *(étiquette)*;

En début d'application Sup est initialisée à $C_S * \overline{charge}$.

Nous avons travaillé dans le cadre de la régulation de charge pour des application à charges régulières. L'algorithme de régulation présenté a eu comme point de départ une stratégie originale basée sur un calcul de préfixe généralisé. C'est une stratégie correcte et exacte. En plus de sa simplicité de mise en œuvre, la stratégie s'applique à toute topologie du réseau d'interconnexion des processeurs et peut prendre en compte, par un recalcul de l'arbre recouvrant, tout changement du réseau d'interconnexion. Elle est donc extensible.

La stratégie proposée a été analysée pour son temps d'exécution de point de vue algorithmique et aussi de point probabiliste. Nous avons trouvé qu'une loi de probabilité de Gumbel modélise le temps

total d'exécution.

Les estimations probabilistes très fines du temps d'exécution de cette stratégie nous ont permis d'inférer des politiques d'initiation et de décision efficace. En plus, l'algorithme de régulation obtenu (stratégie + politiques) est stable, car les transferts de charge entre deux processeurs se font toujours, au maximum, dans un seul sens et, dans un temps fini, tous les processeurs du réseau disposent de la même charge.

Chapitre 4

Application : Simulation parallèle de la déformation-recristallisation des agrégats polycristallins

L'algorithme de régulation de charges régulières que nous avons présenté dans le chapitre précédent a été mis en oeuvre sur la machine parallèle VOLVOX et utilisé dans une application dynamique concrète qui concerne la simulation d'un modèle mécanique de déformation à chaud des agrégats polycristallins.

Ce travail a fait l'objet de plusiers publication dont [JMSM97c] sur les résultats mécaniques obtenus et [JMSM97a] et [JMSM97b] sur l'implatation parallèle.

Dans ce chapitre, nous présenterons cette application en mettant l'accent sur sa parallélisation et sur l'implantation de l'algorithme de régulation.

4.1 Introduction à la déformation à chaud des agrégats polycristallins

Les expériences physiques montrent que dans des agrégats polycristallins soumis à des déformations à chaud, à part le changement de forme des grains il apparaît deux phénomènes de recristallisation dynamique : continue et discontinue. Une monographie à ce sujet est l'article de Montheillet et Jonas [MJ96]. La recristallisation dynamique discontinue (RDD) désigne la formation de nouveaux grains dans l'agrégat par germination et croissance de certains grains qui ont tendance à englober d'autres grains. La recristallisation dynamique continue (RDC) désigne la formation de nouveaux grains par subdivision de certains grains déformés. Les phénomènes de RDD et RDC sont liés; plus précisément la RDC peut précéder l'apparition de la RDD par croissance. Montheillet et Jonas [MJ96] montrent que la RDC apparaît dans le fer α, les aciers ferritiques, les alliages d'aluminium, matériaux qui ont une forte énergie de défaut d'empilement. La RDC est particulièrement étudiée pour l'aluminium 1200 et 1199 dans [GKMM96] et [GM95].

Notre intérêt porte uniquement sur la recristallisation dynamique continue en tant qu'exemple d'applications composées d'un ensemble de tâches susceptibles d'en générer de nouvelles de façon tout

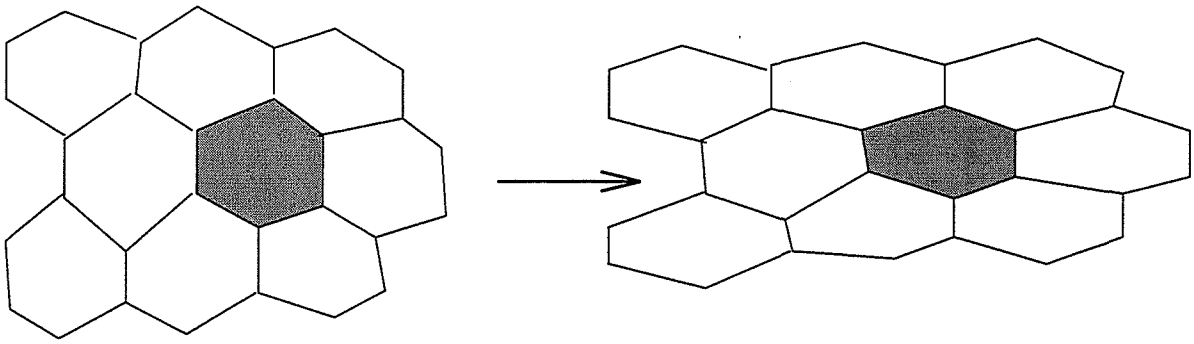


FIG. 4.1 – *Schématisation d'une compression plane lors de la déformation à chaud*

à fait aléatoire. Le but de cette section n'est pas de faire une analyse approfondie des phénomènes physiques, mais de modéliser la RDC afin de comprendre ce qui se passe réellement pendant la déformation à chaud.

4.1.1 Brève description des phénomènes physiques

Nous allons considérer un agrégat de grains soumis à une déformation plane, une compression, par exemple, et nous essayerons de donner un aperçu des phénomènes auxquels on s'intéresse.

Dans un agrégat, chaque grain est caractérisé par sa densité de dislocation ρ_i qui correspond à sa dureté[1]. C'est ce paramètre qui gouverne les transformations qui se produisent avec un grain pendant la déformation. Deux transformations apparaissent lors de la déformation à chaud d'un agrégat:

1. **Les grains se déforment**

   C'est une évolution de la forme des grains, pour un grain i et de leur densité de dislocation; ρ_i évolue selon une équation du type :

   $$d\rho_i = f_i(\rho_i)d\varepsilon_i \tag{4.1}$$

   où $d\varepsilon_i$ désigne l'incrément de déformation, qui dépend du grain, de son voisinage et de la contrainte globale appliquée au matériau. f_i dépend du grain.

   La figure 4.1 schématise une telle déformation.

2. **Les grains recristallisent**

   Lorsque un grain i atteint une condition limite qui sera détaillée ultérieurement, le grain allongé se subdivise en n grains (en moyenne) équiaxes de taille donnée comme dans la figure 4.2. Pour

1. Il y a aussi d'autres caractéristiques d'un grain : taille, forme, orientation cristallographique.

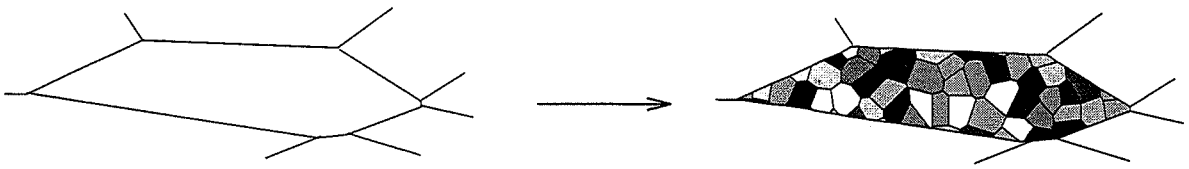


FIG. 4.2 – *Schématisation de la recristallisation*

chaque nouveau grain j, on a $\rho_j = \rho_0 \leq \rho_i$; ils héritent donc en partie des propriétés mécaniques et physiques du grain recristallisé.

4.1.2 Simulation numérique

La simulation numérique d'un modèle d'agrégats polycristallins soumis à la déformation à chaud requiert une puissance de calcul et des capacités de stockage, toutes deux très importantes. En effet, il faut prendre en compte la taille du modèle qui, même si elle est raisonnable en début de simulation (de l'ordre de 10^3 grains), peut atteindre 10^5 - 10^6 grains suite à des recristallisations. D'autre part, le nombre de pas successifs (équations non linéaires) de simulation doit être suffisamment grand afin de mieux discrétiser les transformations subies par le matériau. L'utilisation d'une machine parallèle s'impose donc naturellement pour faire face à cette demande en puissance de calcul et en capacité de stockage.

Cette simulation numérique sera une application dynamique, c'est à dire que pendant l'exécution, de nouvelles unités à traiter surgissent suite à une recristallisation. Nous appliquerons alors la régulation de charge.

La suite de ce chapitre est structurée de la manière suivante: la section 4.2 sera consacrée à la description complète d'un modèle d'agrégats polycristallins et aussi à la description dans le cadre de ce modèle des phénomènes de déformation et de recristallisation dynamique continue. La section suivante 4.3 présentera la simulation numérique du modèle sur une machine parallèle à mémoire distribuée en utilisant pour la phase de recristallisation l'algorithme de régulation de charge proposé dans le chapitre 3. Dans la dernière section du chapitre, nous montrerons la signification mécanique des résultats de simulation obtenus et les perspectives de ce travail.

4.2 Modélisation

4.2.1 Modèle de base d'un agrégat polycristallin

Le modèle que nous allons présenter comporte trois aspects : géométrique, mécanique et métallurgie physique. Ceci constitue la base pour modéliser ensuite les transformations de compression plane et de recristallisation dynamique continue, mais il est possible de modéliser aussi une multitude d'autres transformations mécaniques.

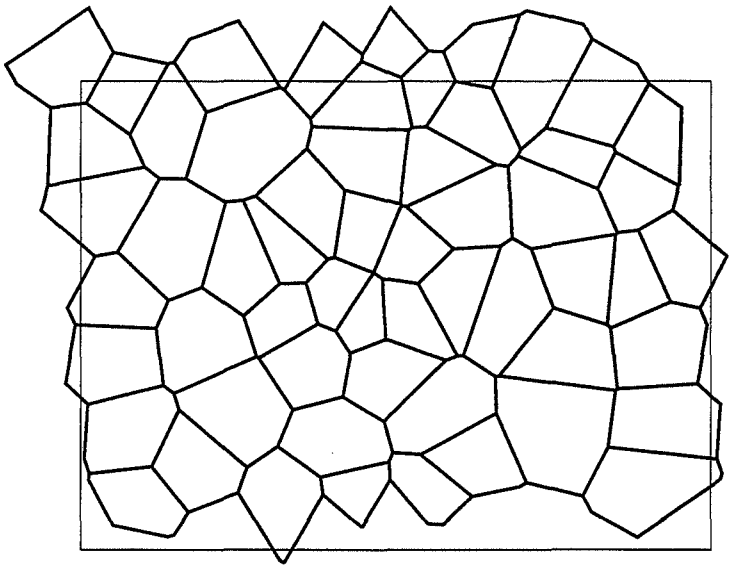


FIG. 4.3 – *Pavage périodique avec 57 polygones*

4.2.1.1 Aspects géométriques

Intuitivement, un agrégat polycristallin peut être imaginé comme un empilement de polyèdres dans l'espace euclidien. La compression subie par l'agrégat dans notre étude est plane (dans une unique direction), il suffit donc de regarder seulement un plan qui contient la direction de compression.

Nous allons modéliser l'agrégat par un pavage périodique de polygones, comme dans la figure 4.3. Ce pavage est obtenu par une tesselation de Voronoï [BY95][2] ce qui assure que chaque point de la tesselation est un point triple (appartient à trois polygones) et que, en moyenne, chaque polygone a 6 sommets.

A chaque grain i correspond un polygone $P_i = [V_i^1, V_i^2, \cdots, V_i^n]$ où $V_i^k = (x_i^k, y_i^k)$. Ceci permet de prendre en compte la forme du grain λ_i lors de la compression et de la recristallisation. Il est possible aussi de décrire les voisins physiques d'un grain i.

L'évolution de chaque grain dépend de ses propres caractéristiques physiques et mécaniques, des conditions auxquelles l'agrégat est soumis, mais aussi du voisinage physique du grain. Donc cette évolution pourrait être assimilée au fonctionnement d'un réseau d'automates[3] comme dans les travaux de Montheillet et al. [MG96, MG94], avec la généralisation que les voisins considérés sont physiquement voisins en partageant entre eux une frontière commune.

4.2.1.2 Aspects mécaniques

Chaque grain i se verra associer, en plus de sa frontière géométrique, des paramètres physiques et mécaniques. Dans la modélisation que nous sommes en train d'expliciter trois paramètres importants

2. La tesselation est calculée pour des germes tirés aléatoirement à l'intérieur d'un rectangle de base.
3. Voir [Neu66] pour une monographie du sujet.

pour l'évolution des grains sont :

- ε_i – la déformation locale du grain i ($\dot{\varepsilon}_i$ désigne le taux de déformation);
- σ_i – la contrainte microscopique;
- k_i – la viscosité (dureté) du grain i, de faibles valeurs indiquent des milieux mous, et de grandes valeurs indiquent des milieux durs.

A ces paramètres correspondent les paramètres globaux associés à l'agrégat :

- ε_∞ – la déformation globale;
- σ_∞ – la contrainte macroscopique;
- K – la viscosité globale.

Les paramètres locaux sont liés par la loi du comportement local viscoplastique :

$$\sigma_i = k_i \dot{\varepsilon}_i^m \tag{4.2}$$

où m est une constante associée au matériau, pour l'aluminium à 400°C $m = 0.2$. La contrainte macroscopique est obtenue par homogénéisation :

$$\sigma_\infty = < \sigma_i > \tag{4.3}$$

ainsi que la vitesse de déformation :

$$\dot{\varepsilon}_\infty = < \dot{\varepsilon}_i >$$

où la notation $< . >$ indique une moyenne globale pondérée par les surfaces. La vitesse de déformation ε_i est obtenue par une localisation de $\dot{\varepsilon}_\infty$ en tenant compte du contour polygonal du grain, de sa viscosité et des viscosités des grains voisins j :

$$\dot{\varepsilon}_i = \phi(k_i, \lambda_i, (k_j, j \text{ voisin de } i)) \tag{4.4}$$

Nous développons dans la sous-section suivante le calcul des déformations locales pour le cas de la compression plane.

4.2.1.3 Aspects métallurgie physique

Pour décrire la variation de la viscosité d'un grain i pendant les transformations à chaud il faut tenir compte de l'évolution du grain, plus particulièrement de la déformation subie :

$$k_i = \left[k_{si}^2 - (k_{si}^2 - k_{0i}) \mathrm{e}^{(-r_i \varepsilon_i)} \right]^{1/2} \tag{4.5}$$

loi dite de Laasraoui et Jonas, où k_{0i} et k_{si} sont les viscosités locales pour déformations incipientes et à grande déformation, respectivement, et r_i est un paramètre de restauration. Les simulations numériques du modèle ont utilisé pour ces constantes les valeurs établies expérimentalement par

Gourdet [Gou97] : $k_{0i} = 0$ et $r_i = 3$ pour tous les grains et k_{si} est choisi uniformément distribué dans l'intervalle 68 - 85 MPa et reflète l'orientation cristallographique du grain.

Le taux d'énergie plastique $\dot{w}_i$ d'un grain i est :

$$\dot{w}_i = k_i \dot{\varepsilon}_i^{m+1} \tag{4.6}$$

et le taux de l'énergie plastique globale par unité de surface est :

$$\dot{W} = K \dot{\varepsilon}_\infty^{m+1} \tag{4.7}$$

L'énergie totale plastique d'un grain w_i joue le rôle du paramètre de contrôle ρ_i qui gouverne la recristallisation.

4.2.2 Modélisation de la compression et de la recristallisation dynamique continue

C'est un modèle qui est complet avec trois ingrédients de base : géométrique-morphologique, mécanique, physique (métallurgique). Il permet d'englober d'autres paramètres associés aux cristaux composants et aussi de modéliser des phénomènes de compression, recristallisation dynamique continue et discontinue, glissement des joints de grain, etc. Nous allons décrire juste la modélisation de la compression plane et de la recristallisation dynamique continue.

4.2.2.1 Compression plane

Lors d'une compression plane, tous les grains de l'agrégat s'aplatissent (i.e. leurs axes dans le sens de la déformation se réduisent) sans changer de voisins, ni de volume. Par contre, les coordonnées de polygones P_i associés changent selon la déformation :

$$\mathrm{d}\varepsilon_i = \dot{\varepsilon}_i \, \mathrm{d}t \tag{4.8}$$

subie dans un intervalle de temps $\mathrm{d}t$.

Nous allons supposer dans la suite que la compression est faite dans le sens de l'axe des ordonnées, ce qui signifie que les grains s'aplatissent selon l'axe des y et s'allongent selon l'axe des x.

A chaque pas de déformation ($\mathrm{d}t = \text{const.}$), la viscosité moyenne du voisinage de chaque grain i est alors calculée :

$$\overline{k_i} = \sum_{j \text{ voisin de } i} \alpha_j k_j \tag{4.9}$$

où les α_j sont des coefficients de pondération, $\sum_{j \text{ voisin de } i} \alpha_j = 1$, proportionnels à la frontière commune entre le grain i et les grains j voisins.

Ensuite un facteur de localisation δ_i est calculé selon Briottet et al. [BGM97] :

$$\delta_i = \frac{(1 + aF)(1 - \tanh\left((b + cF)(\Sigma_i - 1)\right))}{1 + aF + (-1 + dF)\tanh\left((b + eF)(\Sigma_i - 1)\right)} \tag{4.10}$$

où :

- $F = \dfrac{2\,\lambda_i}{\lambda_i^2 + 1}$, λ_i étant le rapport de forme du polygone P_i,

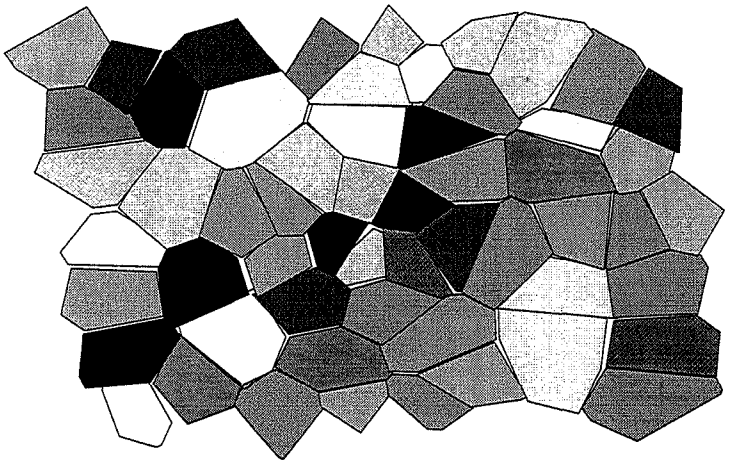


FIG. 4.4 – *Structure déformée* $d\varepsilon_\infty = 0.01$, $\varepsilon_\infty = 0.1$, *sans réaliser le collage des sommets.*

- Σ_i est le rapport des viscosités : $\frac{k_i}{\overline{k_i}}$
- a, b, c, d, e sont des constantes déterminées numériquement, voir [BGM97].

Le taux de déformation peut s'exprimer selon la loi du comportement viscoplastique (4.2) et le principe d'homogénéisation (4.3) :

$$\dot{\varepsilon}_i = \frac{\delta_i}{< \delta_i >} \dot{\varepsilon}_\infty \tag{4.11}$$

($< . >$ signifie toujours la moyenne globale des valeurs).

Une fois $\dot{\varepsilon_i}$ déterminé, on connaît l'incrément $d\varepsilon_i$ et on peut calculer les nouvelles coordonnées (x_i^G, y_i^G) du barycentre G du polygone P_i :

$$x_i^G(t + dt) = x_i^G(t)\,(1 + d\varepsilon_i)$$

$$y_i^G(t + dt) = y_i^G(t)\,(1 - d\varepsilon_i)$$

Ensuite par la résolution d'un système linéaire on calcule la translation du grain i $\vec{t_i}$ qui se compose avec la compression $(d\varepsilon_i, -d\varepsilon_i)$ et donc on calcule les nouvelles coordonnées de tous les sommets provisoires du grain i, notons-les $W_i^k, k = 1, n$.

Un point du pavage appartient à trois polygones; si ses nouvelles coordonnées sont calculées pour chacun des polygones, il est très possible qu'on obtienne trois points différents. La figure 4.4 montre une structure déformée avec un incrément global $d\varepsilon_\infty = 0.01$ pendant 10 pas de déformation, aucune précaution concernant les nouvelles coordonnées des points n'ayant été prise, les polygones ne collent plus entre eux, ou se chevauchent.

Pour résoudre ce problème, il suffit, comme dans la figure 4.5, de considérer le barycentre des ces points.

A chaque pas de déformation on calcule le taux d'énergie microscopique par unité de surface :

$$\dot{W}_{micro} =< \dot{w}_i > \tag{4.12}$$

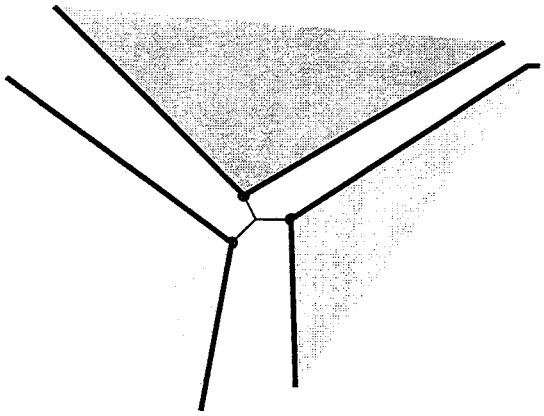


FIG. 4.5 – *Les trois points obtenus suite aux calculs des coordonnées et leur collage.*

le taux d'énergie macroscopique donnée par l'équation (4.7) où intervient la viscosité globale de l'agrégat :

$$\dot{K} = \frac{< k_i \delta_i^m >}{< \delta_i >^m} \tag{4.13}$$

4.2.2.2 Recristallisation

Comme nous l'avons précisé dans la sous-section 4.2.1.3 la condition de recristallisation est la suivante :

$$w_i \geq w_C \tag{4.14}$$

où l'énergie plastique stockée w_i est donnée par :

$$w_i = \int k_i \, \dot{\varepsilon}_i^{m+1} \mathrm{d}t \tag{4.15}$$

La recristallisation signifie la création des nouveaux grains à la place du grain i comme dans la figure 4.2. Du point de vue géométrique, nous construisons un pavage du polygone P_i toujours par une tesselation de Voronoï. Le polygone P_i est couvert de n_i petits polygones, où le nombre n_i tourne autour d'une valeur N_R fixée. Les nouveaux grains ont ainsi une structure géométrique bien précise, et leurs paramètres sont toujours gérés par les mêmes équations que les grains non-recristallisés. Pour un nouveau grain j, sa viscosité à grande déformation k_{sj} est générée aléatoirement dans un intervalle centré autour de la valeur du k_{si}, ce qui signifie que l'orientation cristallographique d'un grain obtenu par la recristallisation est proche de l'axe cristallographique du grain recristallisé.

4.3 Simulation parallèle du modèle

Nous allons présenter d'abord dans la partie 4.3.2 le code générique de cette simulation numérique en mettant en évidence son parallélisme inhérent. Ensuite nous montrerons les solutions d'implantation que nous avons proposées ainsi que l'environnement dans lequel nous avons réalisé ces simulations. Les sous-sections 4.3.4 et, respectivement, 4.3.5 seront consacrées à présenter les approches utilisées

et les résultats en termes de temps et d'espace mémoire pour la simulation de la compression plane et de la recristallisation.

4.3.1 Buts de la simulation

Toute simulation aura comme point de départ une structure géométrique, associée à des caractéristiques physiques et mécaniques. Le but est d'étudier la transformation de l'agrégat ainsi constitué lors de sa déformation à chaud. Les sorties du programme de simulations numériques sont d'une part, les représentations graphiques de l'agrégat et, d'autre part, l'évolution dans le temps de ses paramètres globaux.

4.3.2 Parallélisme du modèle

Nous allons montrer le principe de la simulation du modèle dans la première partie en mettant ensuite en évidence les problèmes qui surgissent : la synchronisation dans 4.3.2.2 et les communications dans 4.3.2.3.

4.3.2.1 Les principes de la simulation

La simulation numérique du modèle consiste d'abord en la discrétisation de la déformation avec un nombre assez important de pas :

$$d\varepsilon = \dot{\varepsilon}\, dt$$

nous allons prendre une déformation uniforme $\dot{\varepsilon} =$ constant et $dt = 1$.

A chaque pas, pour tous les grains, une compression plane est calculée puis, si l'énergie du grain dépasse le seuil fixé, on procède à une recristallisation dynamique continue.

Remarquons que les recristallisations sont des traitements exceptionnels, car un grain recristallise au plus une seule fois et les grains qui en sont issus n'atteindront jamais la condition de recristallisation. La déformation globale d'un agrégat s'écrit alors comme la figure 4.7, où la déformation d'un grain est décrit dans 4.6 et la recristallisation dans la figure 4.8.

Les pas de déformation sont successifs pour tout l'agrégat, car il faut garder la cohérence du modèle (i.e. respecter les équations 4.3 et 4.11) et il faut aussi calculer l'évolution de la géométrie de l'agrégat.

La recristallisation en elle-même demande des calculs locaux quant à la génération des nouveaux grains mais aussi des calculs autour du grain recristallisé afin de mettre à jour les voisinages.

4.3.2.2 Synchronisation. Variables globales

Nous avons déjà signalé que les pas de déformation sont synchrones, un pas de déformation ne commencera donc qu'après que le pas précédent soit fini. Le fait que l'agrégat ait une structure périodique impose qu'un pas de déformation sur un grain ne commence qu'après que le pas précédent soit fini sur tout l'agrégat. Donc il n'y a pas la possibilité de mettre en parallèle des déformations à des pas différents.

Par contre, il est fort possible d'exécuter un pas de déformation en même temps sur tous les grains. Il se pose quand même un autre problème de synchronisation : il faut assurer le calcul des variables globales $< \delta_i >$, K, $\dot{W}_{micro}$, $\dot{W}_{macro}$, surtout qu'une d'entre elles, $< \delta_i >$, doit être connue tout de suite au niveau de chaque grain afin de continuer les calculs.

Deformer_grain(i)

debut
calculer k_i (équation (4.5))
obtenir les viscosités des voisins : k_j
calculer la viscosité moyenne $\overline{k_i}$ (équation (4.9))
calculer rΣ_i et δ_i (équation (4.10))
..... calculer $< \delta_i >$
obtenir la valeur moyenne globale $< \delta_i >$
calculer $\dot{\varepsilon}_i$ (équation (4.11) et σ (équation (4.2)
calculer les coordonnées temporaires W_i^l
obtenir les coordonnées temporaires des voisins W_j^m
calculer (par la méthode du barycentre) les coordonnées finales V_i^l
..... calculer les variables globales K, $\dot{W}_{micro}$, $\dot{W}_{macro}$
calculer l'énergie stockée w_i
si $w_i \geq w_C$ **Recristalliser_grain(i)**
fin

FIG. 4.6 – *Le code de la déformation d'un grain.*

Deformer_globalement

debut
pour tout pas de déformation dt
pour chaque grain i
Deformer_grain(i)
fin

FIG. 4.7 – *Le code de la déformation globale*

Ainsi l'implantation naturelle qui consiste à faire exécuter en parallèle autant de processus de déformation que de grains ne convient pas en raison de problèmes de synchronisation.

4.3.2.3 Communications

Un examen des équations mécaniques met en évidence trois types de communications :

- pour échanger des informations avec les grains voisins (k_i, coordonnées des sommets);
- pour calculer et obtenir les variables globales;
- pour mettre à jour les voisinages suite à une recristallisation.

Dans le premier type les communications sont "symétriques" au sens où des grains voisins échangent le même type d'information. Elles sont aussi intensives, car elles se produisent pour tous les grains.

Recristalliser_grain(i)

debut
    calculer une tessellation de Voronoï du polygone P_i
    créer des nouveaux grains
    mettre à jour les voisins des nouveaux grains et de leurs voisins
fin

FIG. 4.8 – *Le code de la recristallisation d'un grain.*

Ceci est dû à la nature des équations mécaniques. En ce qui concerne les variables globales, les communications sont aussi simples à mettre en évidence : une concentration des messages et une diffusion. Les communications nécessaires à la mise à jour des voisinages sont beaucoup moins régulières; seuls les grains voisins du grain recristallisé et les nouveaux grains qui sont sur la frontière en auraient besoin.

L'implantation de ces communications sera développée dans la sous-section 4.3.3.4.

4.3.3 Environnement parallèle. Structures de données

Nous allons décrire la machine parallèle MIMD sur laquelle nous avons porté les simulations, ainsi que l'environnement de programmation. L'autre partie 4.3.3.3 sera consacrée à la présentation des structures des données choisies.

4.3.3.1 Environnement de programmation

Nous avons eu à notre disposition une machine parallèle VOLVOX-TS306 à base de transputers T805^{TM} [INM89]. C'est une architecture reconfigurable [ARC92] qui dispose de 24 transputers de travail, numérotés de T0 à T23 connectés à 4 crossbar C004^{TM} contrôlés par 6 transputers T225. 6 transputers T805, dits d'attaque, étiquetté de T24 à T29 hébergés dans des machines frontales permettent 6 accès simultanés à la VOLVOX.

Un transputer dispose de 4MBytes de mémoire et de 4 liens physiques de communication. Un transputer d'attaque peut utiliser uniquement 2 de ses liens pour se connecter avec d'autres transputers, car un lien est dédié à la connexion au bus de la machine frontale et un autre à un des T225.

Nous avons choisi de travailler avec 16 de ces 24 transputers et de les configurer en tore 4×4 (c'est à dire hypercube de dimension 4), avec un lien qui est interrompu pour insérer le transputer d'attaque, comme dans la figure 4.9.

Cette topologie a de bonnes propriétés quant à la longueur des chemins. Le diamètre est de 4 et la distance moyenne est de 2.5.

Le code de simulation a été fait en C-Inmos [man95a, man95b] et nous avons utilisé aussi les outils fournis dans le package D4414A [man95c].

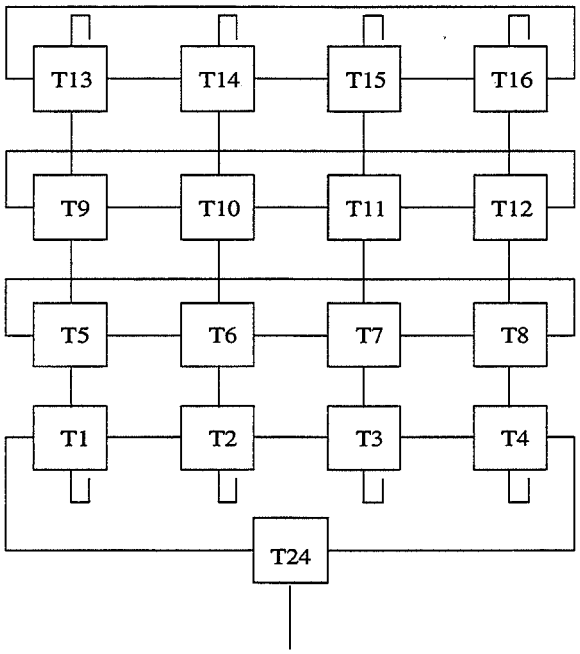


FIG. 4.9 – *La configuration choisie de la VOLVOX.*

4.3.3.2 Choix d'implantation

L'idée d'associer à chaque grain un processus chargé de le traiter semble naturelle et n'exige qu'un effort de programmation minime. Mais on ne dispose que d'un nombre très réduit de processeurs par rapport au nombre des processus (par exemple, un rapport de 1 / 100 au début qui peut arriver jusqu'à 1 / 5000 en fin de calcul pour les agrégats considérés) et que de 4 Mbytes de mémoire par processeur, insuffisant pour pouvoir lancer en parallèle un nombre de processus de l'ordre du millier.

A cause donc de cette sévère limitation de l'espace mémoire nous avons considéré une autre approche un seul processus par processeur de travail, appelons ce processus un "**Contremaître**". Chaque processus "**Contremaître**" traitera un certain nombre de grains. Sachant que au début les pas de déformation ne sont pas accompagnés de la recristallisation, c'est à dire qu'il faut calculer juste la compression, il est naturel d'assurer à tout les processus le même nombre de grains à charge.

Sur le processeur d'attaque nous avons mis aussi un seul processus qui a comme fonctions :

- faire les entrées - sorties, car le T24 est le seul processeur connecté à l'extérieur;
- calculer les variables globales et diffuser la valeur de $< \delta_i >$;
- envoyer des signaux de synchronisation avant le début de chaque pas de déformation.

Nous avons préféré calculer sur ce site les variables globales, car la plupart d'entre elles n'interviennent pas dans le calcul et sont destinées à être affichées. Seule la valeur de $< \delta_i >$ intervient dans le calcul, mais pour simplifier le code nous avons décidé que ce calcul se ferait aussi sur le processeur

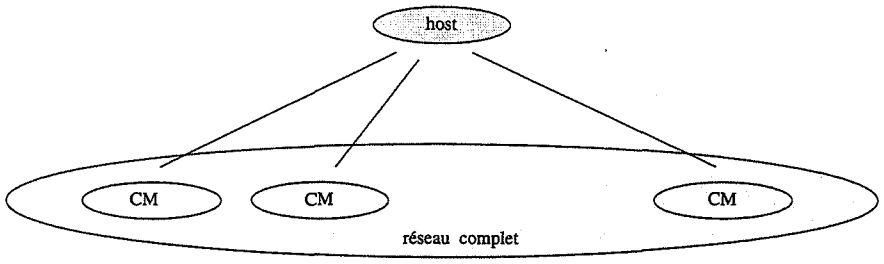


FIG. 4.10 – *Le réseau logique de processus.*

d'attaque. Quant aux signaux de synchronisation, ceux-ci n'étaient pas strictement nécessaires, mais ils rendent le code plus fiable. Nous avons appelé ce processus "**Host**" en raison des signaux envoyés et de sa relation avec un processus **Contremaître**. Chaque processus **Contremaître** est relié logiquement au processus "**Host**" et tous les processus "**Contremaître**" sont relié logiquement en réseau complet, comme dans la figure 4.10.

L'acheminement des messages sur les canaux logiques a été fait en nous appuyant sur les outils de configuration offerts par le C Toolset.

4.3.3.3 Structures de données

Présentons maintenant les choix faits pour les structures de données employées. Ceux-ci ont été dictés par les besoins de l'application, à savoir les calculs à faire et le temps d'accès aux informations.

Structure de donnée associée à un grain Pour un grain, la solution naturelle à adopter est la structure classique d'enregistrement en C, `struct`, à plusieurs champs.

Le calcul mécanico-plastique d'un grain correspond au calcul de ses valeurs physiques et mécaniques et des coordonnées des sommets de son polygone représentatif. Donc des champs de l'enregistrement associé correspondront à ces valeurs. Pour des raisons de précision du calcul toutes ces valeurs sont traitées en double précision.

Pour mémoriser les coordonnées des sommets du polygone associé plusieurs solutions sont envisageables : une liste simplement ou doublement chaînée ou un vecteur. Quand le grain est traité localement, l'ordre de traitement de ses coordonnées importe peu, mais il faut aussi envisager la possibilité d'accès direct à une coordonnée du polygone. Dans ce cas, le vecteur assure un accès en temps constant à n'importe quelle coordonnée à condition de savoir son numéro d'ordre.

En plus de ces informations, cette structure doit contenir un identificateur du grain lui-même et des informations qui permettraient de retrouver "facilement" les grains voisins.

Identification d'un grain Cet identificateur doit être unique pour tout grain de l'application et doit permettre de le retrouver. D'autre part, l'unicité de l'identificateur impose que celui-ci contienne une autre information qui permettrait de distinguer les grains localisés sur un même site.

Pour cela nous avons choisi une étiquette constituée :

· du numéro du processeur qui est chargé du traitement du grain;

· d'un numéro d'ordre du grain sur ce processeur.

Pendant la compression mécanique la localité des grains ne change pas, mais après une recristallisation il est fort probable qu'on soit amené à transférer des grains sur d'autres sites. Dans ce cas l'identificateur d'un tel grain devra changer. Nous détaillerons ce mécanisme plus loin dans la sous-section 4.3.5.

Identification des grains voisins Pendant le traitement d'un grain il faut accéder à des informations concernant ses voisins, à savoir :

- la valeur de la viscosité locale,
- les coordonnées de l'arête commune.

Les coordonnées des sommets communs sont gardées dans les deux grains sur des positions consécutives dans les vecteurs des coordonnées. Compte tenu de cette structure, il est donc nécessaire de connaître le numéro d'ordre de cette arête (du premier sommet commun).

Quant aux grains voisins il y a deux cas possibles :

- le grain voisin est situé sur le même processeur : dans ce cas un minimum de temps d'accès est obtenu en gardant, en plus du numéro d'ordre de l'arête, un pointeur vers la structure qui contient ce grain.
- le grain voisin est situé sur un autre processeur : dans ce cas, il faut connaître l'identificateur de grain de ce voisin.

Un tel identificateur de voisin sera constitué :

- du numéro de l'arête commune,
- de l'identificateur de grain,
- d'un pointeur vers la structure associée au grain (NULL, si voisin distant).

Pour chaque grain nous avons pris un vecteur d'identificateurs de voisins indexé de la même façon que le vecteur du polygone associé.

Structures adjacentes Lors d'une recristallisation on assiste à la création de nouveaux grains qui remplacent les anciens grains. Afin de réduire le temps d'insertion - suppression nous avons considéré une liste doublement chaînée des pointeurs vers les structures associées aux grains.

Dans la figure 4.13 nous avons représenté une partie d'un agrégat de grains. Les figures 4.14 et 4.15 montrent une partie des structures de données de deux processeurs en mettant en évidence la façon d'identifier les grains.

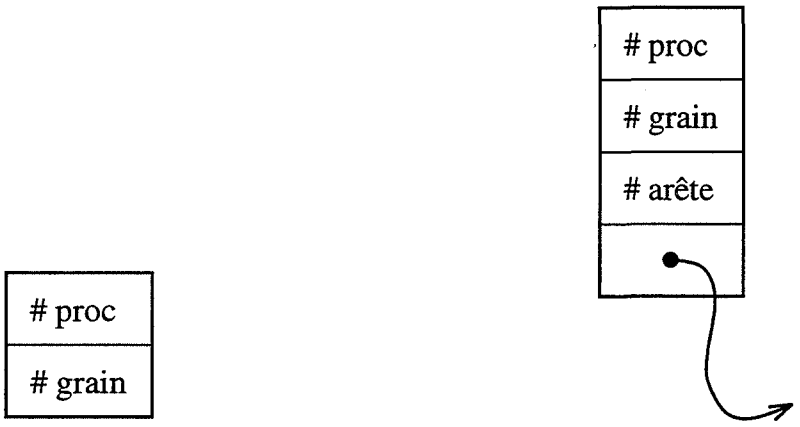


FIG. 4.11 – *Les identificateurs des grains et la structure de données utilisée pour identifier et accéder aux grains voisins.*

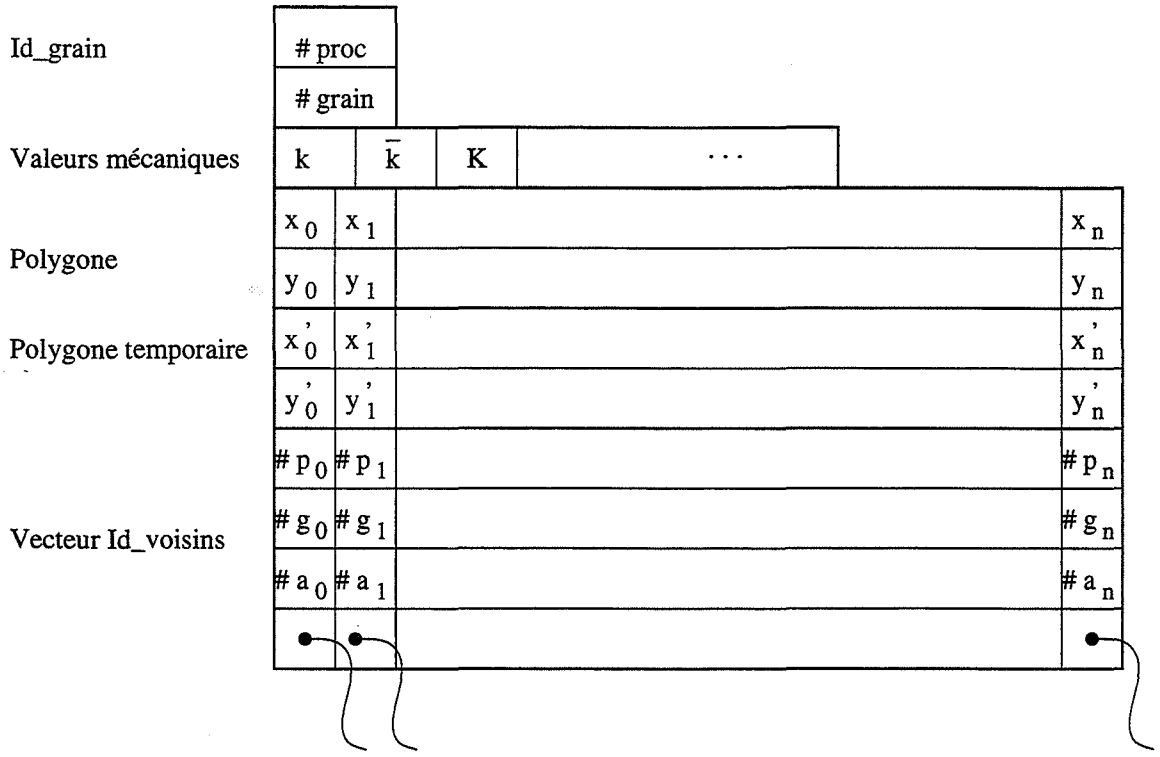


FIG. 4.12 – *La structure de données associée à un grain.*

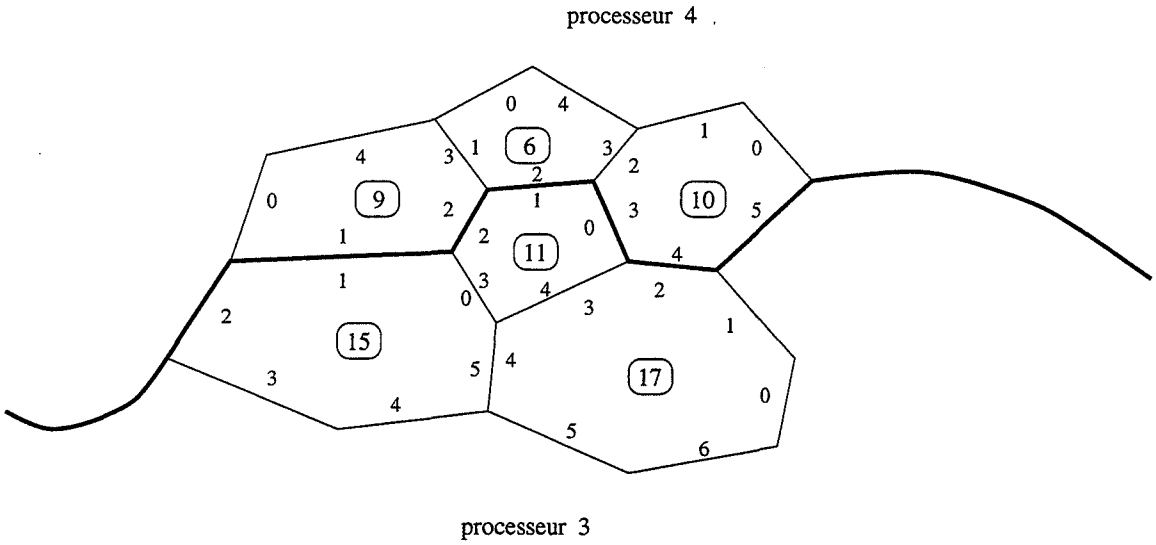


FIG. 4.13 – *Une partie de la structure géométrique d'un agrégat avec les grains et les arêtes des grains numérotés.*

4.3.3.4 Mise en œuvre des communications

Dans 4.3.2.3 nous avons détaillé les communications qui interviennent lors du traitement d'un unique grain. Dans l'implantation choisie nous avons plusieurs grains par processus et un seul processus par processeur. Les communications à mettre en œuvre restent toujours de trois types :

- pour échanger des informations avec des grains voisins situés sur des processeurs distants;
- pour le calcul des variables globales;
- pour mettre à jour le voisinage après la recristallisation d'un grain voisin distant.

Les plus intéressants à regarder quant à leur mise en œuvre sont les deux premiers types de communication.

Les variables globales sont des sommes et des rapports de sommes de caractéristiques locales d'un grain (voir les formules (4.7, 4.12, 4.13)), d'où l'idée de faire des sommes partielles de ces valeurs pour les grains alloués à un processus **Contremaître** et de les envoyer au processus **Host**.

Le nombre d'arêtes communes entre deux processus **Contremaître** peut devenir important (jusqu'à 150 pour un agrégat à 3000 grains), donc envoyer dans un même intervalle de temps des centaines de messages dans tout le réseau peut s'avérer extrêmement pénalisant. Nous avons fait appel à un seul envoi entre les processus **Contremaître** qui détiennent des grains voisins. Au fur et à mesure que les calculs demandent des informations distantes, ces demandes étant symétriques, nous mettons les informations locales avec l'identificateur du voisin correspondant dans une zone tampon qui sera envoyée dans une seule communication. Ces envois massifs se passeront à peu près simultanément, le noyau de routage les réalise correctement, sans interblocage.

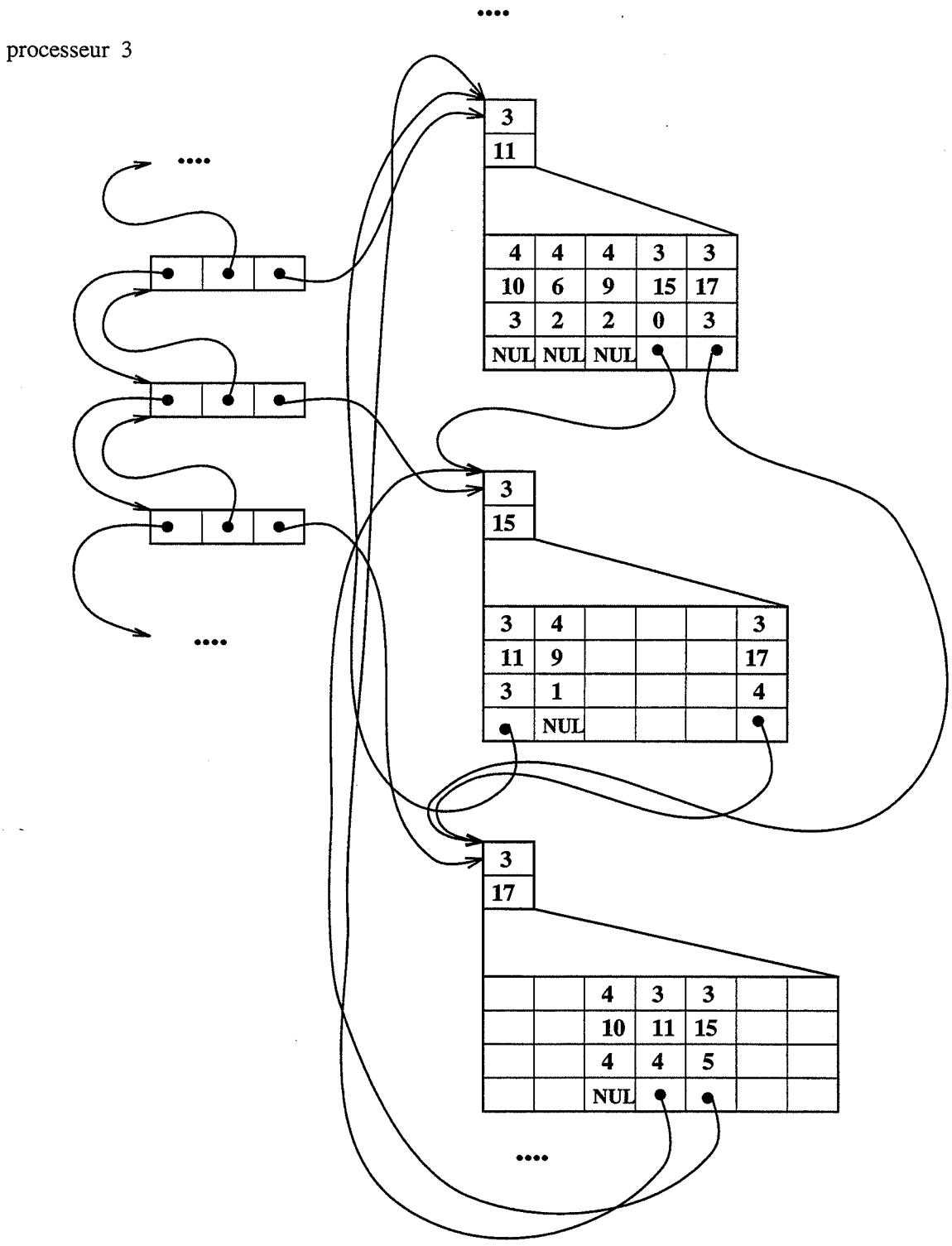


FIG. 4.14 – *Une partie des structures des données du processeur 3 (figure 4.13).*

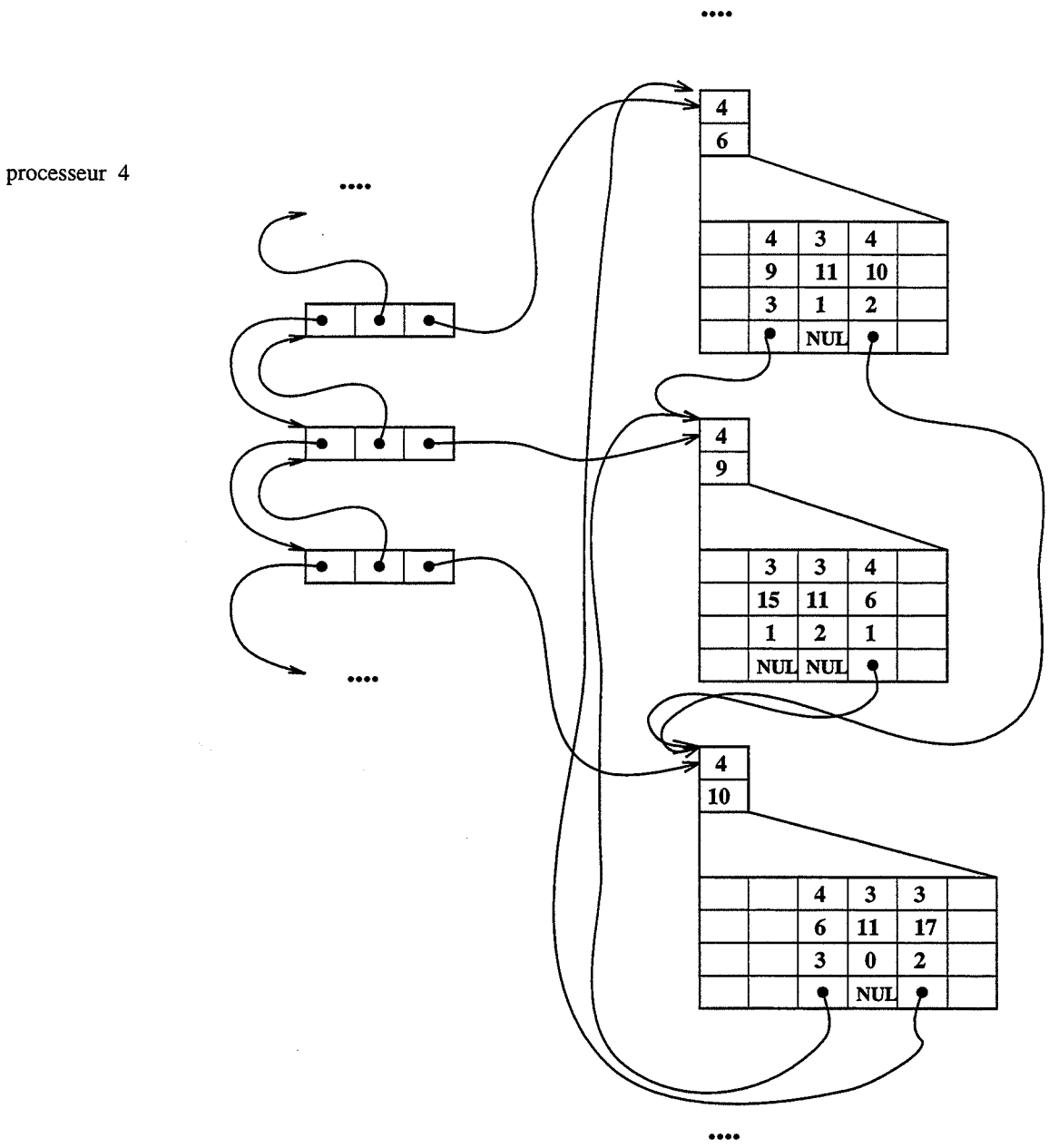


FIG. 4.15 – *Une partie des structures des données du processeur 4 (figure 4.13).*

4.3.4 Déformation (Compression)

Dans une première étape des simulations, nous nous sommes penchés sur la simulation du phénomène de compression uniquement. La motivation étant de valider le modèle proposé lors de cette simulation, nous avons été confrontés à un problème d'allocation statique assez intéressant, développé dans 4.3.4.1. Dans la partie 4.3.4.2 nous présenterons les mesures de temps effectuées en mettant en évidence l'accélération de calcul obtenue par rapport au calcul séquentiel.

Les simulations partent d'une structure périodique telle que celle de la figure 4.3, générée par une tesselation de Voronoï[4].

Outre les représentations graphique des transformations subies par un agrégat, telles que la figure 4.16, le but de cette simulation a été aussi de calculer divers paramètres de l'agrégat et de suivre leurs variations pendant toute la déformation ou par rapport à tous les grains de l'agrégat. Toujours dans le but de comprendre le phénomène mécanique par le biais de la simulation, nous avons reproduit les calculs en prenant à la place de l'équation d'écrouissage (4.5) des valeurs constantes pendant la déformation pour la viscosité d'un grain. Les résultats mécaniques font le sujet d'une section à part 4.4.

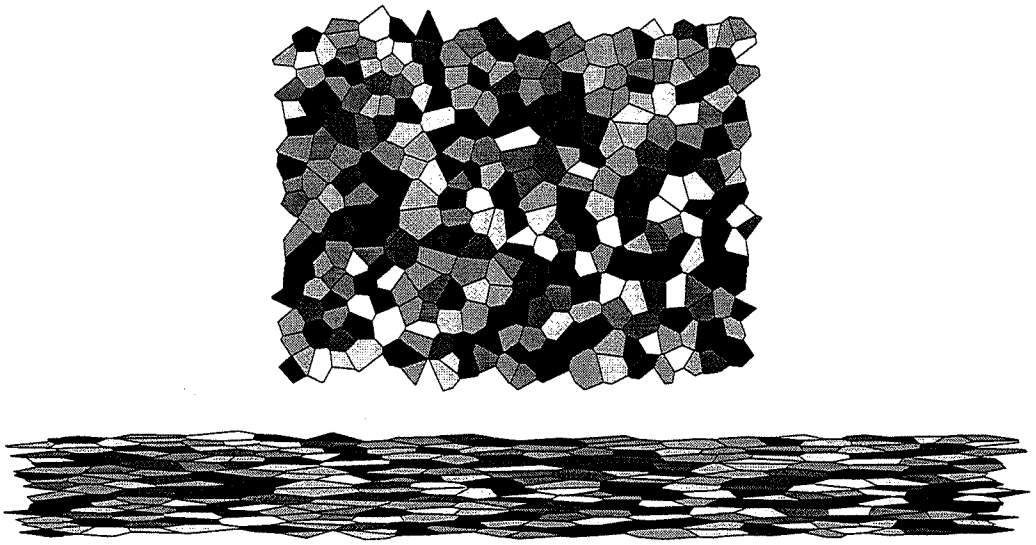


FIG. 4.16 – *La compression d'une structure de 311 grains avec* $\varepsilon = 1$, $\dot{\varepsilon} = 0.01$.

4.3.4.1 Problème d'allocation

La simulation de la compression consiste à appliquer des pas successifs de déformation. La structure de l'agrégat en terme de géométrie reste inchangée pendant cette simulation, donc le traitement d'un grain ne change pas d'un pas de simulation à l'autre. Nous avons affaire à une application quasi-statique dont les performances dépendent essentiellement de l'allocation initiale des grains aux processeurs. Nous allons décrire ce problème d'allocation et la solution que nous avons choisie.

4. La fonction de génération de la structure géométrique de base a été réalisée par Claire Maurice.

Le temps d'un pas de compression étant "toujours" le même, optimiser le temps de simulation revient à minimiser le temps d'un pas de déformation (compression) de l'agrégat.

Soit $T_{pas\_agregat}$ ce temps et $T_{pas(k)}$ le temps d'un pas de compression sur le processeur k. Soit $\gamma : Grains \rightarrow \{1, 2, \cdots 16\}$ la fonction d'allocation des grains. L'objectif est alors :

$$\min_{\gamma - \text{allocation}} T_{pas\_agregat}$$

où :

$$T_{pas\_agregat} = \max_{k \text{ processeur}} T_{pas(k)} \tag{4.16}$$

Le traitement des grains impose des calculs et des communications. Le volume de calcul dépend essentiellement du nombre des grains alloués au processeur. Concernant les communications, il y a deux types distincts (voir les sous-sections 4.3.2.3, 4.3.3.4); les communications concernant le calcul des variables globales ont toujours le même volume (des concentrations et des diffusions [dR95]), indépendant du nombre de grains alloués et du numéro du processeur. Les communications faites au niveaux des processus **Contremaître** sont des communications entre tous les processeurs connectés en réseau virtuel complet. Leur volume n'est pas facile à exprimer, d'une part il s'agit d'un échange total d'information [dR95] d'autre part, nous utilisons un noyau de routage standard, sans avoir des connaissances des longueurs des chemins.

Il y a donc deux critères suffisants d'allocation statique qui ressortent :

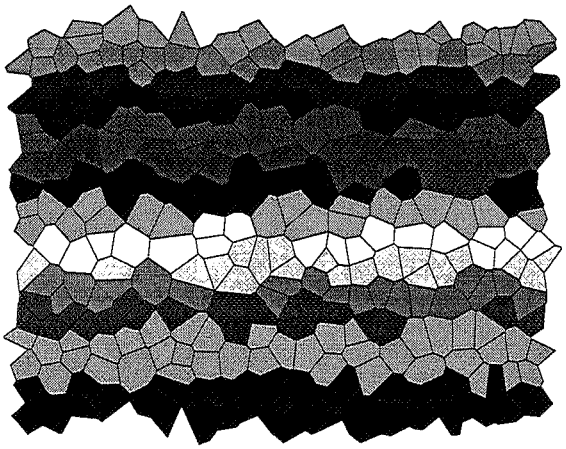
- assurer le même nombre de grains sur chaque processeur;
- garder la localité des grains : les grains voisins sont alloués soit sur le même processeur, soit sur des processeurs directement connectés.

Le deuxième critère engendre des communications à distance 1 uniquement, donc il n'y a pas de conflits d'accès aux liens de communication lors des envois, et dans ce cas le temps de communication est directement proportionnel à la taille de la frontière commune entre les processeurs. Le principe de localité se traduit aussi par l'allocation des parties de structure *contiguës* : tout couple de grains situés dans une même partie peut être relié par un chemin des grains voisins et si un grain a tout ses voisins situés dans une même partie, lui aussi est situé dans cette partie.

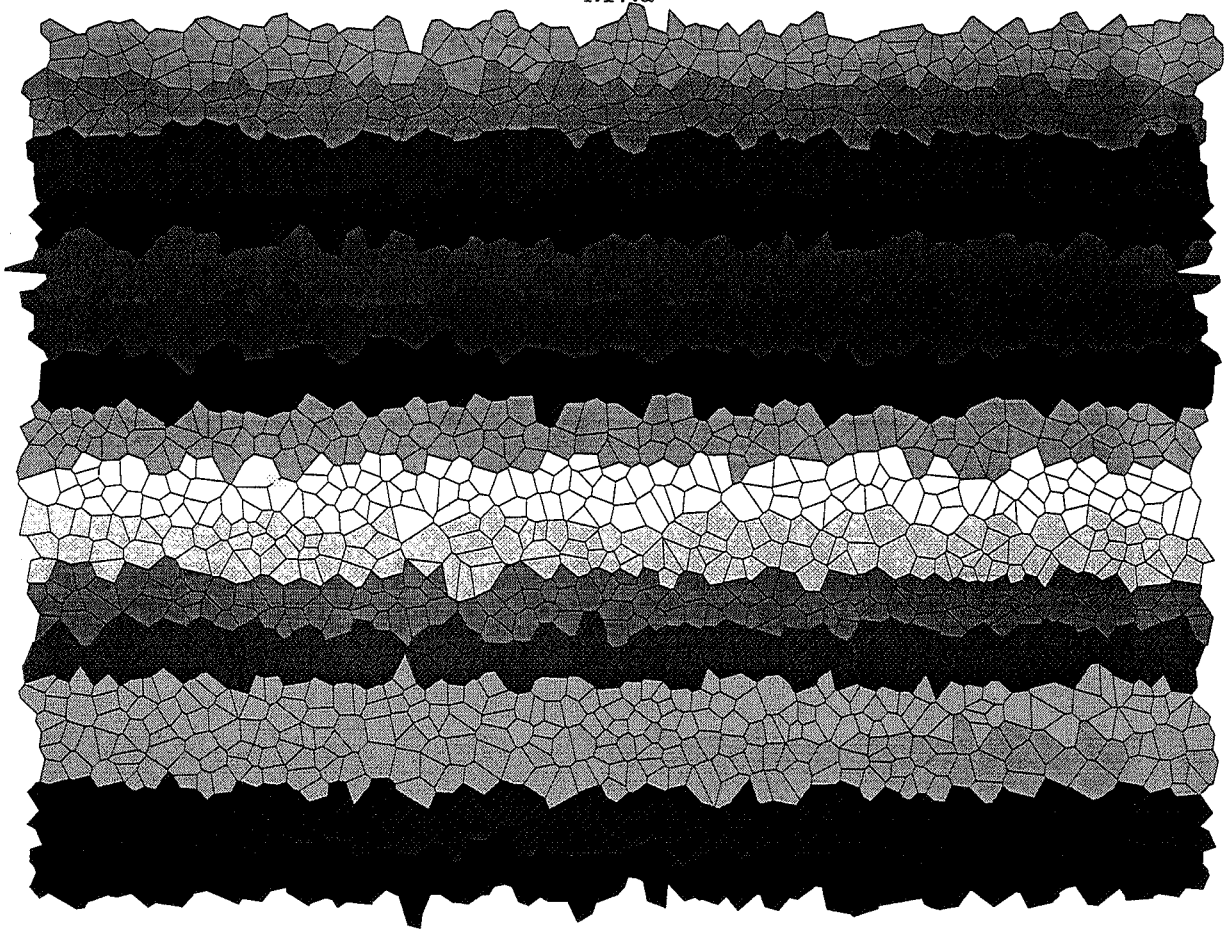
Les problèmes du partitionnement d'un graphe en sous-graphes sous des contraintes de limitation de frontière sont, en général, NP-complets [BB87], [GJ79]. Dans notre cas la topologie de l'agrégat et le réseau d'interconnexion des processeurs peuvent être plongés[5] sur une surface torique, de plus nous avons 2^4 processeurs, donc ce problème semblerait faisable à l'aide d'une méthode récursive.

Rappelons que notre but est non pas de faire une implantation parfaite de la simulation de la compression, mais d'étudier le comportement de l'allocateur dynamique lors de la simulation de la recristallisation. Ainsi, nous avons choisi une solution d'allocation peu coûteuse à calculer et qui assure pour des topologies de grande taille (plus de 500 grains) des parties contiguës allouées aux processeurs. Nous avons, pour cela, alloué les grains dans l'ordre de génération de leurs polygones, c'est à dire dans l'ordre croissant de la coordonnée en x_2 du centre (de triangulation) du polygone. Pour une structure à 311 grains cette allocation est indiquée dans la figure 4.17.a et pour une structure à 2134 grains dans la figure 4.17.b. On peut constater que pour la structure à 311 grains, les grains alloués sur un processeur ne forment pas encore une partie contiguë.

5. Voir l'annexe A pour la définition du plongement de graphes.



4.17.a



4.17.b

FIG. 4.17 – *L'allocation des grains calculée par la méthode Ordre-génération pour une topologie à 311 grains et, respectivement, 2134 grains.*

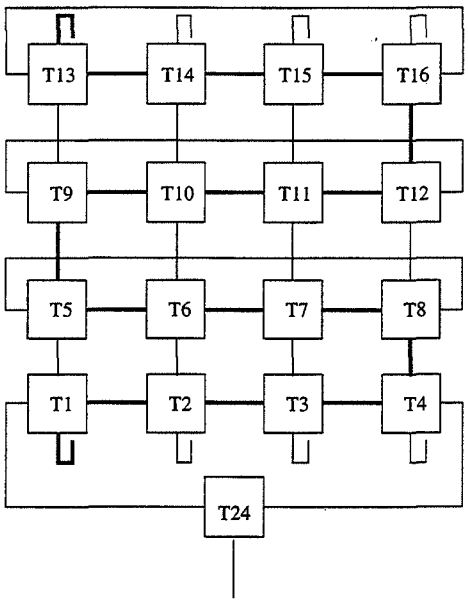


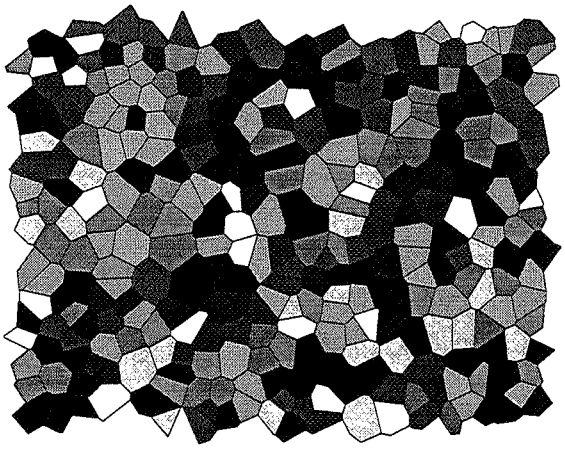
FIG. 4.18 – *Le plongement du circuit à 16 nœuds sur le réseau d'interconnexion, utilisé pour la méthode Ordre-génération2.*

Le temps d'allocation est en $O(N)$, N étant le nombre de grains de l'agrégat. Appelons cette méthode *Ordre-génération*. Même si l'allocation se fait en parties contiguës, le principe de localité n'est pas toujours respecté. Du fait de la structure régulière de la topologie initiale de l'agrégat, les parties allouées sont dans une relation circulaire de voisinage, d'où l'idée de chercher un circuit de longueur 16 qui se plonge sans dilatation sur le réseau en tore du réseau d'interconnexion des processeurs et d'allouer ainsi des parties voisines de l'agrégat sur des processeurs voisins. La figure 4.18 donne un plongement du circuit sur le tore 4×4, ce circuit est utilisé pour cette version de la première méthode d'allocation, appelée *Ordre-génération2*. Le temps nécessaire est toujours en $O(N)$.

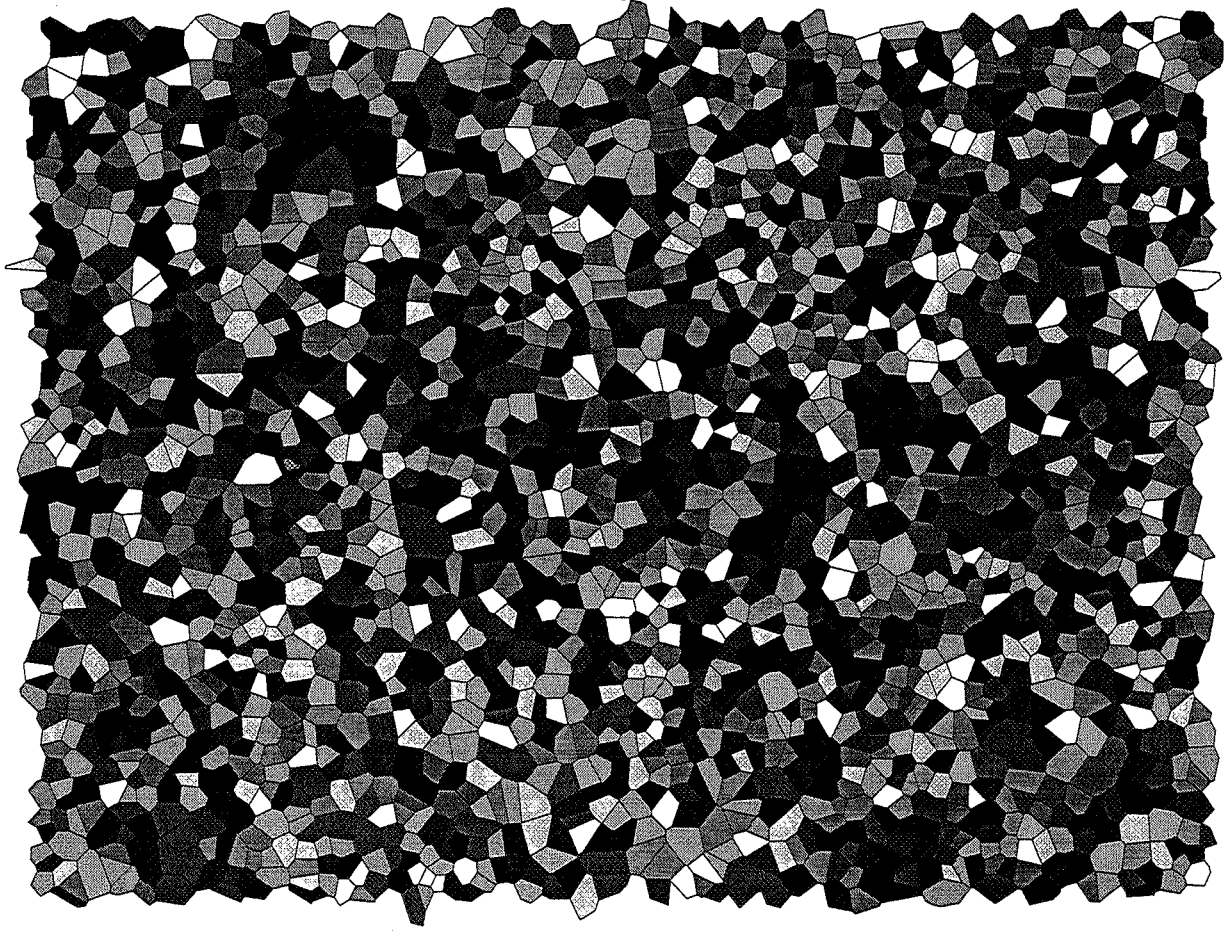
Afin de se faire une idée de l'efficacité de cette méthode simple nous avons aussi implanté une méthode *Aléatoire* qui consiste à allouer les grains de façon aléatoire sous la restriction qu'un processeur n'a jamais plus de $\left\lceil \frac{N}{16} \right\rceil$ grains. Dans la figure 4.19 deux allocations aléatoires sont représentées.

A cause de la taille importante de la structure de données associée à un grain et de la limitation à 4Mbytes pour la mémoire d'un transputer T805, nous avons testé les méthodes d'allocation statique pour des topologies qui ne dépassent pas 2200 grains. Le tableau 4.1 donne le temps (en secondes) nécessaire pour un pas de compression en fonction de la topologie initiale (nombre de grains) et de la méthode d'allocation employée. Il est facile de constater que les deux variantes *Ordre-génération* et *Ordre-génération2* donnent presque les même performances, *Ordre-génération2* est meilleure de 0.4 - 1%. *Ordre-génération* est nettement meilleure (de 21% à 56%) que la méthode *Aléatoire*, où il n'y a aucune contrainte de localité respectée.

Notons que les développements qui suivront pour la simulation de la recristallisation utilisent la



4.19.a



4.19.b

FIG. 4.19 – *Une allocation des grains calculée par la méthode Aléatoire.*

Méthode / Taille	57	60	311	1070	2134
Ordre-génération	0.0433	0.04055	0.10009	0.28768	0.56777
Ordre-génération2	0.04306	0.04039	0.09969	0.28327	0.56088
Aléatoire	0.0525	0.05437	0.13603	0.39635	0.86707

TAB. 4.1 – *Le temps d'un pas de simulation de la compression en fonction de la taille de l'agrégat et de la méthode d'allocation statique utilisée.*

méthode d'allocation la plus simple à mettre en œuvre et qui ne requiert pas de l'espace mémoire supplémentaire, à savoir la méthode *Ordre-génération.*

4.3.4.2 Accélération de la simulation

Afin de mesurer l'efficacité de la mise en œuvre de cette simulation sur une architecture parallèle, nous avons aussi implanté le code séquentiel sur un seul transputer de la machine parallèle avec quasiment les mêmes fonctions et librairies et le même séquencement des opérations.

Si on suppose que l'accès mémoire aux informations concernant les grains est constant, l'implantation séquentielle d'un pas de simulation est en $O(N)$ [6], car si le nombre des grains est N, le nombre des sommets des polygones est $\frac{3}{2}N$ et le nombre d'arêtes $3\,N$ (dû à la périodicité de la structure). Donc :

$$T_{seq} = C\,N \tag{4.17}$$

Comme nous l'avons remarqué auparavant(dans la sous-section 4.3.4.1) le temps d'un pas de simulation englobe le temps de calcul des paramètres géométriques et mécaniques des grains (T_{calcul}), le temps de communication pour le calcul et la diffusion des variables globales (notons-le $T_{globale}$, toujours constant) et le temps de communication entre processus **Contremaître** (notons-le T_{comm}). Nous avons donc la relation suivante pour un processeur k :

$$T_{pas(k)} = T_{calcul(k)} + T_{globale} + T_{comm(k)}$$

On peut toujours supposer que l'accès aux informations se fait toujours en temps constant, ce qui signifie : $T_{calcul(k)} = C_2\,N(k)$, $N(k)$ étant le nombre de grains alloués au processeur k. On a donc, d'après la relation 4.16 :

$$T_{par} = \max_k T_{pas(k)}$$

On peut donc déduire que :

$$T_{par} = C_2\,\frac{N}{16} + T_{globale} + T_{comm}(\gamma) \tag{4.18}$$

où $T_{comm}(\gamma)$ est le temps des échanges entre tous les processus **Contremaître** et dépend de l'allocation γ choisie.

Selon le principe d'Amdal, dans le cas général l'accélération d'une implantation parallèle ne dépassera pas 16, le nombre de processeurs. Dans notre cas, l'accélération obtenue croît avec la taille

6. Nous avons toutefois remarqué que pour un nombre important de grains l'accès mémoire n'est plus constant, la linéarité étant satisfaite pour $N \geq 2000$

Taille de l'agrégat	60	311	1070	2134	3847
Temps séquentiel	0.143	0.743	3.415	5.100	12.242
Temps parallèle	0.0375	0.083	0.285	0.448	1.017
Speedup	3.82	8.89	11.98	11.36	12.04

TAB. 4.2 – *Les temps de simulation et l'accélération obtenus pour la méthode Ordre-génération.*

de l'agrégat, mais elle semble bornée par une valeur de 12.05 pour la méthode d'allocation *Ordre-génération.* Ceci s'explique par le fait que les communications entre **Contremaîtres** $T_{comm}(\gamma)$ ont tendance à croître avec la taille de l'agrégat et dépend toujours de la méthode d'allocation. Donc le rapport d'accélération $\frac{T_{seq}}{T_{par}}$ qui est égal à (selon les équation 4.17 et 4.18) :

$$\frac{C\,N}{\frac{C_2\,N}{16} + T_{globale} + T_{comm}(\gamma)}$$

devient :

$$\frac{C}{\frac{C_2}{16} + \frac{1}{N}T_{globale} + C_3} \longrightarrow \frac{16\,C}{C_2 + C_3} \text{ quand } N \to \infty$$

où C_2 est légèrement plus grande que C, C_3 est une constante qui dépend de la distribution des grains (i.e. du nombre des arêtes communes à partager entre processeurs) et des performances de la couche de routage de la machine. Donc ce rapport est toujours borné par une valeur plus petite que 16. Pour notre implantation, on estime cette borne à 12.5.

Le tableau 4.2 donne les temps de simulation en séquentiel et en parallèle, ainsi que l'accélération obtenue.

4.3.5 Recristallisation

Dans la sous-section précédente 4.3.4, nous avons mis en évidence le phénomène de compression des grains produit lors de la déformation à chaud, en ignorant volontairement le phénomène de recristallisation dynamique continue qui aurait pu se produire.

4.3.5.1 Implantation de base

Simulation d'une recristallsation Telle que la recristallisation a été définie dans la section 4.2.2.2, à chaque pas de compression, après avoir calculé l'énergie cumulée w_C, si la condition de recristallisation (4.14) est satisfaite, il faut procéder à la création de nouveaux grains issus des grains qui recristallisent et ensuite il faut assurer la cohérence des données.

Créer de nouveaux grains à partir d'un "vieux" grain signifie :

- construire un pavage du polygone du grain initial par des petits polygones;
- pour chaque polygone du pavage, considérer le grain associé;

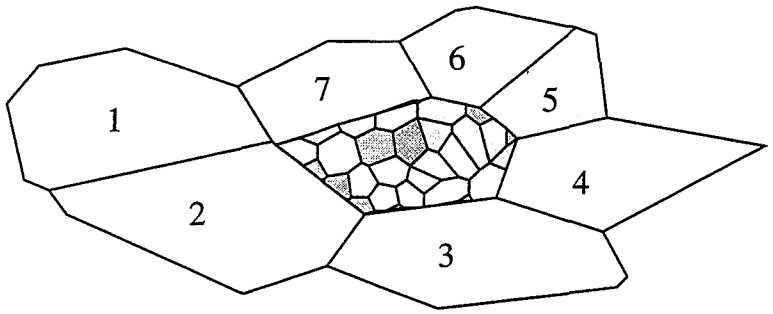


FIG. 4.20 – *Un grain qui recristallise.*

– mettre au point pour chaque nouveau grain son voisinage.

Pour la construction du pavage du polygone initial, nous avons fait appel à une fonction[7] qui construit un pavage à l'aide d'une tesselation de Voronoï. Le nombre des polygones du pavage est aléatoire autour d'une valeur fixée. Dans toutes les simulation, nous avons des pavages qui comportent entre 28 et 42 polygones. Signalons que cette fonction est très laborieuse et demande un espace mémoire important (au moins 2.5 Kbytes); par ailleurs, un appel de cette fonction coûte au moins 6 secondes.

A partir de la construction géométrique, un nouveau grain est obtenu facilement en lui associant les caractéristiques mécaniques héritées du grain recristallisé. Notons aussi que tout pavage construit ne sera pas accepté automatiquement pour la construction des nouveaux grains, car il faut assurer que chaque sommet d'un polygone de pavage soit un point triple.

En ce qui concerne la cohérence de données, il faut assurer, après la recristallisation, que chaque grain connaisse ses voisins directs ainsi que l'indexation des arêtes commune. Pour les grains nouveaux créés comme dans la figure 4.20 il y a deux types de frontières :

– celles qui se situent à l'intérieur du polygone recristallisé;

– celles qui se trouvent sur les arêtes de l'ancien polygone.

Dans le premier cas, déterminer la frontière est une opération "interne" au polygone recristallisé. Dans le deuxième cas, la frontière est déterminée par les nouveaux points et l'ancienne frontière entre le polygone recristallisé et ses voisins.

Pour un grain voisin d'un grain recristallisé, la frontière change car une arête est remplacée par plusieurs arêtes de nouveaux grains. Ces grains voisins changent aussi leurs nombre de sommets et de ce fait l'indexation des arêtes. La nouvelle indexation doit être connue par tous leurs voisins[8]. Donc la recristallisation d'un grain se reflète jusqu'aux voisins situés à distance 2.

Il est fort probable que lors d'un pas de compression plusieurs grains recristallisent et donc que deux grains voisins recristallisent en même temps comme dans la figure 4.21 et que plusieurs voisins d'un grain recristallisent en même temps comme dans la figure 4.22. Dans ce cas la mise à jour pour la fontière de ce point doit se faire de façon cohérente.

7. La fonction a été écrite par Claire Maurice, centre SMS, Ecole de Mines de St. Etienne.

8. Nous rappelons que pour chacune de ses arêtes i un grain doit connaître l'indexe j utilisé par son voisin qui la partage.

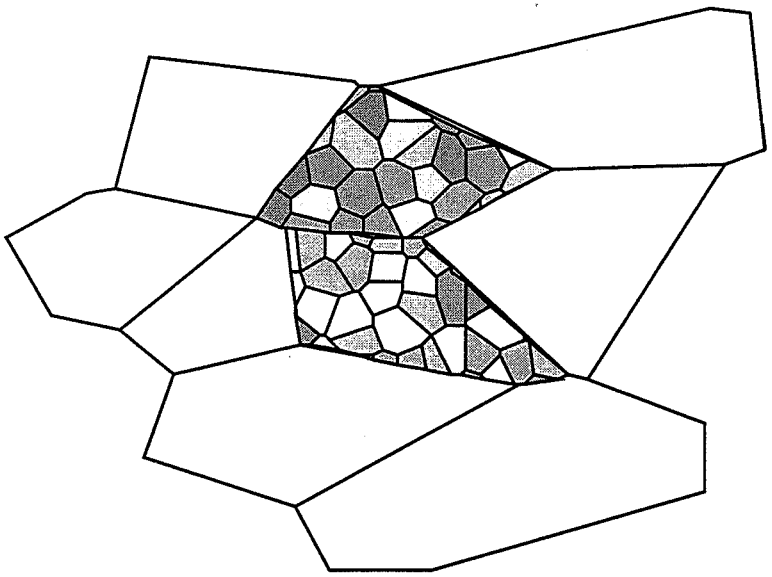


FIG. 4.21 – *Deux grains qui recristallisent.*

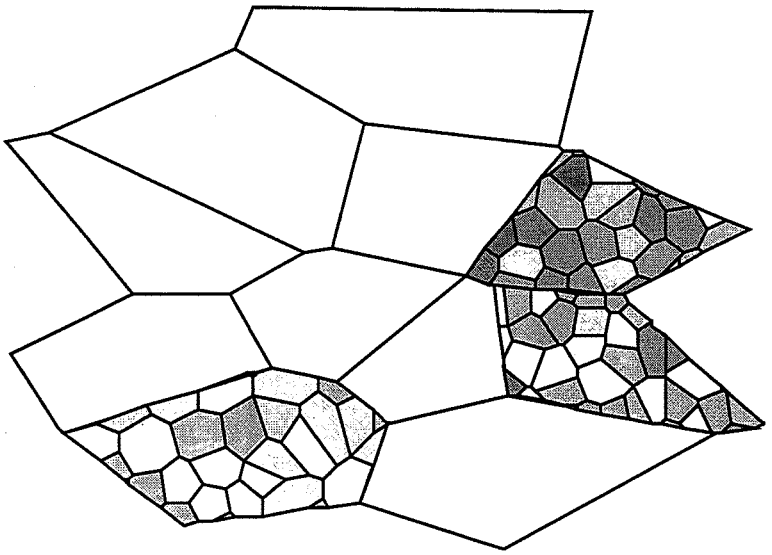


FIG. 4.22 – *Un grain dont plusieurs voisins recristallisent.*

Lors d'un pas de compression, si au moins une recristallisation se produit, après avoir créé tous les nouveaux grains, une mise à jour des voisinages est impérative. Afin de réaliser ceci de manière cohérente et unitaire, nous avons mis en place les quatre pas suivants :

- chaque grain qui a recristallisé informe chacun de ses anciens voisins sur les transformations produites sur leur frontière commune;
- chaque grain centralise l'information reçue en reconsidérant sa frontière et en indexant correctement ces sommets (arêtes);
- chaque grain (nouveau ou pas) envoie la nouvelle indexation à son voisin;
- chaque grain enregistre au bon endroit les index reçus.

Implantation séquentielle. Implantation parallèle Dans un premier temps nous avons implanté la version séquentielle de la déformation. Dû à la mémoire limitée d'un transputer et du fait que le code de la construction d'un pavage de polygone demande une certaine quantité de mémoire dynamique, pas plus de 80 recristallisations peuvent être simulées. Pour l'implantation sur plusieurs processeurs la limitation du nombre de recristallisations sur un seul processeur est encore plus stricte à 50 grains, car une partie de la mémoire est utilisée pour la communication. De ce fait, nous n'avons pu pas faire des mesures pour des agrégats de taille supérieure à 57 grains en séquentiel et 492 grains en parallèle.

Nous avons vu que la parallélisation du code de compression était bénéfique pour l'accélération du calcul. Le calcul de recristallisation d'un seul grain est extrêmement coûteux (un rapport de l'ordre de 10^3 minimum entre le calcul de recristallisation et celui de compression d'un grain). D'autre part, lors d'un pas de déformation, plusieurs grains peuvent demander des calculs de recristallisation et ces calculs peuvent se faire dans n'importe quel ordre, seule la mise à jour du voisinage requiert la synchronisation. Donc si ces grains sont répartis entre différents processeurs, le gain de temps est évident. Le tableau 4.3 montre les temps totaux de calcul et les accélérations obtenus lors des simulations de la déformation-recristallisation pour une déformation globale de $\varepsilon = 1$ et pour des agrégats de taille 22 et 57 grains. Les paramètres choisis font que les recristallisation de grains se produit relativement tôt pour des déformations comprises entre 0.3 et 0.46. Les valeurs de l'accélération de calcul pour ces agrégats de petite taille sont plus grandes que les valeurs de l'accélération obtenus pour la simulation de la déformation sans recristallisation.

Notons aussi que si les résultats mécaniques obtenus pour la déformation sans recristallisation étaient identiques pour l'implantation séquentielle et parallèle, ce n'est plus exactement le cas à partir du moment où les recristallisations des grains commencent, car la génération d'un pavage se fait aléatoirement. Toutefois, les résultats finaux ne varient pas beaucoup d'une version à l'autre, la structure de l'agrégat final n'est pas la même, mais elles se ressemblent[9]. De plus, les paramtres mécaniques telle que la viscosité globale varient très peu (de ordre de 10^{-3}). Les même variations sont obtenues pour les exécutions du même code (séquentiel ou parallèle) pour des valeurs différentes du noyau de génération des nombres aléatoires[10].

9. Dans la partie 4.4.1.3 nous verrons que les nouveaux grains gardent la forme de l'ancienne frontière.

10. Les résultats qui seront présentés sont des valeurs moyennes obtenues à partir des différents noyaux de génération de nombres aléatoires.

Taux de déformation	10^{-1}	10^{-2}	$0.5 * 10^{-2}$	10^{-3}
Temps séquentiel	158.5879	261.6769	179.1173	209.2662
Temps parallèle	49.5739	38.9565	41.8845	65.3095
Speedup	3.1990	6.7171	4.2765	3.2042

Taux de déformation	10^{-1}	10^{-2}	$0.5 * 10^{-2}$	10^{-3}
Temps séquentiel	416.9491	437.3921	428.4771	523.2542
Temps parallèle	88.7609	66.1269	64.9649	91.3317
Speedup	4.6974	6.6144	6.5955	5.7292

TAB. 4.3 – *Les temps totaux (en secondes) de calcul pour l'implantation séquentielle, respectivement parallèle, et les accélération de calcul obtenus pour des agrégats de 22 et, respectivement, 57 grains pour une déformation globale de $\varepsilon = 1$, seuil de l'énergie de recrisatallisation de $w_C = 0.57 * 10^7$ et pour différentes valeurs du taux de déformation.*

4.3.6 Allocation dynamique lors de la simulation de la recristallisation

Nous avons vu que le processus de recristallisation appliqué à un grain génère un certain nombre de nouveaux grains qui peuvent engendrer un déséquilibre des états de charge des processeurs. Nous allons donc appliquer l'algorithme générique de régulation de charge proposé dans le chapitre 3 pour l'application réelle de simulation de la déformation à chaud des agrégats polycristallins. Cette application est une application dynamique qui se prête parfaitement à l'application d'une régulation de charges.

4.3.6.1 Préliminaires

Les simulations que nous avons faites avec les deux premières versions de l'application (séquentielle et parallèle) nous ont montré que lors d'une déformation à chaud, soit il n'y pas de recristallisation, soit les recristallisations se produisent sur tous le grains de l'agrégat pendant un laps de temps relativement court : 1 ou 2 pas d'itération pour un taux de déformation $\dot{\varepsilon} = 0.1$, au maximum 10 pas pour $\dot{\varepsilon} = 0.01$, au maximum 40 pas pour $\dot{\varepsilon} = 0.001$[11]. Pendant cet intervalle de temps, à chaque pas d'itération, chaque processeur calcule un nombre variable de recristallisations, mais il est impossible de prédire ce nombre.

Une fois les recristallisations de grains entamées, les processeurs commencent à avoir des variations de charge sensibles les unes par rapport aux autres; par exemple, le nombre total de grains sur chaque site s'accroît, mais il est différent d'un site à un autre. Dans la figure 4.23, nous présentons à titre d'exemple la variation de charge d'un pas à l'autre pour un agrégat de 492 grains soumis à une déformation de $\varepsilon = 1$ avec un taux de déformation de $\dot{\varepsilon} = 10^{-2}$. Nous avons choisi de montrer la variation de la charge moyenne, de l'écart type std(*charge*) et aussi de la valeur max(*charge*) – min(*charge*).

Notre application est donc composée de trois grandes étapes successives :

– compressions uniquement, chaque pas d'itération coûte toujours la même chose en termes de

11. Pour plus de détails voir la partie 4.4.1.3 qui présente la signification des résultats mécaniques.

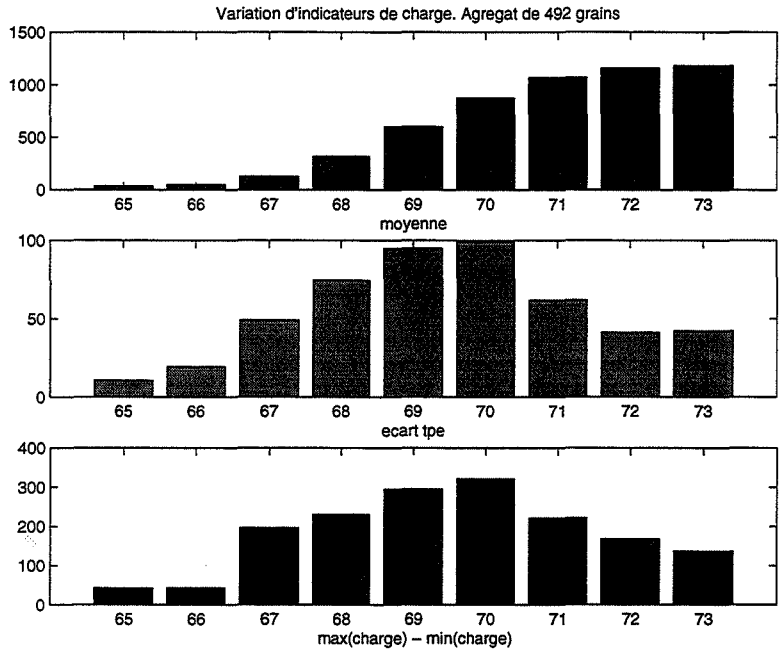


FIG. 4.23 – *La variation des différents indicateurs de charge pendant la recristallisation d'un agrégat de 492 grains, taux de déformation de* $\dot{\varepsilon} = 10^{-2}$.

temps et d'espace mémoire;

- compressions et recristallisations, un pas d'itération débute avec le calcul de compression, comporte un nombre variable de calculs de recristallisation et puis une étape de synchronisation qui permet de reconstituer le voisinage des grains, son coût n'est pas prévisible;

- compressions des nouveaux grains, les pas d'itération sont de nouveaux semblables.

L'allocation de charge initiale qui assurait à chaque processeur le même nombre d'unités de charge permet d'optimiser le temps de la première étape. Pour cette application, il est évident qu'un algorithme de régulation de charge peut être largement bénéfique pour le temps de la dernière étape de l'application. Nous avons adapté l'algorithme proposé dans la partie 3.6 pour réguler la charge des processeurs afin d'améliorer le temps total d'exécution de l'application.

4.3.6.2 Politiques d'initiation et décision

Si au départ on assure à chaque processeur la même charge (i.e. le même nombre de grains à traiter), l'apparition des premières recristallisations des grains génère un déséquilibre évident. La question qui se pose pour tout algorithme de régulation est d'abord quand faire appel à la régulation? Dans le cas de notre application deux réponses sont possibles :

- à la fin de toutes les recristallisations;

- pendant le laps de temps où des recristallisations se produisent.

Nous avons choisi de ne faire la régulation qu'après que toutes les recristallisations soient finies. Nous avons écarté le deuxième choix parce que durant les itérations de recristallisation deux catégories d'unités de charge co-existent : les nouveaux grains créés suite à des recristallisation et les grains de départ qui ne sont pas encore recristallisés. Même si on assure un équilibre entre les processeurs pour ces deux catégories, plus particulièrement pour les "vieux" grains, il est fort probable de faire de nouveau appel à la régulation de charge, le temps d'exécution varie beaucoup d'un processeur à l'autre; d'une part, les grains recristallisent de façon aléatoire et, d'autre part, le temps de calcul d'une seule recristallisation n'est pas prévisible et il est plus important que le temps de calcul de toutes les compressions. Autrement dit, le fait qu'un processeur dispose pendant cette étape du plus grand nombre de grains ne signifie pas que c'est lui qui aura besoin du plus grand temps de calcul.

Nous proposons une légère adaptation de l'algorithme initial de régulation et ce sont la politique d'initiation et la politique de décision qui doivent tenir compte de l'existence des deux catégories d'unités de charges. Donc uniquement les processeurs qui viennent de finir toutes les recristallisations sur leur site peuvent initier la régulation. La décision de régulation se fait dans la racine de l'arbre recouvrant de profondeur minimale et ne sera prise que si tous les processeurs ont fini les recristallisations.

Un processeur v qui a fini les recristallisations (i.e. il ne dispose que des grains nouveaux) compare sa propre charge au seuil supérieur adaptable Sup:

$$charge[v] > Sup$$

Au départ ce seuil a la valeur :

$$Sup = C_S * \overline{charge}$$

où $C_S > 0$ est une constante que nous avons choisie égale à 1.1. Il est fort probable que la décision prise soit négative, car les recristallisations s'étalent pendant plusieurs pas d'itération. L'initiation de la régulation est suivie d'une remontée d'information vers la racine de l'arbre, de la prise de décision et de la diffusion de celle-ci. La diffusion de la décision s'accompagne de la diffusion des informations qui permet de mettre à jour la valeur de la borne Sup :

$$Sup = C_S * \left(\overline{charge} + \frac{\tau}{\theta} * \mathsf{std}(charge) * \frac{-\ln(-\ln p) - b}{C} \right)$$

où :

- p est une probabilité constante, nous avons choisi $p = 0.8$;
- b et C sont les constantes de la loi de Gumbel qui modélise le coût de la régulation; pour l'arbre recouvrant extrait du tore 4×4 nous avons estimé dans la section 3.5 leurs valeurs : $b = -2.5211$ et $C = 0.3763$;
- θ est le temps de traitement d'un nouveau grain et τ le temps de transfert d'une unité de charge, nous verrons un peu plus loin comment obtenir ces valeurs.

Si tous les processeurs ont fini les calculs de recristallisation la décision sera prise en évaluant la relation 3.13 qui s'écrit sous la forme équivalente :

$$\frac{\mathsf{std}(charge)}{\max\limits_{v \in V} charge[v] - \overline{charge}} < \alpha * \frac{\theta}{\tau} * \frac{-\ln(-\ln p) - b}{C} \tag{4.19}$$

où α est une constante que nous avons choisie égale à 1.

Le temps de transfert τ d'une unité de charge qui se fait à travers un unique lien physique est constant et nous l'avons estimé (4 ticks). Quant au temps de traitement θ il est fonction de nombre de pas d'itérations restants et du temps de traitement θ_0 d'une unité pendant un pas de compression :

$$\theta = \left(\frac{\varepsilon}{\dot{\varepsilon}} - pas \right) * \theta_0$$

où $\frac{\varepsilon}{\dot{\varepsilon}}$ donne le nombre de pas d'itérations et pas est le numéro de l'itération courante. θ_0 est estimé d'abord localement sur chaque processeur lors du premier pas de compression (le premier pas d'itération ne contient pas de calcul de recristallisation) :

$$\theta_0[v] = \frac{T}{charge[v]}$$

où T est la durée de la première itération. Lors de la première initiation de la régulation θ_0 est calculé comme $\max\limits_{v \in V} \theta_0[v]$.

4.3.6.3 Politique de transfert. Reconstitution de l'espace d'adresses

Si la décision de faire la régulation est prise, les transferts entre processeurs se font à travers l'arbre recouvrant du réseaux. Aucun choix parmi les grains à transférer n'est fait, car un processeur ne connaît pas à priori la destination finale des unités de charge qu'il envoie.

Après la régulation, pour les unités de charge (i.e. grains) qui ont été transférées se pose le problème de pouvoir les retrouver lors de communications. Rappelons que lors des calculs de déformation chaque grain échange avec ses voisins (voisinage géométrique) toute une série d'information et que chaque grain doit être capable de retrouver ces voisins et d'être retrouvé. Un grain est uniquement déterminé par le biais de son identificateur. Nous avons choisi l'identificateur d'un grain comme le couple (v, l), où v est le processeur chargé à traiter le grain et l est le numéro d'ordre du grain sur ce processeur. Si un grain avec l'identificateur (v, l) est transféré sur un processeur v', son identificateur change en (v', l').

Il y a deux possibilités pour continuer le calcul après le transfert de charge :

1. Le transfert d'un grain est transparent aux grains voisins.

2. Le transfert est suivi d'une mise à jour de la nouvelle adresse auprès de ses voisins.

La première possibilité implique que pour tout grain transféré le processeur qui le possédait avant le transfert doit faire suivre les messages qui lui sont destinés et donc, d'une part, prévoir les fonctions de redirection de messages et, d'autre part, un sur-coût est ajouté à chaque pas d'itération. La deuxième possibilité laisse le code de déformation inchangé; par contre, il faut mettre en place des procédures qui assurent la cohérence des adresses. Dans les deux cas, tout processeur surchargé doit connaître les nouvelles adresses des grains qui lui appartenaient.

Nous avons choisi de mettre en place un protocole qui assure à la fin de son exécution la cohérence de l'espace d'adresses des unités de charge : chaque grain dispose de l'adresse exacte de ses voisins. Un fois les transferts de charges finis, chaque processeur exécute de manière synchrone les 4 pas suivants :

1. Si un processeur v a reçu des nouveaux grains, il leurs donne de nouveaux identificateurs.

2. Si un processeur v a reçu de la charge, il envoie aux anciens propriétaires de sa nouvelles charge les nouveaux identificateurs.
   Si un processeur u a envoyé de la charge, il reçoit les nouvelles adresses de ses anciens grains.

3. Pour tout nouveau grain g qui a reçu d'adresse (v, l), le processeur v envoie aux processeurs qui contiennent les voisins de g la nouvelle adresse de celui-ci.
   Si un grain voisin d'un grain transféré est lui aussi transféré, l'ancien processeur "propriétaire" fait suivre l'information.

4. Tout les processeurs mettent à jour les adresses des grains voisins.

Malgré sa forme, ce protocole s'est avéré facile à mettre en œuvre et pas trop coûteux en terme de temps (les mesures que nous avons prises estiment le temps d'exécution de ce protocole de 100 à 150 ticks, dépendant de la taille de l'agrégat).

4.3.6.4 Performances de l'algorithme d'allocation dynamique

La régulation de charge a pour but, dans le cadre précis de notre application, d'optimiser le temps de la dernière étape, i.e. les calculs de déformation des nouveaux grains. De ce fait nous présenterons les résultats en terme d'accélération globale de calcul et également, en terme d'accélération de la partie de l'application concernée par la régulation.

Lors de l'exécution de l'application il y a trois paramètres d'entrée : l'agrégat initial, le taux de déformation $\dot{\varepsilon}$ et la déformation totale ε. Les résultats obtenus varient, mais pas de manière significative, en fonction du germe de génération de nombre aléatoires. Les résultats présentés sont obtenus à partir des valeurs moyenne calculées pour les mêmes paramètres d'entrée et plusieurs germes différents.

Les agrégats choisis sont de taille 22, 57, 102, 198, 311 et 492 grains. Les taux de déformation $\dot{\varepsilon}$ sont de 10^{-1}, 10^{-2}, $0.5 * 10^{-3}$ et 10^{-3}. Afin d'uniformiser, nous avons pris la même déformation globale $\varepsilon = 1$.

Nous avons remarqué des moins bonnes performances pour les agrégats de grande taille. Ceci était prévisible et est dû à un effet de loi centrale limite : un nombre plus important d'unités de charge qui génèrent des nouvelles charges selon une même loi engendre un déséquilibre moins important.

Globalement, la régulation de charges a été décidée d'être appliquée dans tous les cas, sauf pour l'agrégat de taille 492 et $\dot{\varepsilon} = 10^{-1}$; dans ce cas, il reste 7 pas de compression à réaliser, mais les variations de charge d'un processeur à l'autre ne sont pas si importantes pour couvrir le coût de la régulation. En forçant l'application de la régulation, l'accélération obtenue est en dessous de 1. Pour tous les autres cas la régulation a été appliquée et l'accélération que nous avons mesurée justifie cette régulation.

Le temps total nécessaire pour la simulation de la déformation-recristallisation, en absence de tout allocateur de charges, est représenté dans la figure 4.24. Il varie en fonction de la taille de l'agrégat et du nombre total de pas de calcul (le taux $\dot{\varepsilon}$ de déformation).

En raison des calculs de recristallisations coûteux, l'accélération globale n'est pas trop importante, est comprise entre 1.001 et 1.228. La figure 4.25 montre ces valeurs. Si T_{appl_sans} est le temps total de l'application sans le module de régulation et T_{appl_avec} est le temps de l'application avec le module de régulation, l'accélération globale s'exprime comme :

$$\frac{T_{appl\_sans}}{T_{appl\_avec}}$$

Le gain de temps $T_{appl\_sans}T_{appl\_avec}$ est compris entre 0.12 secondes et 24.3 secondes. Le temps de régulation $6 * 10^{-4}$ pour l'agrégat de taille 22 et 10^{-2} pour l'agrégat de taille 492. Il varie peu pour une même taille de l'agrégat, nous avons enregistré une variation maximale de 30%.

L'accélération de la dernière étape de compression, qui est une valeur relative, est légèrement plus grande, comprise entre 1.001 et 1.6487; les valeurs sont représentées dans la figure 4.26. Si T_o est le temps nécessaire depuis le début de l'application pour finir les recristallisations, l'accélération relative qui nous intéresse s'exprime comme :

$$\frac{T_{appl\_sans} - T_o}{T_{appl\_avec} - T_o}$$

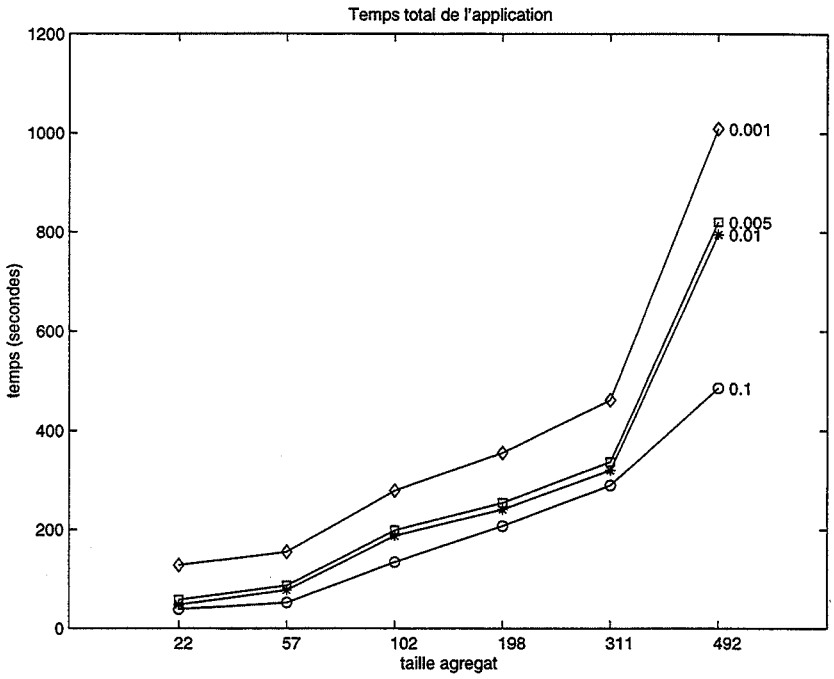


FIG. 4.24 – *Temps de simulation de la déformation-recristallisation sans régulation de charges.*

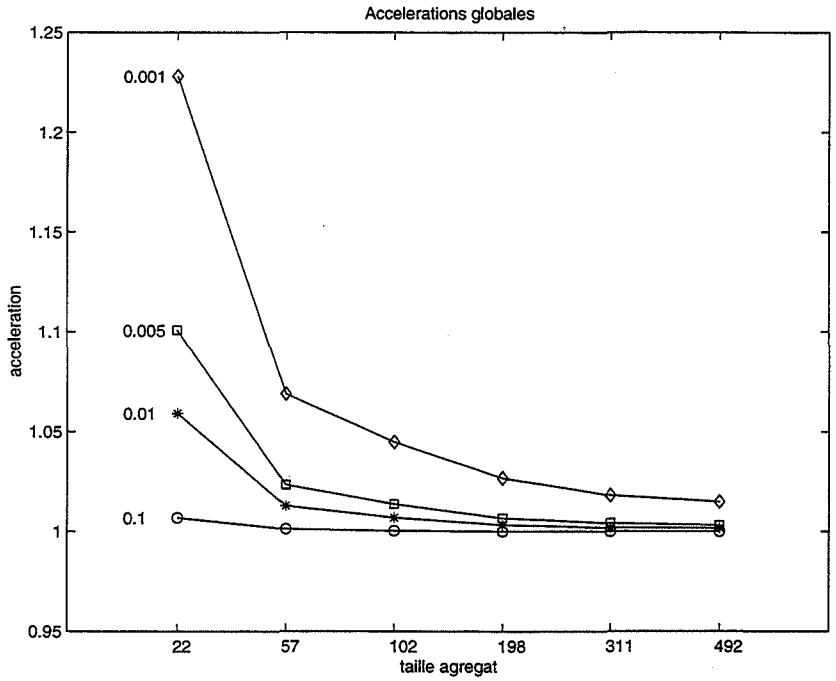


FIG. 4.25 – *Accélérations globales de l'application de déformation obtenues suite à la régulation de charges.*

Nous avons calculé l'accélération théorique relative comme :

$$\frac{pas\_restants * \theta * \max_{v \in V} charge[v]}{pas\_restants * \theta * \overline{charge}} = \frac{\max_{v \in V} charge[v]}{\overline{charge}}$$

Les valeurs obtenues sont comprises entre : 1.0307 et 1.6837.

4.3.6.5 Perspectives de développement

Nous dégageons deux grands axes d'une possible amélioration des performances de l'application.

Le choix de ne faire la régulation des charges qu'à la fin de toutes les recristallisations a été imposé par le coût très élevé du calcul de recristallisation par rapport au calcul de compression. Au cas où nous disposerions d'une fonction de recristallisation moins coûteuse, il serait sans doute intéressant de regarder la possibilité de faire la régulation pendant l'étape même de recristallisation.

Une autre direction possible de développement concerne la politique de transfert, à savoir connaître la destination des charges transférées et choisir les unités de charge (grains) qui minimisent mieux le temps de communication lors des calculs de compression. Il faudrait alors disposer d'une stratégie pour le découpage efficace de la structure géométrique de l'agrégat.

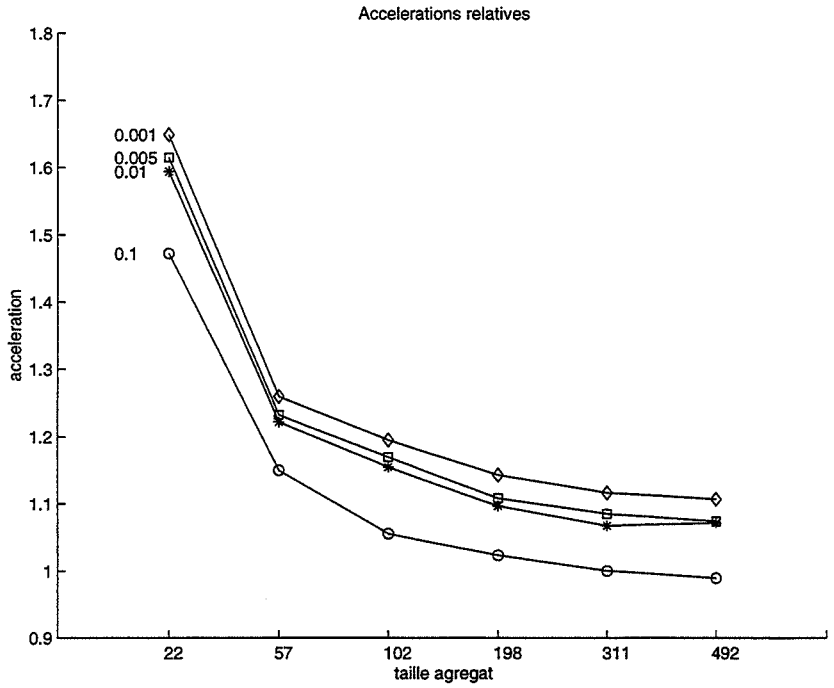


FIG. 4.26 – *Accélérations relatives à la dernière étape de compression de l'application de déformation obtenues suite à la régulation de charges.*

4.4 Résultats. Signification. Perspectives

4.4.1 Résultats mécaniques

Le but de cette partie est d'énoncer de manière succincte les résultats obtenus du point de vue mécanique. Il s'agit d'abord de valider le modèle original proposé (section 4.4.1.1), puis de mettre en évidence le comportement du modèle lors de la phase de compression (section 4.4.1.2) et de recristallisation (section 4.4.1.3), respectivement. Nous présenterons aussi dans 4.4.1.4 les extensions possibles du modèle.

4.4.1.1 Validité du modèle

Le matériau est incompressible, mais dans la description du modèle cette condition simple n'a pas été explicitée. La première chose que nous avons donc vérifiée était la variation du volume de l'agrégat, ce qui revenait à analyser la variation de la surface, car le modèle est bidimensionnel. Pendant une compression à $\varepsilon_\infty = 1$ avec des taux de compression de $d\varepsilon_\infty = 0.01$ et $d\varepsilon_\infty = 0.001$, nous avons pu constater une variation de la surface totale de l'ordre de 0.4

Par ailleurs, vu l'influence d'un faible taux de déformation sur la qualité des résultats de simulation du point de vue mécanique, toutes les simulations seront faites avec un taux de déformation de $d\varepsilon_\infty = 0.001$.

Par construction, le modèle respecte deux conditions de base : la loi de comportement locale viscoplastique (4.2) et le principe d'homogénéisation macroscopique (4.3). Parmi les caractéristiques de l'agrégat, on s'est intéressé au taux d'énergie plastique globale $\dot{W}$ (4.7) et au taux d'énergie microscopique $\dot{W}_{micro}$ (4.12). Pendant une déformation à $\varepsilon_\infty = 1$ le rapport $\frac{\dot{W}}{\dot{W}_{micro}}$ est toujours très proche de 1 (1.002). Donc la troisième condition de conservation de l'énergie est satisfaite. Ainsi le modèle vérifie les conditions de macro-homogénéité de Hill.

4.4.1.2 Déformation

L'agrégat n'est pas homogène, car par construction les grains sont supposés avoir des viscosités initiales différentes. Il est donc naturel de constater que les grains ne se déforment pas de la même façon. Dans la figure 4.27, nous avons représenté une structure initiale de 311 grains soumise à une déformation totale de $\varepsilon_\infty = 1$. Les nuances de gris indiquent la viscosité initiale des grains (entre 0.8 et $1.2 * 10^6$). La structure déformée est donnée dans la figure 4.28, les nuances de gris indiquent la déformation locale subie par un grain (entre 0.879 et 1.124). Il est facile de constater que les grains mous ont tendance à se déformer plus fortement que les grains plus durs.

Nous avons aussi fait des simulations dans les mêmes conditions, mais en ignorant la loi d'écrouissage, c'est à dire supposant que la viscosité des grains reste inchangée pendant la déformation. La représentation graphique de la structure déformée dans ces conditions semble similaire au cas précédent (voir la figure 4.29), mais les déformations locales sont dans une autre plage de valeurs : entre 0.8. et 1.2. Dans la figure 4.30, les histogrammes de ces déformations locales, avec et sans la loi d'écrouissage, sont représentés, pour une structure déformée de 3847 grains. On peut constater que l'écart-type est plus important en l'absence de la loi d'écrouissage (0.0602 contre 0.044) ce qui correspond à un effet d'homogénéisation de la structure au cas où on prend en compte la loi d'écrouissage. Dans les

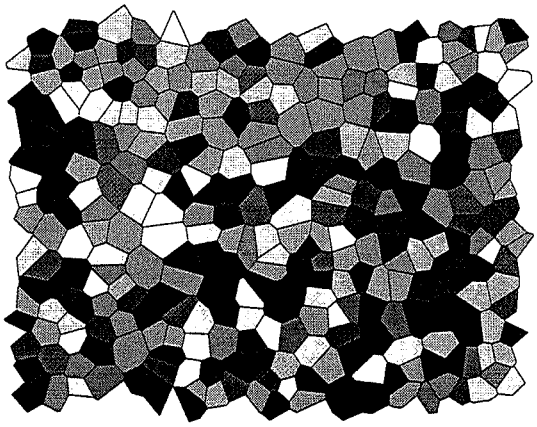


FIG. 4.27 – *Une structure initiale de 311 grains, les niveaux de gris indiquent la viscosité initiale des grains (k_{si}).*

FIG. 4.28 – *La structure de 311 grains déformé avec $\varepsilon_\infty = 1$, les niveaux de gris indiquent la déformation locale des grains (ε_i).*

FIG. 4.29 – *La structure de 311 grains déformé avec $\varepsilon_\infty = 1$ en absence de la loi d'écrouissage.*

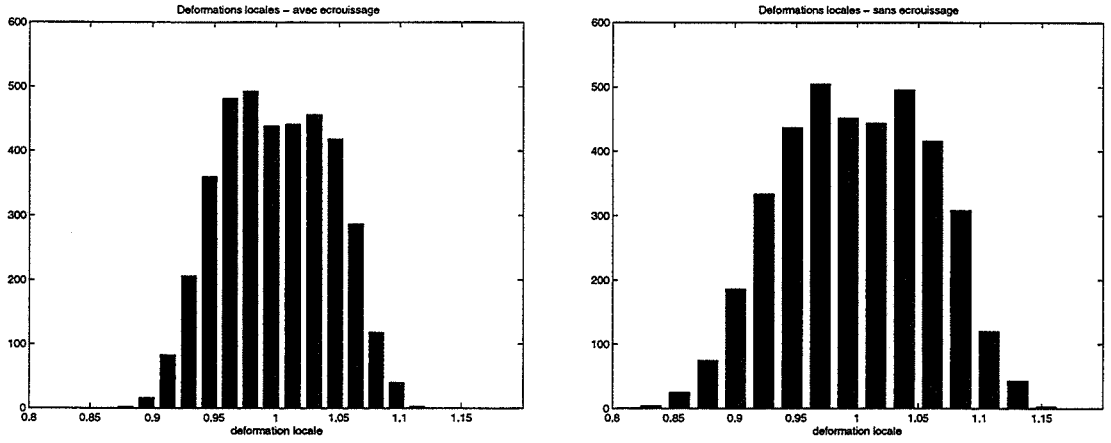


FIG. 4.30 – *Les histogrammes des déformations locales ε_i avec et, respectivement, sans écrouissage pour un agrégat de 3847 grains.*

deux cas, on a pu constater que les déformations locales ε_i présentaient une distribution normale pour leur valeurs.

Nous avons aussi calculé la viscosité globale de l'agrégat. La courbe croissante de cette valeur tout au long de la déformation rejoint le cas d'une déformation réelle [GM95]. Les simulations faites sans prendre en compte la loi d'écrouissage ont montré une légère croissance de la viscosité globale, donc un phénomène d'écrouissage morphologique. La figure 4.31 indique les deux courbes de la viscosité globale.

4.4.1.3 Recristallisation dynamique continue

Les simulations de recristallisation dynamique ont montré que les conditions de simulation, à savoir les paramètres choisis, sont très importantes.

Pour le même seuil d'énergie de recristallisation w_C et pour la même déformation globale ε, la recristallisation des grains se produit "plus tard" ou "plus tôt", c'est à dire pour des plus faibles ou plus grandes valeurs des déformations locales, en fonction de la vitesse de déformation imposée $\dot{\varepsilon}$; et même si cette vitesse est très réduite, la recristallisation ne se produit pas.

Par exemple, pour $w_C = 1.56 * 10^7$ et $\varepsilon = 1$, si $\dot{\varepsilon} = 0.1$ (un grand taux de déformation, 10 pas de déformation) tous les grains d'un agrégat recristallisent au même moment, lors du 6-ème pas de déformation (i.e. $\varepsilon = 0.5$). Si $\dot{\varepsilon} = 10^{-2}$ (100 pas), la première recristallisation d'un grain se produit autour du 70-ème pas de déformation et tous les grains recristallisent pendant une dizaine de

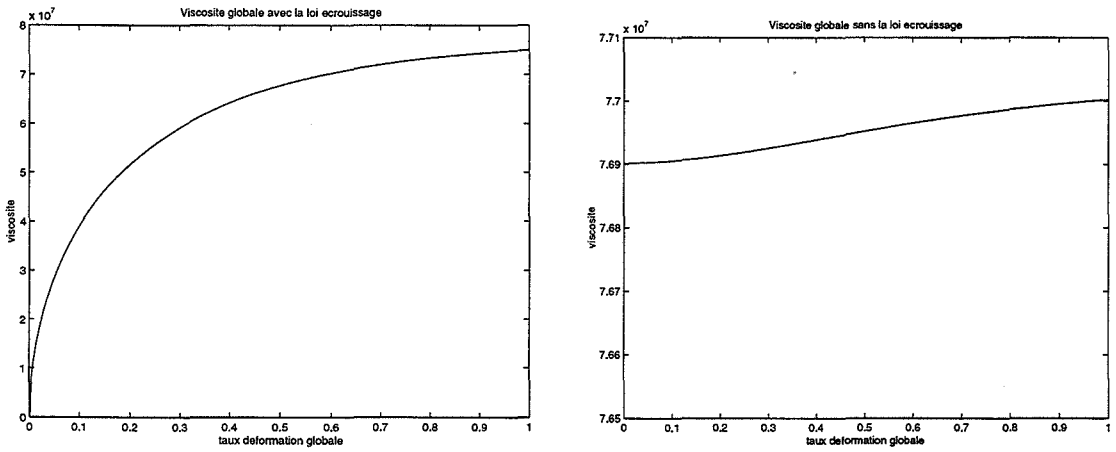


FIG. 4.31 – *Les courbes de la viscosité globale de l'agrégat de 3847 grains pour une évolution de la viscosité locale selon la loi d'écrouissage (à gauche) et en absence de cette loi (à droite).*

pas, i.e. quand $\varepsilon = 0.82$ tous les grains sont recristallisés. Si $\dot{\varepsilon} = 10^{-3}$ la première recristallisation se produit vers le 970-ème pas de simulation et un pourcentage d'environ 25% de grain recristallisent. Les histogrammes de la figure 4.32 montrent l'évolution du nombre de recristallisations produites à chaque pas pour le même agrégat de 492 grains. Nous avons pu constater aussi que tous les agrégats (sans exception de taille) ont le même comportement par rapport au premier (dernier) pas de déformation quand des recristallisations se produisent.

Un comportement de même type mais de plus faible envergure concernant le nombre des grains qui recristallisent simultanément peut être observé en faisant varier, pour un taux de déformation fixé $\dot{\varepsilon}$, le seuil de l'énergie de recristallisation w_C : pour des faibles valeurs de ce paramètre, il y a plus de grains qui recristallisent lors d'un même pas de compression. Les histogrammes de la figure 4.33 montrent le nombre des grains qui recristallisent pour un agrégat de 492 grains pour $\dot{\varepsilon} = 10^{-2}$ et $w_C = 1.56 * 10^7$ et $w_C = 0.56 * 10^7$.

Lors de l'analyse du point de vue mécanique de la compression, nous avons pu constater que les grains plus mous (viscosité initiale faible) subissaient une déformation plus forte que les grains plus durs. En mettant en évidence les grains qui recristallisent en premier, nous avons pu voir que ni les grains les plus durs, ni les plus mous ne recristallisent en premier. Toutefois, pour un même agrégat initial et pour des taux de déformation proches, ce sont les mêmes grains qui recristallisent en premier. La figure 4.34 montrent les grains qui recristallisent en premier pour un agrégat de 198 grains pour des taux de déformation proches de $\dot{\varepsilon} = 0.01$.

Du point de vue géométrique la frontière commune de deux grains voisins dont au moins un recristallise se transforme dans en jonction des frontières des nouveaux grains. Cette jonction est déterminée par les arêtes dont les sommets sont initialement sur la frontière commune, comme dans la figure 4.35. Un constat simple que nous avons pu faire est que cette jonction garde une forme à peu près linéaire[12]. Nous avons mesuré le rapport entre la moyenne des distances des nouveaux points au segment initial et la longueur de ce segment. Ce rapport est toujours resté plus petit que $0.4 * 10^{-2}$.

12. Vu les approximations (arrondis) faites lors de la représentation graphiques des polygones, ces points apparaissent toujours comme colinéaires.

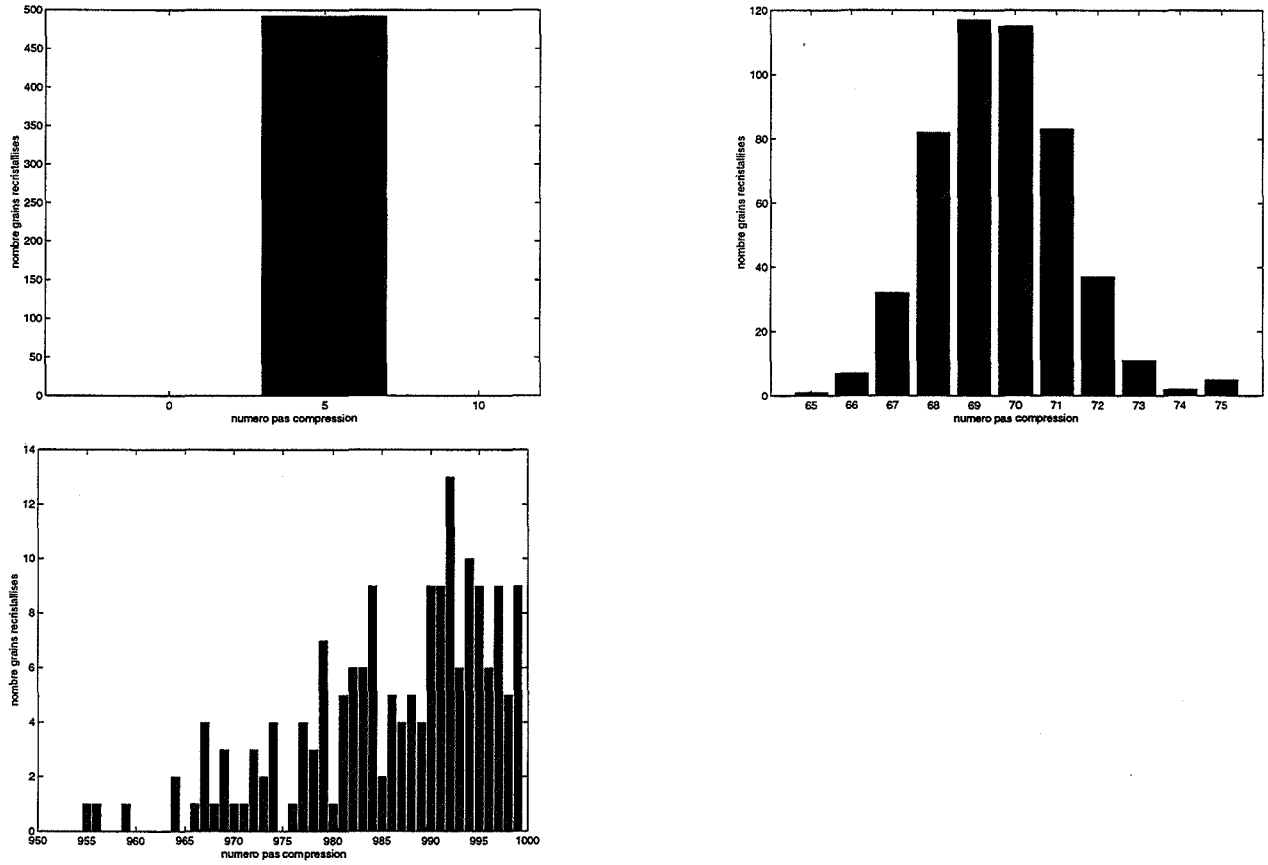


FIG. 4.32 – *Le nombre de recristallisations produites à chaque pas de simulation pour un agrégat de 492 grains, le même seuil d'énergie de recristallisation* $w_C = 1.56 * 10^7$ *et des taux de déformation de* 10^{-1}, 10^{-2} *et, respectivement,* 10^{-3}.

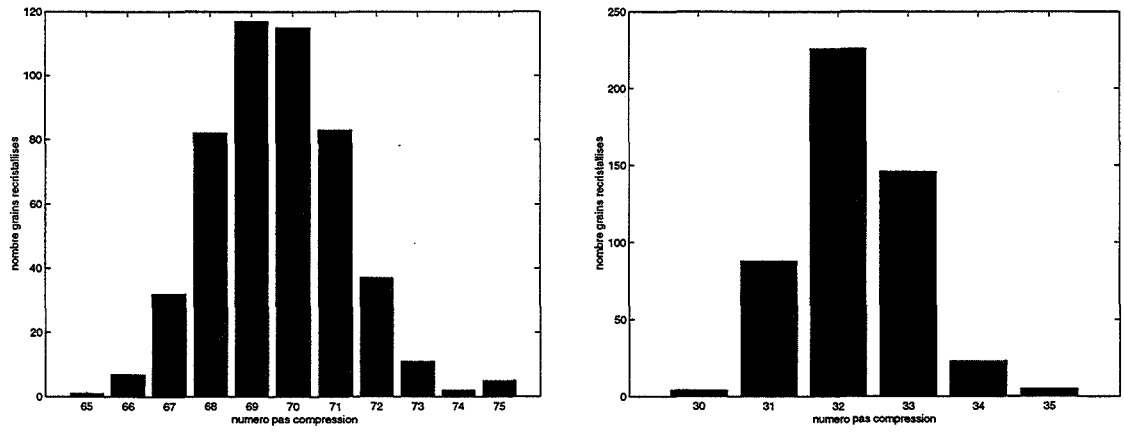


FIG. 4.33 – *Le nombre de recristallisations produites à chaque pas de simulation pour un agrégat de 492 grains quand* $\dot{\varepsilon} = 10^{-2}$ *et* $w_C = 1.56 * 10^7$ *et* $w_C = 0.56 * 10^7$, *respectivement.*

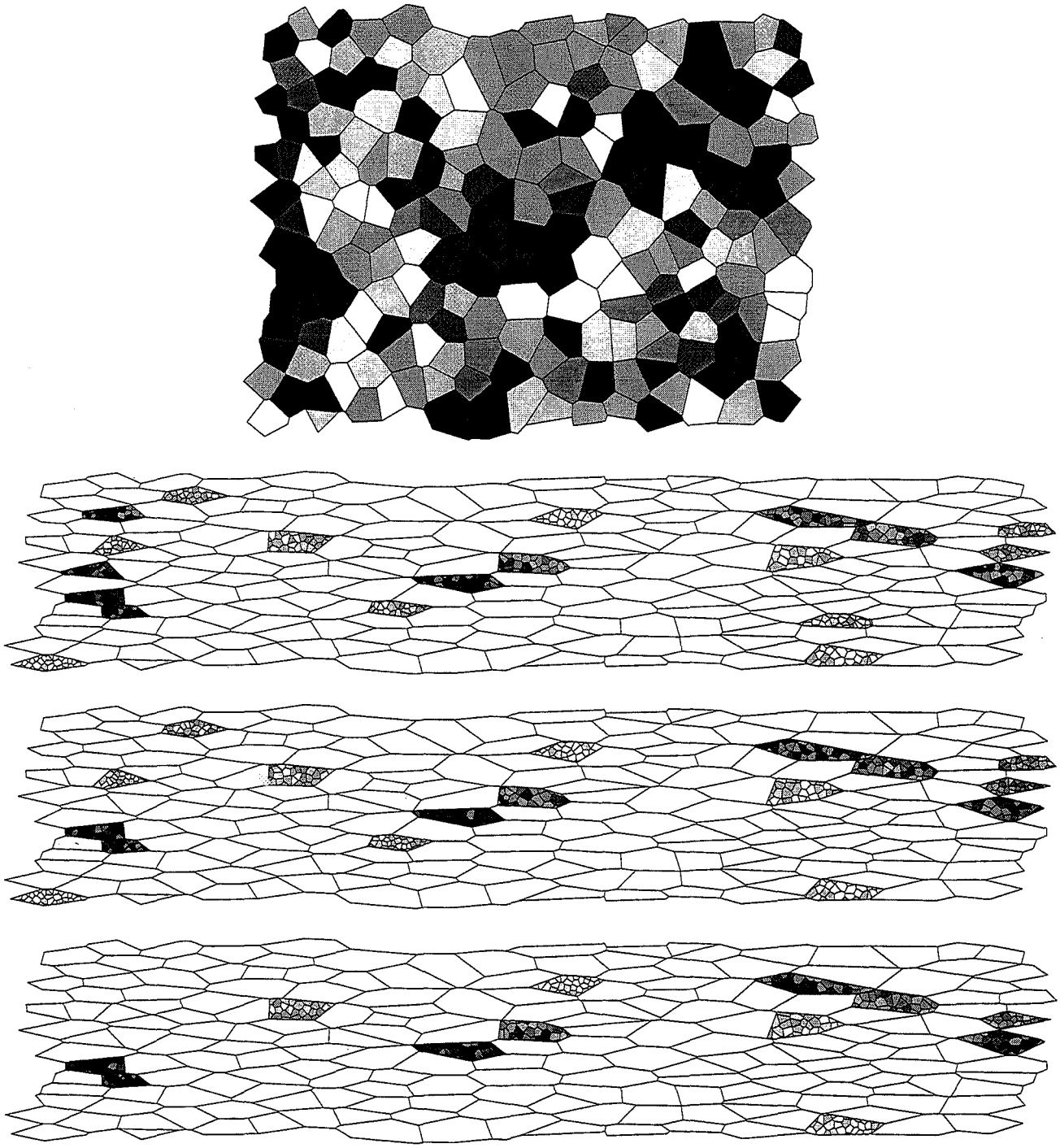


FIG. 4.34 – *Un agrégat de taille 198 et les grains qui recristallisent en premier pour de taux de déformation de* $0.99 * 10^{-2}$, 10^{-2}, $1.01 * 10^{-2}$ *(l'énergie de recristallisation est* $w_C = 1.56 * 10^7$*).*

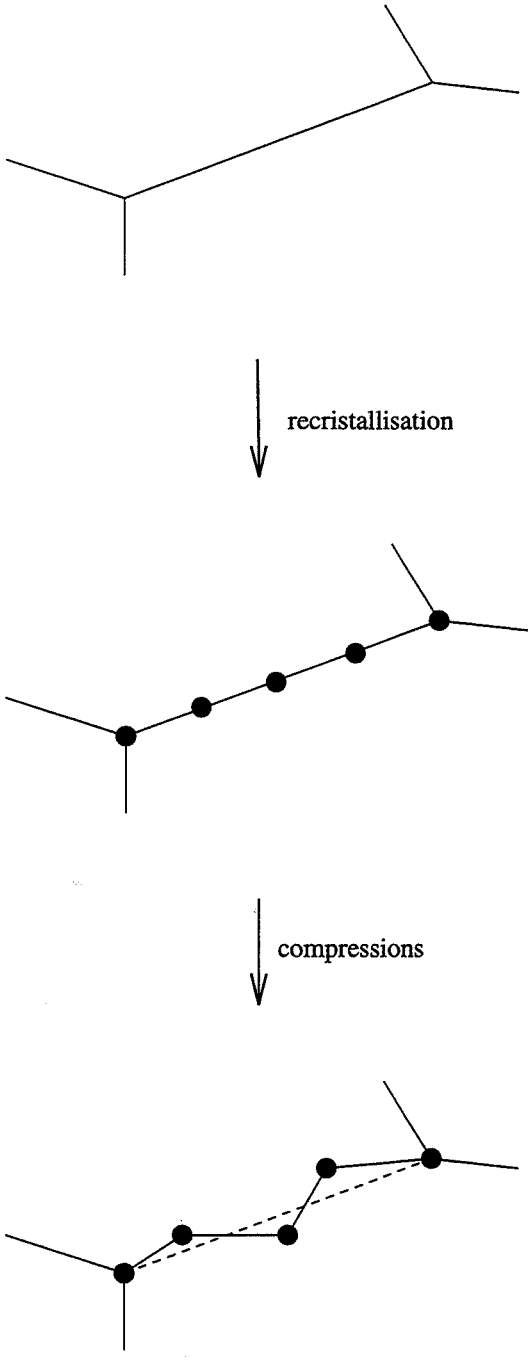


FIG. 4.35 – *Les transformations d'une frontière entre deux grains dont un recristallise.*

4.4.1.4 Perspectives

Il est évident que le modèle peut se généraliser au cas tridimensionnel, sans rien changer, la difficulté étant purement géométrique, liée à la génération de la topologie de base et des structures recouvrantes lors de la recristallisation.

Ce modèle a constitué un cadre idéal pour étudier les phénomènes mécaniques de compression et de recristallisation dynamique continue dans leur forme la plus simple. On peut enrichir ce modèle afin qu'il puisse englober d'autres caractéristiques physiques et mécaniques, i.e. l'orientation cristallographique, et afin qu'il se prête à la modélisation des formes plus élaborées de compression (i.e. compression selon les deux axes ou compression à température variable) ou à la modélisation d'autres phénomènes mécaniques tels que le glissement des joints de grains ou la recristallisation dynamique discontinue.

Le modèle proposé, simulé et validé par simulation est donc la base d'un modèle qui permet de contrôler et prévoir la méso-structure des matériaux polycristallins.

4.4.2 Résultats et perspectives informatiques

L'application, dont nous venons d'expliciter son implantation et les résultats mécaniques obtenus, s'est montrée aussi très fructueuse sur le plan informatique.

Tout d'abord l'algorithme de régulation de charge proposé s'est avéré correct et efficace, ce qu'était l'objectif de départ. Si le temps nous avait permis, nous aurions sans doute comparé les performances de notre algorithme avec un autre algorithme de même type, comme, par exemple celui de Jàjà et Ryu ou une méthode diffusive ou la méthode "échange de dimensions".

Des nombreux problèmes liés à l'implantation du modèle mécanique, indépendamment de la régulation de charge, se sont posés :

- quelles structures de données choisir;
- comment synchroniser les calculs;
- comment organiser les communication;
- comment retrouver les grains physiquement voisins situés sur des processeurs distants;
- comment retrouver le voisinage d'un grain obtenu suite à la recristallisation ou d'un grain ayant un grain voisin recristallisé;

La résolution et la mise en œuvre de certaines de ces questions se sont avérées parfois très laborieuses.

Quant à la mise en œvre de la régulation, le problème essentiel posé était celui de la migration du contexte des unités de charges transférées.

Une autre question très simple qui se pose, mais qui n'a pas été traitée faute d'une plus importante plateforme de programmation parallèle est la suivante : quels seraient les résultats et les performances de l'algorithme de régulation si la topologie de départ était autre que le tore 4×4?

Autres comparaisons qui auraient pu être faites concernent la politique de transfert des unités de charge. Nous avons été encouragé, par les performances du noyau de routage utilisé et par l'hypothèse de régularité des unités de charge, de ne pas tenir compte d'un choix réel des gains à transférer. Il serait, sans doute, intéressant d'analyser plus finement la relation qui existe entre l'éloignement physique des deux unités de charge et le temps total de traitement qui englobe le temps de communication entre ces deux unités. Dans la même ligne de développement, une analyse plus poussée de l'allocation statique de départ serait bienvenue, mais le cadre de cette thèse serait dépassé. Une telle allocation statique s'appuyerait sur la recherche d'un homomorphisme entre le graphe des grains et de tore 4×4 de développement qui aurait une dilatation minime.

Conclusions et perspectives

Bilan de la thèse et problèmes ouverts

Ce travail a débuté avec une étude bibliographique du problème d'allocation de charges dans les machines parallèles qui nous a fait dans un premier temps prendre connaissance de l'étendue du sujet et puis prendre conscience de l'importance de ce problème quant à la mise en œuvre des applications parallèles.

Cette recherche bibliographique nous a permis de dégager une voie qui nous a semblé prometteuse pour nos futures recherches : l'allocation dynamique des charges régulières. Nous avons donc commencé par proposer une stratégie nouvelle de régulation des charges régulières qui par rapport à celles qui existent dans la littérature travaille à la fois sur tout réseau d'interconnexion de processeurs sans demander des facilités spéciales de communication autres que la communication point à point et permet d'aboutir en temps fini à l'état global d'équilibre des charges. Nous avons prouvé la correction de cette méthode et analysé sa complexité en temps de calcul, nombre des messages et temps total d'exécution. Pour ce dernier critère il s'est révélé que les analyses classiques ne permettent pas d'obtenir des complexités fines et que, lorsqu'on en obtenait, celles-ci s'avéraient inexploitables.

Nous avons donc procédé à une analyse probabiliste de cette complexité en temps total. Pour cela la méthodologie a consisté d'abord à modéliser la valeur qui nous intéresse en terme de variable aléatoire qui dépend de variables aléatoires indépendantes. Suite à une telle modélisation nous avons procédé à des simulations systématiques dont les résultats ont été traité statistiquement. Cette méthodologie est générique et peut donc s'appliquer pour analyser toute méthode de régulation.

Le résultat principal de cette analyse probabiliste est que le temps total d'exécution de la stratégie de régulation suit une loi de Gumbel, fait qui permet de donner en probabilité une borne fine de ce temps. Ceci est non-négligeable si on veut proposer des politiques d'initiation et de décision associées à cette stratégie.

L'étape suivante du travail a été donc de décrire un algorithme complet d'allocation dynamique de charges régulières qui par construction est exact, extensible, correct et efficace.

L'efficacité de l'algorithme de régulation étant purement théorique, nous sommes allés à la recherche d'une application réelle qui pourrait se paralléliser et générer dynamiquement de la charge. Une telle application nous a été proposée par Frank Montheillet, directeur de recherche CNRS dans le département Matériaux et Mise en Forme à l'Ecole des Mines de St. Etienne. Il s'agissait de simuler un modèle de recristallisation dynamique continue d'un agrégat polycristallin. Cette collaboration auquelle Claire Maurice, chargé de recherche du même département, s'est associée, s'est avérée très fructueuse aussi pour l'équipe mécanique, car les simulations sur architectures parallèles leur ont permis de mieux comprendre les phénomènes étudiés.

Les résultats des expériences de simulations du modèle mécanique avec différents paramètres ont montré que l'algorithme proposé est, en practique, correct et efficace.

L'approche dévoppée est viable et peut s'appliquer aussi bien pour l'équilibrage du traitement que pour l'équilibrage des données. En plus de sa correction, sa stabilité et son efficacité, l'algorithme proposé montredes propriétés importantes: il est extensible, indépendant du rséau d'interconnexion des processeurs et robuste.

Il reste néanmoins quelques problèmes ouverts concernant le travail mené. Nous allons les lister et ensuite essayer d'apporter quelques éléments de réponse à un d'entre eux.

Au cours de notre travail nous nous sommes heurtée a beaucoup plus de questions que celles de la liste, mais une bonne partie ont été résolues convenablement. Il demeure encore trois grand problème ouverts:

- Est-il possible d'améliorer les performances de la stratégie initiale? c'est à dire réduire le temps des échanges des charges pour atteindre d'état d'équilibre.

  Nous avons tenté de proposer des modifications successives de l'arbre construit initialement, en essayant d'enlever une arête qui est sur le chemin le plus coûteux, en ajoutant des liens extérieurs à l'arbre. Les améliorations ne sont pas négligeables; pour des topologies de taille importante (plus d'une trentaine des nœuds) elles sont de 50 à 80%. Toutefois ces calculs sont faits en séquentiel et leurs temps sont de l'ordre de $O(n^3)$, n étant les nombre des nœuds. La solution semble lourde et coûteuse à mettre en œuvre en tant qu'algorithme distribué.

- Peut-on prouver de manière formelle que la loi de la variable aléatoire W^* est une loi de Gumbel? peut-on également déterminer de manière formelle en partant de la description de l'arbre recouvrant les paramètres a et b de la loi de Gumbel?

- Comment trouver des éstimateurs "exacts" (plus proches de la réalité) pour les valeurs de la charge moyenne et de l'écart-type lors de l'exécution de la politique d'initiation quand ces valeurs ne sont pas encore connues?

Perspectives

Le domaine de l'allocation de charges dans les architectures parallèles est très vaste, laissant libre cours à beaucoup de développements et extensions.

Dans l'immédiat, il serait intéressant de travailler toujours sur l'application dynamique de simulation mécanique, mais en partant d'un modèle moins régulier[13] qui assurerait un déséquilibre différent ou plus important entre les charges des processeurs. Aussi pour la même application on pourrait faire

13. Un modèle bi-phases ou un modèle qui comporte un (des) groupement des grains de dureté différente par rapport au reste de l'agrégat sont des exemples de modèles moins réguliers.

la régulation avec d'autres stratégies accompagnées ou pas des mêmes politiques d'initiation et de décision.

De point de vue pratique, on pourrait intégrer le noyau de régulation construit dans un outil d'allocation déjà existant. Ceci fournirait à long terme des comparaisons sur l'efficacité de la méthode proposée pour différents types d'application et par rapport à d'autres méthodes d'allocation dynamique.

La méthode d'analyse probabiliste employée est originale et a permis de trouver la loi de probabilité qui modélise la borne supérieure du temps d'exécution. Les questions qui se posent sont les suivantes :

- pour d'autres stratégies de régulation pourrait-on trouver une telle loi?
- serait-elle aussi une loi de Gumbel?
- si oui, comment varieront les paramètres?
- aurons-nous le paramètre b le même pour toute stratégie et le même réseau d'interconnexion?

On peut aussi imaginer différentes extensions de la stratégie de régulation de départ :

- si on calculait l'arbre recouvrant au moment de l'initiation de la régulation, car l'arbre en parcours en largeur d'abord n'est pas forcément unique. La construction de l'arbre tiendrait compte de la charge des nœuds, on essayerait, par exemple, de relier des nœuds qui ont des écarts importants de charge.
- si on construisait plusieurs arbres recouvrants (arêtes disjoints) afin d'utiliser mieux toutes les arêtes du réseau d'interconnexion.
- si les unités de charge était linéairement dépendantes, comme dans [Mig92], comment on pourrait modifier notre stratégie afin de faire de la régulation élastique : qui préserve une distance physique de maximum 1 entre unités de charge dépendantes.

Nous avons travailler sous l'hypothèse de la régularité de la charge de travail, dans le cas où on ne dispose pas d'information sur le coût d'une unité de charge on peut faire aussi cette supposition de régularité et appliques l'algorithme présenté. Du travail reste à faire pour appliquer le calcul de préfixe généralisé au problème d'allocation dynamique en général, sans l'hypothèse de la régularité de charge.

Annexe A

Petit complément de théorie des graphes

A.1 Notions et notations

Nous allons présenter, sans rentrer dans les détails, les quelques notions de la théorie des graphes qui apparaîssent tout au long de cette thèse. Une large monographie qui contient la plupart des notions qui seront décrites ici est [Ber83].

A.1.1 Graphes en général

Un **graphe orienté** $G = (V, A)$ est constitué de deux ensembles : V, un ensemble de points, dits **sommets**, et $A \subset V \times V$ un ensemble des couples des points $(x, y) \in V \times V$, dits **arcs**.

Un graphe **non-orienté** $G = (V, E)$ diffère d'un graphe orienté par le fait que les couples des points de l'ensemble E, des **arêtes**, ne sont pas ordonnés, i.e. si $(x, y) \in E$, alors $(y, x) \in E$.

A.1.2 Graphes non-orientés

Si $G = (V, E)$ est on graphe non-orienté, on parle alors de :

- **relation d'adjacence** : si $(u, v) \in E$, on dit que les deux nœuds sont adjacents;
- **voisinage** d'un sommet v : $N(v) = \{w \in V : (v, w) \in E\}$; voisinage d'un ensemble des nœuds $S \subset V$: $N(S) = \{w \in V : \exists x \in S$ tel que $(x, w) \in E\}$;
- **degré** d'un nœud $d_G(v)$: représente le nombre de voisins du sommet v : $d_G(v) = |N(v)|$;
- **chemin** (ou chaîne) entre deux sommets v et u : une succession des sommets distincts $[v = x_0, x_1, x_2, \cdots, x_{n-1}, x_n = u]$ telle que $(x_{i-1}, x_i) \in E$ pour tout $i = 1, n$; n représente la longueur du chemin;
- **cycle** : un chemin dont le premier sommet coïncide avec le dernier;
- **distance** entre v et u notée $dist(v, u)$ qui est la longueur minimale des chemins entre v et u;

- **rayon** en v: la distance maximale entre v et tout sommet de G;
- **centre** du graphe: un sommet ayant un rayon minimal (égale au diamètre du graphe);
- **graphe partiel** $H = (V', E')$: un graphe où $V' \subset V$ et $E' \subset E$;
- **sous-graphe** $H = (V', E')$: un graphe où $V' \subset V$ et E' eat la restriction à E de $V' \times V'$.

On associe aussi des valeurs numériques au graphe G:

- **ordre**: le nombre des sommets $|V|$;
- **degré**: le maximum des degrés de ses sommets $d(G) = \max_{v \in V} d_G(v)$;
- **expansion**: la plus grande valeur μ telle que pour tout sous-ensemble S de V, ayant au plus $\frac{|V|}{2}$ éléments, le voisinage de S, $N(S)$, ait au moins $\mu|S|$ élements à l'extérieur de S;
- **diamètre** du graphe $diam(G)$, qui est la distance maximale entre deux sommets:

$$diam(G) = \max_{u,v \in V} dist(u, v)$$

Si $\omega : E \to I\!R^+$ est une fonction coût associée aux arêtes du graphe G, on parle de:

- **poids d'un chemin** $[u, v]$ par rapport à ω: $\omega([u, v]) = \sum_{e \in [u,v]} \omega(e)$, par abus on va noter simplement $\omega(u, v)$;
- **poids d'un sous-graphe** $H = (V', E')$ par rapport à ω: $\omega(H) = \sum_{e \in E'} \omega(e)$, où $S \subset V$ et $\bar{S} = V - S$;
- **poids d'une coupe** $(S, \bar{S})$ par rapport à ω: $\omega(S, \bar{S}) = \sum\{\omega(u, v) | (u, v) \in E, u \in S, v \in \bar{S}\}$;
- **conductance fluide** de G: $\Phi(G) = \min\{\frac{\omega(S, \bar{S})}{min(|S|, |\bar{S}|)} | S \subset V\}$;
- **résistance électrique** de l'arête (u, v): $\frac{1}{\omega(u, v)}$;
- **résistance électrique effective** entre les points u et v: $RES(u, v)$ calculée selon les lois physiques pour une unité de courant électrique appliqué entre l'entrée u et la sortie v;
- **conductance électrique** de G: $\Gamma(G) = \min_{u,v \in V} \frac{1}{RES(u, v)}$.

A.1.3 Fonctions entre graphes

Soient $G = (V_1, E_1)$ et $H = (V_2, E_2)$ deux graphes et $\phi : V_1 \to V_2$ une application entre les ensembles des sommets.

L'application ϕ est dite **isomorphisme**, si :

- ϕ est une bijection;
- $\forall u, v \in V_1 : (u, v) \in E_1$ si et seulement si $(\phi(u), \phi(v)) \in E_2$.

L'application ϕ est dite **homéomorphisme**, si :

- ϕ est surjection;
- $\forall u, v \in V_1$: si $(u, v) \in E_1$ alors $\phi(u) = \phi(v)$ ou $(\phi(u), \phi(v)) \in E_2$.

Ceci correspond à un groupement des nœuds du graphe G dans des nouveaux nœuds en respectant la relation d'adjacence.

L'application ϕ accompagnée d'une autre application P_ϕ qui associe à chaque arête de G une chaîne dans H est dite **plongement** de G dans H. Afin de mesurer la qualité d'un plongement on introduit la notion de dilatation qui est la longueur maximale des chaînes de H associées aux arêtes de G.

A.1.4 Produit des graphes

Soient $G = (V_1, E_1)$ et $H = (V_2, E_2)$ deux graphes, leur **produit cartésien** noté $G \square H = (V, E)$ est définit par :

- l'ensemble de sommets V qui est le produit cartésien des deux ensembles de sommets de départ : $V = V_1 \times V_2$;
- l'ensemble des arêtes E qui est donné par :

$$\begin{aligned} E \;=\; & \{((x, v), (y, v)) : (x, y) \in E_1, v \in V_2\} \\ & \cup \{((x, v), (x, u)) : x \in V_1, (v, u) \in E_2\} \end{aligned}$$

A.2 Graphes particuliers

Nous nous sommes intéressé uniquement aux graphes non-orientés qui correspondent aux topologies réelles ou virtuelles (par le biais du noyau de routage) des machines parallèles à mémoire distribuée les plus répandues; leur caractéristiques sont décrites en détail dans [Lei93].

Dans la plupart des cas nous préférons exprimer ces graphes par des produits d'autres graphes.

- Graphe complet d'ordre n, K_n : quels que soient deux sommets, ils sont reliés par une arête.

- Réseau linéaire d'ordre n, P_n : tous les nœuds sauf deux ont exactement 2 voisins, les deux autres en ont un seul;
- Cycle d'ordre n, C_n : tous les nœuds ont exactement 2 voisins;
- Grille de dimension m et avec $n1$, n_2, ... n_m nœuds dans chaque dimension, respectivement, $G_{n_1 \times n_2 \times \cdots n_m} = P_{n_1} \Box P_{n_2} \Box \cdots \Box P_{n_m}$;
- Tore de dimension m et avec $n1$, n_2, ... n_m nœuds dans chaque dimension, respectivement, $T_{n_1 \times n_2 \times \cdots n_m} = C_{n_1} \Box C_{n_2} \Box \cdots \Box C_{n_m}$;
- Hypercube de dimension d, H_d définit récursivement comme :
  - $H_2 = K_2$
  - $H_d = H_{d-1} \Box K_2$.

Annexe B

Petit complément de probabilités et statistiques

Le but de cette annexe est de présenter toutes les notions de la théorie des probabilités et de statistiques utilisées tout au long de la thèse. Nous répertorions d'abord les différentes lois de distribution de variables aléatoires; dans la section suivante B.2 les tests d'adéquation, papiers fonctionnels et tests statistiques, utilisés seront brièvement énoncés. Dans la dernière section B.3, nous montrons comment nous avons estimé les paramètres de la loi de Gumbel.

B.1 Quelques lois de probabilité

Soit X une variable aléatoire; X peut être discrète ou continue. Si X est discrète, nous la définirons par son tableau de valeurs, sinon par F sa fonction de répartition et par f sa densité. Nous donnerons également la valeur de l'espérance et de la variance.

B.1.1 Lois discrètes

Loi de Bernoulli

Si X suit une loi de Bernoulli, X prend les valeurs 0 et 1 avec une probabilité de $1-p$ et, respectivement, p, où $0 \leq p \leq 1$. $E(X) = 1$ et $D^2(X) = p * (1-p)$.

Loi binomiale $\mathcal{B}(n;p)$

Si $X_1, X_2, ..X_n$ sont n variables aléatoires indépendantes de Bernoulli de paramètre p et X est la somme de ces variables, alors X suit une loi binomiale :

$$P(X = k) = C_n^k * p^k * (1-p)^{n-k}, 0 \leq k \leq n$$

$E(X) = n$ et $V(X) = n * p * (1 - p)$

Loi de Poisson $\mathcal{P}(\lambda)$

Si X suit une loi de Poisson $\mathcal{P}(\lambda)$, on a:

$$P(X = x) = \mathrm{e}^{-\lambda} * \frac{\lambda^x}{x!}, x \in I\!N$$

$E(X) = \lambda$ et $V(X) = \lambda$.

B.1.2 Lois continues

Loi uniforme $\mathcal{U}[a, b]$

Si X suit une loi uniforme, sa densité est:

$$f(x) = \begin{cases} \frac{1}{b-a}, & \text{si } x \in [a, b] \\ 0, & \textit{sinon.} \end{cases}$$

$E(X) = \frac{a+b}{2}$ et $V(X) = \frac{(b-a)^2}{12}$.

Loi normale (de Laplace-Gauss) $\mathcal{N}(m, \sigma)$

Si X suit une loi normale, sa densité est:

$$f(x) = \frac{1}{\sigma * \sqrt{\pi}} * \mathrm{e}^{-\frac{1}{2} * \left(\frac{x-m}{\sigma}\right)^2}$$

sa fonction de répartition est:

$$F(x) = \Phi\left(\frac{x-m}{\sigma}\right)$$

où Φ est la fonction de Laplace: $\Phi(x) = \frac{1}{\sqrt{2\pi}} * \int_{-\infty}^{x} \mathrm{e}^{-\frac{y^2}{2}} \mathrm{d}y$, $x > 0$. Notons que cette loi a la propriété d'additivité: si X_1 et X_2 sont des variables aléatoires normales et indépendantes, alors $X_1 + X_2$ est aussi une v.a. normale.

$E(X) = m$ et $V(X) = \sigma^2$.

Loi lognormale

X suit une loi lognormale, si son logarithme népérien suit une loi normale :

$$\log X \sim \mathcal{N}(m, \sigma)$$

Sa fonction de densité est :

$$f(x) = \begin{cases} \dfrac{1}{\sigma * \sqrt{2\pi} * x} * \mathrm{e}^{-\dfrac{(\ln x - m)^2}{2 * \sigma^2}} & x > 0 \\ 0 & x \leq 0 \end{cases}$$

$E(X) = \mathrm{e}^{\left(m + \frac{\sigma^2}{2}\right)}$ et $V(X) = \mathrm{e}^{(2*m+\sigma^2)} * \left(\mathrm{e}^{\sigma^2} - 1\right)$.

Loi de Weibull

Si X suit une loi de Weibull, sa densité est :

$$f(x) = \begin{cases} \beta * \theta^{-\beta} * x^{\beta-1} * \mathrm{e}^{-(\frac{x}{\theta})^{\beta}}, & x \geq 0 \\ 0, & x < 0 \end{cases}$$

$\beta > 0$ est le paramètre de forme et θ est le paramètre d'échelle.
$E(X) = \theta * \Gamma(\frac{1}{\beta})$ et $V(X) = \theta^2 * \left[\Gamma(\frac{2}{\beta} + 1) - \Gamma^2(\frac{1}{\beta} + 1)\right]$ où la fonction Γ est : $\Gamma(u) = \int_0^{+\infty} y^{u-1} * \mathrm{e}^{-y} \mathrm{d}y$.
Pour des valeurs particulières de β on retrouve d'autres lois :

· $\beta = 1$ – la loi exponentielle négative :

$$f(x) = \begin{cases} \dfrac{1}{\theta} * \mathrm{e}^{-\dfrac{x}{\theta}}, & x \geq 0 \\ 0, & x < 0 \end{cases}$$

· $\beta = 2$ – la loi de Rayleigh :

$$f(x) = \begin{cases} 2 * \dfrac{1}{\theta^2} * x * \mathrm{e}^{-\dfrac{x^2}{\theta^2}}, & x \geq 0 \\ 0, & x < 0 \end{cases}$$

Loi de Gumbel

Si X suit une loi de Gumbel, sa fonction de répartition est :

$$F(x) = \mathrm{e}^{-\mathrm{e}^{-(a*x+b)}}$$

et sa fonction de densité est :

$$f(x) = a * \mathrm{e}^{-(a*x+b)} * \mathrm{e}^{-\mathrm{e}^{-(a*x+b)}}$$

a est le paramètre d'échelle et b est le paramètre de centrage. Si X suit une loi de Gumbel de paramètre a et b, alors $\dfrac{X-b}{a}$ suit une loi de Gumbel de paramètres 0 et 1.

Les formules de l'espérance et de la variance sont :

$$E(X) = \frac{\gamma - b}{a} \tag{B.1}$$

$$V(X) = \frac{\frac{\pi^2}{6}}{a^2} \tag{B.2}$$

où γ est la constante d'Euler.

B.2 Papiers fonctionnels

Les papiers fonctionnels sont des outils statistiques utilisés, avant de mettre en œuvre de vrais tests statistiques, pour avoir une représentation graphique facile des échantillons de valeurs par rapport à une certaine loi de probabilité. Dans cette annexe nous montrerons comment obtenir les papiers de la loi normale, de la loi lognormale, de la loi de Weibull et de la loi de Gumbel. Nous expliquerons d'abord le principe des papiers fonctionnels.

Formules générales

Un problème essentiel de la statistique et de trouver la loi de distribution d'une variable aléatoire X, inconnue, dont on ne dispose que d'un échantillon de n valeurs : $(x_1, x_2, \cdots, x_n)$. Ceci revient à trouver la fonction de répartition F :

$$F(x) = P(X \leq x)$$

Quelle que soit la méthode employée pour résoudre ce problème, la première étape est de formuler une hypothèse :

$$(\mathcal{H}) \text{ “}X \text{ suit la loi } \mathcal{L} \text{ ”}$$

La loi $\mathcal{L}$ étant représentée par la fonction de répartition F_0, l'hypothèse $(\mathcal{H})$ est équivalente à :

$$(\mathcal{H}') \text{ “ } F = F_0 \text{ ”}$$

F_0 est dite la fonction théorique de X.

Le papier fonctionnel de la loi $\mathcal{L}$ appliqué à l'échantillon $(x_1, x_2, \cdots, x_n)$ permet de se faire vite une idée de la validité de l'hypothèse $(\mathcal{H}')$, car si $(\mathcal{H}')$ est valide, la graphique obtenu est proche d'une droite.

A présent voyons comment on construit un papier fonctionnel. Supposons que $x_1 \leq x_2 \leq \cdots \leq x_n$, la fonction de répartition empirique F_n de X est définie par :

$$F_n(x_i) = \frac{i}{n} \tag{B.3}$$

Si l'hypothèse $(\mathcal{H}')$ est vraie, $F_n(x_i)$ est une bonne approximation $F_0(x_i)$, alors :

$$F_0(x_i) \simeq \frac{i}{n} \tag{B.4}$$

Si on représentait les points de coordonnées $(F_0(x_i), \frac{i}{n})$, ils seraient tous situés près de la droite $y = x$, mais la fonction F_0 dépend le plus souvent de paramètres qui ne sont pas toujours faciles à évaluer. Pour contourner le calcul de ces paramètres on cherchera des fonctions mathématiques $\mathcal{G}$ et $\mathcal{J}$ en partant de la formule de la fonction F_0, telles que l'ensemble des points $(\mathcal{G}(x_i), \mathcal{J}(\frac{i}{n}))$, $i = 1, n$ appartiennent à une même droite.

Papier de la loi normale

La fonction de répartition de la loi normale est :

$$F(x) = \Phi\left(\frac{x - m}{\sigma}\right)$$

Selon la formule (B.4) il faut transformer : $\left(\Phi\left(\frac{x_i - m}{\sigma}\right), \frac{i}{n}\right)$. On applique l'inverse de la fonction de Laplace : $\left(\frac{x_i - m}{\sigma}, \Phi^{-1}\left(\frac{i}{n}\right)\right)$.

Le papier fonctionnel consiste à représenter les points $\left(x_i, \Phi^{-1}\left(\frac{i}{n}\right)\right)$.

Papier de la loi lognormale

Si X suit une loi lognormale, alors $\log X$ suit une loi normale. Alors le papier fonctionnel de la lognormale s'obtient à partir du papier de la normale : on représente donc les points $\left(\log x_i, \Phi^{-1}\left(\frac{i}{n}\right)\right)$.

Papier de la loi de Weibull

La fonction de répartition d'une variable aléatoire de Weibull est :

$$F(x) = \begin{cases} 0 & \text{si } x < 0 \\ 1 - e^{-b * x^a} & \text{si } x \geq 0 \end{cases}$$

En remplaçant cette formule dans (B.4), on a :

$$1 - e^{-b * {x_i}^a} \simeq \frac{i}{n}$$

Si on applique deux fois la fonction logarithme, on obtient successivement : $(b * {x_i}^a, -\log(1 - \frac{i}{n}))$ et $(\log b + a * \log x_i, \log(-\log(1 - \frac{i}{n})))$. On représente alors les points de coordonnées : $(\log x_i, \log(-\log(1 - \frac{i}{n})))$.

Papier de la loi de Gumbel

La fonction de répartition de la loi de Gumbel est :

$$F(x) = e^{-e^{-(a*x+b)}}$$

Alors on doit transformer $(e^{-e^{-(a*x_i+b)}}, \frac{i}{n})$. On applique toujours deux fois la fonction logarithme $(a * x_i + b, -\log(-\log(-\frac{i}{n})))$.

Le papier de Gumbel consiste alors à représenter les points : $(x_i, -\log(-\log(\frac{i}{n})))$.

B.3 Estimation des paramètres d'une loi de Gumbel

Le problème qui se pose est le suivant : étant donné un échantillon $(x_1, x_2, \cdots, x_n)$ de n valeurs de la variable inconnue X, comment trouver les paramètres a et b. Nous détaillerons comment obtenir ces paramètres dans le cas où X suit une loi de Gumbel.

Nous avons vu qu'une v.a. de Gumbel X a la fonction de répartition :

$$F(x) = e^{-e^{-(a*x+b)}}$$

et que sa densité est :

$$f(x) = a * e^{-(a*x+b)} * e^{-e^{-(a*x+b)}}$$

B.3.1 La méthode des moments

Une méthode simple est la méthode des moments qui consiste, d'une part, à estimer les premiers moments de la v.a. à l'aide de l'échantillon et, d'autre part, d'essayer d'exprimer les paramètres de la loi en fonction de ces moments.

Les formule de l'espérance et de la variance de X sont :

$$E(X) = \frac{\gamma - b}{a}$$

$$V(X) = \frac{\frac{\pi^2}{6}}{a^2}$$

Alors :

$$\begin{aligned} a &= \pi * \sqrt{\frac{V(X)}{6}} \\ b &= \gamma - a * E(X) \end{aligned}$$

Les quantités $E(X)$ et $\sqrt{V(X)}$ sont inconnues, elles peuvent toutefois s'exprimer à l'aide de la moyenne de l'échantillon :

$$\overline{x} = \frac{\sum_{i=1}^{n} x_i}{n}$$

et de son écart-type :

$$\text{std}(x) = \sqrt{\frac{\sum_{i=1}^{n} (x_i - \overline{x})^2}{n-1}}$$

On obtient donc l'estimation suivante pour les paramètres :

$$\begin{cases} \tilde{a} &= \pi * \sqrt{\dfrac{\sum_{i=1}^{n} (\overline{x} - x_i)^2}{6 * (n-1)}} \\ \tilde{b} &= \gamma - \tilde{a} * \overline{x} \end{cases} \tag{B.5}$$

Ces estimations ne sont pas trop exactes. En prenant un échantillon d'une v.a. de Gumbel connue, les paramètres estimés sont assez loin des vrais paramètres. Dans la figure B.1, nous avons représenté la distribution d'une v.a. de Gumbel de paramètres connus et la distribution de Gumbel avec les paramètres estimés.

B.3.2 La méthode du maximum de vraisemblance

Une autre méthode générique pour estimer les paramètres d'une loi de probabilité est la méthode du maximum de vraisemblance. Elle consiste à chercher le maximum de la fonction :

$$L(x_1, x_2, \cdots, x_n; a, b) = \prod_{i=1}^{n} f(x_i)$$

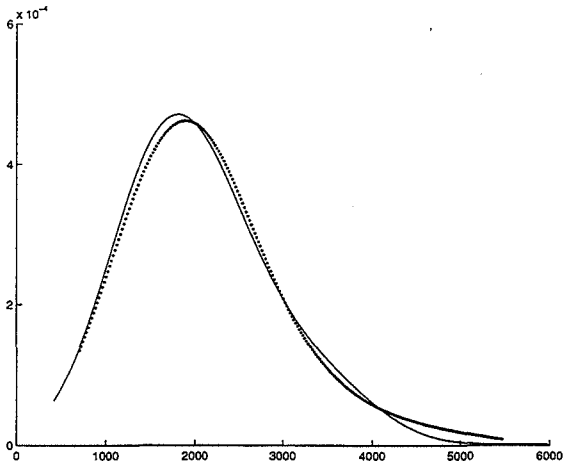


FIG. B.1 – *La distribution d'une loi de Gumbel(a, b)(en continue) et la distribution de la loi de Gumbel($\tilde{a}$, $\tilde{b}$) (en pointillé)* $a = 0.0017$, $b = -3$, $\tilde{a} = 0.001677$, $\tilde{b} = -2.90661$.

Ceci est équivalent à maximiser la fonction $\log L(x_1, x_2, \cdots, x_n; a, b)$ et donc à résoudre le système:

$$\begin{cases} \dfrac{\partial}{\partial a} \log L(x_1, x_2, \cdots, x_n; a, b) &= 0 \\ \\ \dfrac{\partial}{\partial b} \log L(x_1, x_2, \cdots, x_n; a, b) &= 0 \end{cases}$$

qui peut être écrit:

$$\begin{cases} \dfrac{n}{a} - \displaystyle\sum_{i=1}^{n} x_i - \sum_{i=1}^{n} x_i * e^{-(a*x_i+b)} &= 0 \\ \\ -n + \displaystyle\sum_{i=1}^{n} e^{-(a*x_i+b)} &= 0 \end{cases}$$

Ce système se résout par une méthode itérative. Comme point de départ des itérations on peut prendre, par exemple, $(\tilde{a}, \tilde{b})$ obtenus l'aide de (B.5).

Il est connu que la méthode du maximum de vraisemblance fournit des estimations des paramètres plus fines que la méthode des moments.

Bibliographie

[AA97] Agbadje (Pierre) et Antoni (Emanuel). – *Analyse post-mortem de la régulation de charges d'un réseau de processeurs par diffusion.* – Rapport de projet *Modélisation Mathématique*, Ecole Nationale Supérieure des Mines de St. Etienne, 1997.

[AG91] Ahmad (Ishfaq) et Ghafoor (Arif). – Semi-distributed load balancing for massively parallel multicomputer systems. *IEEE Transactions on Software Engineering*, vol. 17, n° 10, 1991, pp. 987–1004.

[AG94] Almasi (G. S.) et Gottlieb (A.). – *Highly Parallel Computing.* – Benjamin/Cummings, 1994, 2 édition.

[AK89] Aarts (Emile) et Korst (Jan). – *Simulated annealing and Boltzman Machines . A stochastic Approach to Combinatorial Optimization and Neural Computing.* – Wiley, 1989.

[AM93] Antonio (John K.) et Metzger (Richard C.). – Hypershere mapper: A nonlinear programming approach to the hypercube embedding problem. *In: Proceedings of the 7-th International Parallel Processing Symposium*, pp. 538–547.

[AP88] André (Françoise) et Pazat (Jean-Louis). – Le placement de tâches sur des architectures parallèles. *T.S.I.*, vol. 7, n° 4, 1988, pp. 385–401.

[ARC92] ARCHIPEL (édité par). – *VOLVOX Machine. Tutorial and Administration.* – 1992.

[AvL85] Aarts (E. H. L.) et van Laarhoven (P. J. M.). – Statistical cooling: a general approach to combinatorial optimization problem. *Philips J. Res.*, vol. 40, 1985, pp. 193–226.

[BA92] Bultan (Tevfik) et Anknat (Cevdet). – A new mapping heuristic based on mean field annealing. *Journal of Parallel and Distributed Computing*, vol. 16, n° 4, 1992, pp. 292–305.

[Bac95] Baccouche (Leïla). – *Un mécanisme d'ordonnancement distribué de tâches temps réel.* – Thèse de PhD, Institut National Polytechnique de Grenoble, novembre 1995.

[Bal92] Baluja (Shumeet). – *A massively distributed parallel genetic algorithm (mpdGA).* – Rapport technique, Carnegie Mellon University, octobre 1992. CMU-CS-92-196R.

[BB87] Berger (Marcha J.) et Bokhari (Shahid H.). – A partitioning strategy for nonuniform problems on multiprocessors. *IEEE Transactions on Computers*, vol. C-36, n° 5, mai 1987, pp. 570–580.

[BBK91] Boillat (J. E.), Bruge (F.) et Kropf (P. G.). – A dynamic load balancing algorithm for molecular dynamics simulation on multiprocessor systems. *J. Comp. Phys.*, vol. 96, n° 1, 1991, pp. 1–14.

[BCS89] Billionnet (Alain), Costa (Marie-Christine) et Sutter (Alain). – Les problèmes de placement dans le systèmes distribués. *T.S.I.*, vol. 8, n° 4, 1989, pp. 309–337.

[BCS92] Billionnet (A.), Costa (M. C.) et Sutter (A.). – An efficient algorithm for a task allocation problem. *Journal of the ACM*, vol. 39, n° 3, juillet 1992, pp. 502–518.

[BE94] Baykal (N.) et Erciyes (K.). – Dynamic load balancing in a parallel processing system. *In: Second International Conference on Software for Multiproces sor and Supercomputing Theory*, pp. 370–379.

[Ben96] Benaichouche (M.). – *Parallélisation des méthodes de recherche arborescente dans les environnements distribués.* – Thèse de PhD, Université Paris 6, 1996.

[Ber83] Berge (Claude). – *Graphes.* – Gauthier–Villars, 1983.

[BF90] Brugè (F.) et Fornili (S. L.). – A distributed dynamic load balancer and its implementation on multi-transuper systems for molecular dynamics simulation. *Computer Physics Communications*, vol. 60, 1990, pp. 39–45.

[BF92a] Boutaba (Raouf) et Folliot (Bertil). – *Load balancing in local area networks.* – Rapport technique, Université Paris VI-VII, Institut Blaise Pacsal, juin 1992. MASI 92.38.

[BF92b] Boutaba (Raouf) et Folliot (Bertil). – *Presentation of a Multicriteria Load Balancing.* – Rapport technique, Université Paris VI-VII, Institut Blaise Pascal, juin 1992. MASI 92.42.

[BF96] Bernard (Guy) et Folliot (Bertil). – Caractéristiques générales du placement dynamique : synthèse et problématique. *In: Placement Dynamique et Répartition de Charge: application aux systèmes parallèles et répartis, Ecole d'été, Presqi'Île de Giens*, pp. 3–22.

[BG89] Boulouias (A.) et Gopal (P. M.). – Some graph partinioning problems and algorithms related to routing in large computer networks. *In: 9th International Conference on Distributed Computing*, pp. 362–369.

[BGM97] Briottet (L.), Gilormini (P.) et Montheillet (F.). – Approximation analytical equations for the deformation of an inclusion in viscoplastic metric. 1997. – submitted for publication to *Acta Mecanica.*

[BGR96] Briat (J.), Gautier (T.) et Roch (J.L.). – On-line scheduling. *In: ESPPE'96*, pp. 95–108.

[BK90] Boillat (J. E.) et Kropf (P.G.). – A fast distributed mapping algorithm. *In: CONPAR'90, LNCS 457.* pp. 405–416. – Springer.

[Ble89] Blelloch (Guy E.). – Scans as primitive parallel operations. *IEEE Transactions on Computers*, vol. 38, n° 11, novembre 1989, pp. 1526–1538.

[BNG92] Browen (N. S.), Nikolaou (C. N.) et Ghafoor (A.). – On the assignment problem of arbitrary process systems to heterogeneous distributed computer systems. *IEEE Transactions on Computers*, vol. 41, n° 3, mars 1992, pp. 257–273.

[Boi88] Boillat (J. E.). – Load balancing and Poisson equation in a graph. 1988. – draft paper.

[Boi90] Boillat (J. E.). – Load balancing and Poisson equation in a graph. *Concurrency : Practice and Experience*, vol. 2, n° 4, décembre 1990, pp. 289–313.

[Bok79] Bokhari (Shahid H.). – Dual processor scheduling with dynamic reassignement. *IEEE Transactions on Software Engineering*, vol. SE-5, n° 4, juillet 1979, pp. 341–349.

[Bok81a] Bokhari (S. H.). – A shortest tree algorithm for optimal assignement across space and time in a distributed processor system. *IEEE Transactions on Software Engineering*, vol. SE-7, n° 6, novembre 1981, pp. 583–589.

[Bok81b] Bokhari (Shahid H.). – On the mapping problem. *IEEE Transactions on Computers*, vol. C-30, n° 3, mars 1981, pp. 207–214.

[Bok88] Bokhari (Shahid H.). – Partitioning problems in parallel, pipelined and distributed computing. *IEEE Transactions on Computers*, vol. 37, n° 1, janvier 1988, pp. 48–57.

[BP86] Barak (Amnon) et Paradise (On G.). – MOS - A Load-Balancing UNIX. *In : EUUG Autumn '86*, pp. 273–280.

[BP92] Baxter (J.) et Patel (J. H.). – Profiling based task migration. *In : Proceedings of the 6-th International Parallel Processing Sympos ium*, pp. 192–195.

[BS79] Bazaraa (M. S.) et Shetty (C. M.). – *Nonlinear Programming : Theory and Applications.* – John Wiley & Sons, 1979.

[BS85] Barak (Amnon) et Shiloh (Amnon). – A distributed load–balancing policy for a multicomputer. *Software – Practice and Experience*, vol. 15, n° 9, septembre 1985, pp. 901–913.

[BS94] Barnard (S. T.) et Simon (H. D.). – A fast multilevel implementation of recursive spectral bisection. *Concurrency : Practice and Experience*, vol. 6, n° 2, 1994, pp. 101–117.

[BSS91] Bernard (G.), Steve (D.) et Simatic (M.). – Placement et migration de processus dans les systèmes répartis faiblement couplés. *T.S.I.*, vol. 10, n° 5, 1991, pp. 355–373.

[BY95] Boissonnat (J.D.) et Yvinec (M.). – *Géométrie algorithmique.* – Ediscience Intl., 1995.

[CE94] Calinescu (R.) et Evans (D. J.). – A parallel simulation model for load balancing in clustered distributed systems. *Parallel Computing*, vol. 20, 1994, pp. 77–91.

[Cer93] Cerf (Raphaël). – *Asymptotic Convergence of Genetic Algorithms.* – Rapport de recherche no 93-03, Université Montpellier, 1993.

[CG88] Chen (E.K.) et Gehringer (E.F.). – A graph-oriented mapping strategy for a hypercube. *In : International Conference on Hypercube Concurrent Computers and Applications*, pp. 200–209.

[CK88] Casavant (Thomas L.) et Kuhl (John G.). – A taxonomy of scheduling in general-purpose distributed computing systems. *IEEE Transactions on Software Engineering*, vol. 14, n° 2, février 1988, pp. 141–154.

[CL91] Coroyer (Christophe) et Liu (Zhen). – *Effectiveness of heuristics and simulated annealing for the scheduling of concurrent tasks - an empirical comparison.* – Rapport de recherche no. 1379, INRIA, janvier 1991.

[CLK92] Cho (Sang-Young), Lee (Cheol-Hoon) et Kim (Myunghwan). – Optimal task assignment in hypercube networks. *IEICE Trans. Fundamentals*, vol. E75-A, n° 4, avril 1992, pp. 504–511.

[CS93] Calzarossa (Maria) et Serazzi (Giuseppe). – Workload Caracterization : A Survey. *Proceedings of the IEEE*, vol. 81, n° 8, août 1993, pp. 1136–1150.

[CT93] Cosnard (Michel) et Trystram (Denis). – *Algorithmes et architectures parallèles.* – Inter-Editions, 1993.

[Cyb89] Cybenko (George). – Dynamic load balancing for distributed memory multiprocessors. *Journal of Parallel and Distributed Computing*, vol. 7, 1989, pp. 279–301.

[DG89] Dragon (Karen M.) et Gustafson (John L.). – A low-cost hypercube load-balance algorithm. *In : Conference on Hypercube Concurrent Computations and Applications*, pp. 583–589.

[dG91] de Souza e Silva (E.) et Gerla (M.). – Queueing network models for load balancing in distributed systems. *Journal of Parallel and Distributed Computing*, vol. 12, n° 1, mai 1991, pp. 24–38.

[DM94] Dutt (Shantanu) et Mahapatra (Nihar R.). – Scalable load balancing strategies for parallel A* algorithms. *Journal of Parallel and Distributed Computing*, vol. 22, 1994, pp. 488–505.

[Dow95] Dowaji (Salah). – Régulation de charge pour des méthodes exactes d'optimisation. *In : Journées de Recherche sur le Placement Dynamique et la Répartition de Charge : Application aux Systèmes Répartis et Parallèles*, pp. 57–59.

[dR95] de Rumeur (J.). – *Communications dans les réseaux de processeurs.* – Masson, 1995.

[Dub96] Duboux (Thibault). – Régulation dynamique du partionnement d'une structure géométrique soumise à des requêtes. *Calculateurs parallèles*, vol. 8, n° 1, 1996, pp. 103–118.

[EAH90] Eiben (A. E.), Aarts (E. H. L.) et Hee (K. M. Van). – Global convergence of genetic algorithms : a Markov chain analysis. 1990, pp. –.

[EB93] Evans (D. J.) et Butt (W. U. N.). – Dynamic load balancing using task-transfer probabilities. *Parallel Computing*, vol. 19, 1993, pp. 897–916.

[EB94] Evans (D. J.) et Butt (W. U. N.). – Load balancing with network partitioning using host groups. *Parallel Computing*, vol. 20, 1994, pp. 325–345.

[Efr82] Efron (Bradley). – *The Jackknife, the Bootstrap and Other Resampling Plans.* – Society for Industrial and Applied Mathematics, 1982.

[Ell94] Elleuch (Ahmed). – *Migration de processus dans les systèmes massivement parallèles.* – Thèse de PhD, Institut National Polytechnique de Grenoble, novembre 1994.

[ELZ86] Eager (Derek L.), Lazowska (Edward D.) et Zahorjan (John). – Adaptive load sharing in homoegeneous distributed systems. *IEEE Transactions on Software Engineering*, vol. SE-12, n° 5, mai 1986, pp. 662–675.

[ELZ88] Eager (D. L.), Lazowska (E. D.) et Zahorjan (J.). – The limited performance benefits of migrating active processes for load sharing. *ACM SIGMETRICS Perfomances Evaluation Review*, vol. 16, n° 1, mai 1988, pp. 63–72.

[EM94] Elleuch (A.) et Muntean (T.). – Process migration protocols for massively parallel systems. *In : Proceedings of the 1st International Conference on Massively Parallel Computing Systems.* pp. 84–95. – Los Alamitos, CA, USA, mai 1994.

[ERAL95] El-Rewini (H.), Ali (H. H.) et Lewis (T.). – Task scheduling in multiprocessing systems. *Computer*, vol. 28, n° 12, 1995, pp. 27–37.

[ERS90] Ercal (F.), Ramanujan (J.) et Sadayappan (P.). – Task allocation onto a hypercube by recursive mincut bipartitioning. *Journal of Parallel and Distributed Computing*, vol. 10, 1990, pp. 35–44.

[FDM93] Fonlupt (Cyril), Dekeyser (Jean-Luc) et Marquet (Philippe). – *Redistribution dynamique des données sur machine massivement parallèle.* – Rapport technique, Laboratoire d'Informatique Fondamentale de Lille, juin 1993. ERA-128.

[FHR94] Falk (Michael), Hüsler (Jürg) et Reiss (Rolf-Dieter). – *Laws of Small Numbers : Extremes and Rare Events.* – Birkhäuser Verlag, 1994.

[Fie75] Fiedler (Miroslav). – A proprety on eigenvectors of nonnegative symetric matrices and its application to graph theory. *Czechoslovak Mathematical Journal*, vol. 25, n° 100, 1975, pp. 619–633.

[FKW87] Fox (G.C.), Kolawa (A.) et Williams (R.). – *The implementation of a dynamic load balancer*, pp. 114–121. – in M.T. Heath (ed.) *Hypercube Multiprocessors*, SIAM, 1987.

[Fly72] Flynn (Michael J.). – Some computer organizations and their effectiveness. *IEEE Transactions on Computers*, vol. C-21, 1972, pp. 948–960.

[Fon94] Fonlupt (Cyrille). – *Distribution dynamique de données sur machine parallèles SIMD.* – Thèse de PhD, Université des Sciences et Technologies de Lille, 1994.

[FS93] Folliot (Bertil) et Sens (Pierre). – *Fault tolerence and load balancing management in distributed systems.* – Masi 93.03, Institut Blaise Pascal, janvier 1993.

[Gan66] Gantmacher (F. R.). – *Théorie des matrices.* – Dunod, 1966.

[Gas93] Gastaldo (Michel). – *Dictionary Machine on SIMD Architectures.* – Research report no 93-19, LIP Lyon, juillet 1993.

[GHS83] Gallager (R. G.), Humblet (P.) et Spira (P.). – A distributed algorithm for minimum-weight spanning trees. *ACM Transactions on Programming Languages and Systems*, vol. 5, 1983, pp. 66–75.

[GJ79] Garey (Michael R.) et Johnson (Davis S.). – *Computers and Intractability. A Guide into the Theory of NP-Completeness.* – W. H. Freeman and Company, 1979.

[GKMM96] Gourdet (S.), Konopleva (E. V.), McQueen (H. J.) et Montheillet (F.). – Recrystallization during hot deformation of aluminium. *In: Colloque ICAA5.*

[GLM+96] Ghosh (B.), Leighton (F.T.), Maggs (B. M.), Muthukrishnan (S.) et al. – Tight analyses of two local load balancing algorithms. 1996. – draft.

[Glo89] Glover (Fred). – Tabu Search — Part I. *ORSA Journal on Computing*, vol. 1, n° 3, 1989, pp. 190–206.

[Glo90] Glover (Fred). – Tabu Search — Part II. *ORSA Journal on Computing*, vol. 2, n° 1, 1990, pp. 4–32.

[GM94] Ghosh (B.) et Muthikrishnan (S.). – Dynamic load balancing in parallel and distributed networks by random matchings. *In: 6-th Annual ACM Symposium on Parallel Algorithms and Architectures*, pp. 226–235.

[GM95] Gourdet (S.) et Montheillet (F.). – Recristallisation continue au cours de la déformation à chaud d'un aluminium 1200. *Journal de Physique IV*, vol. 5, supplément au *Journal de Physique III*, April 1995, pp. C3–255—C3–260.

[GO93] Gerogiannis (D.) et Orphanoudakis (S. C.). – Load balancing requirements in parallel implementations of image feature extraction tasks. *IEEE Transactions on Parallel and Distributed Systems*, vol. 4, n° 9, septembre 1993, pp. 994–1013.

[Gol88] Goldberg (David E.). – *Genetic Algorithms in Search, Optimization, and Machine Learning.* – Addison – Wesley, 1988.

[Gou97] Gourdet (Sophie). – *Etude des mécanismes de recristallisation au cours de la déformation à chaud de l'aluminium.* – Thèse de PhD, Ecole Nationale des Mines de St. Etienne, octobre 1997.

[GRS91] Germain-Renaud (Cecile) et Sansonnet (Jean-Paul). – *Les ordinateurs massivement parallèles.* – Armand Colin, 1991.

[GS87] Goldberg (David E.) et Segrest (Philip). – Finite Markov chain analysis of genetic algorithms. *In: The 2-nd International Congress of Genetic Algorithms*, pp. 1–7.

[GS89] Goeges-Schleuter (Martina). – ASPARAGOS A Parallel Genetic Algorithm and Population Genetics. *In: Parallelism, Learning, Evolution*, pp. 407–418.

[GS93] Guyennet (H.) et Spies (F.). – A cooperative algorithm for load balancing in interconnected transputer network. *In: Descentralized and Distributed System*, éd. par Cosnard (M.) et Puigjaner (R.), pp. 305–315.

[GT86] Goldberg (Andrew V.) et Tarjan (Robert E.). – A new approach to the maximum flow problem. *In: Theory of Computing – the 18-th Annual Symposium*, pp. 136–146.

[GTdW93] Glover (Fred), Taillard (Eric) et de Werra (Dominique). – A user's guide to tabu search. *Annals of Operations Research*, vol. 41, 1993, pp. 3–28.

[Hac89] Hac (Anna). – Load balancing in distributed systems: A summary. *Performance Evaluation Review*, vol. 16, n° 2, février 1989, pp. 17–19.

[Had92] Haddad (Emile). – Optimal allocation of shared data over distributed memory hierarchies. *In: Proceedings of the 6-th International Parallel Processing Sympos ium*, pp. 518–526.

[Hei94] Heirich (Alan). – *Scalable Load Balancing by Diffusion.* – Caltech–CS–TR–94–04, California Institut of Technology, 1994.

[HJ90] Haćˊ (Anna) et Johnson (Theodore J.). – Sensitivity study of the load balancing algorithm in a distributed system. *Journal of Parallel and Distributed Computing*, vol. 10, 1990, pp. 85–89.

[HL86] Hsu (Chi-Yin Huang) et Liu (Jane W.-S.). – Dynamic load balancing algorithms in homogeneous distributed systems. *In: IEEE*, pp. 216–223.

[HLM+90] Hosseini (S. H.), Litow (B.), Malkawi (M.), McPherson (J.) et Vairavan (K.). – Analysis of a graph coloring based distributed load balancing algorithm. *Journal of Parallel and Distributed Computing*, vol. 10, 1990, pp. 160–166.

[Hor93] Horton (G.). – A multi-level diffusion method for dynamic load balancing. *Parallel Computing*, vol. 19, 1993, pp. 209–218.

[HT95] Heirich (Alan) et Taylor (Stephen). – A parabolic load balancing method. 1995.

[INM89] INMOS (édité par). – *The Transputer Databook.* – 1989.

[JCS96] Juganaru (Mihaela), Carraro (Laurent) et Sakho (Ibrahima). – Une analyse probabiliste de la complexité en temps d'un algorithme de régulation de charge. *In: Placement Dynamique et Répartition de Charge: application aux systèmes parallèles et répartis, Ecole d'été, Presqi'Île de Giens*, pp. 213–218.

[JM93] Jacqmot (C.) et Milgrom (E.). – A systematic approach to load distibution strategies for distributed systems. *In: Descentralized and Distributed Systems*, éd. par Cosnard (M.) et Puigjaner (R.), pp. 291–303.

[JMSM97a] Juganaru (M.), Maurice (C.), Sakho (I.) et Montheillet (F.). – Parallel implementation of a large-strain polycrystal deformation model. *In: International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'97).* – Las Vegas.

[JMSM97b] Juganaru (M.), Maurice (C.), Sakho (I.) et Montheillet (F.). – A parallel simulation of a quantitative large-strain polycrystal deformation. *In: Euro-Par'97 Conference.* – Passau, Allemagne.

[JMSM97c] Juganaru (M.), Maurice (C.), Sakho (I.) et Montheillet (F.). – A topology based model for large–strain deformation of polycrystals. *In: International Conference on The Quantitative Description of Materials Microstructure.* – Warsaw, Pologne.

[JR92] Jájá (Joseph) et Ryu (Kwan Woo). – Load balancing and routing on the hypercube and related networks. *Journal of Parallel and Distributed Computing*, vol. 14, 1992, pp. 431–435.

[JS96] Juganaru (Mihaela) et Sakho (Ibrahima). – Répartition dynamique de données régulières pour des machines MIMD homogènes à mémoire distribuée. *Calculateurs parallèles*, vol. 8, n° 1, 1996, pp. 89–102.

[Kal88] Kalé (L. V.). – Comparing the performance of two dynamic load distribution methods. *In: International Conference on Parallel Processing*, pp. 8–12.

[KB88] Kim (S. J.) et Browne (J. C.). – A genearal approach to mapping of parallel computations upon multiprocessor architectures. *In: International Conference on Parallel Processing*, pp. 1–8.

[KG94] Kumar (Vipin) et Gupta (Anshul). – Analyzing scalability of parallel algorithms and architecture. *Journal of Parallel and Distributed Computing*, vol. 22, n° 379–391, 1994.

[KK93] Kalé (L. V.) et Krishnan (S.). – A comparison based parallel sorting algorithm. *In: International Conference on Parallel Processing*, pp. 196–200.

[Kle75] Kleinrock (Leonard). – *Queueing Systems.* – J. Wiley & Sons, 1975volume 1 : Theory.

[KM87] Krämer (O.) et Mühlenbein (H.). – Mapping strategies in message based multiprocessor systems. *In: PARLE'87, LNCS.* pp. 213–225. – Springer-Verlag.

[KNN87] Kunii (Tosiyasu L.), Nishimura (Satoshi) et Noma (Tsukasa). – *The Design of a Parallel Processing System for Computer Graphics.* – Technical report 87–28, University of Tokyo, décembre 1987.

[Koh95] Kohring (G. A.). – Dynamic load balancing for parallelized particle simulations on MIMD computers. *Parallel Computing*, vol. 21, 1995, pp. 683–693.

[KR93] Keyser (J. De) et Roose (D.). – Load balancing data parallel programs on distributed memory computers. *Journal of Parallel and Distributed Computing*, vol. 19, n° 11, novembre 1993, pp. 1199–1221.

[KRS84] Korach (E.), Rotem (D.) et Santoro (N.). – Distributed algorithms for finding centers and medians in a network. *ACM Transactions on Programming Languages and Systems*, 1984, pp. 380–391.

[KS93] Kon'ya (Seiichi) et Satoh (Tetsuji). – Task scheduling on a hypercube with link contentions. *In: Proceedings of the 7-th International Parallel Processing Symposium*, pp. 363–368.

[KS94] Kumar (Vipin) et Sun (Xian-He). – Special issue on scalability of parallel algorithms and architectures. *Journal of Parallel and Distributed Computing*, vol. 22, 1994, pp. 375–378.

[LA87] Lee (S.-Y.) et Aggrawal (J. K.). – Mapping strategy for parallel processing. *IEEE Transactions on Computers*, vol. 36, n° 4, avril 1987, pp. 433–442.

[Lam91] Lamour (Françoise). – *Contributions aux problèmes de placement sans routage sur un réseau de transputers.* – Thèse de PhD, Université Paris 6, 1991.

[Lav95] Lavault (Christian). – *Evaluation des algorithmes distribués.* – Editions Hermès, 1995.

[Law92] Lawton (George). – Genetic algorithms for schedule optimization. *AI Expert*, mai 1992, pp. 23–24.

[Lei93] Leighton (F. Thomson). – *Introduction to Parallel Algorithms and Archiectures.* – 1993.

[LK87] Lin (Frank C. H.) et Keller (Robert M.). – The gradient model load balancing method. *IEEE Transactions on Software Engineering*, vol. SE-13, n° 1, janvier 1987, pp. 32–38.

[LL94] Lennerstad (H.) et Lundberg (L.). – Optimal scheduling results for parallel computing. *SIAM News*, August/September 1994, pp. 16–18.

[LLK92] Lee (Cheol-Hoon), Lee (Dongmyun) et Kim (Myunghwan). – Optimal task assignment in linear array networks. *IEEE Transactions on Computers*, vol. 41, n° 7, 1992, pp. 877–880.

[LM92] Lin (Zheng) et Minker (Jack). – A distributed load balancing scheme for parallel logic programming. *In: International Conference of Parllel Processing*, pp. 66–73.

[LMR91] Lüling (R.), Monien (B.) et Ramme (F.). – Load balancing in large networks: A comparative study. *In: 3-th IEEE Symposium on Parallel and Distributed Processing*, pp. 686–689.

[Lo84] Lo (Virginia Mary). – Heuristic algorithms of task assignement in distributed systems. *In: International Conference on Distributed Computer Systems*, pp. 30–39.

[Lo85] Lo (Virginia Mary). – Task assignment to minimize completion time. *In: Proceedings of the 5th International Conference on Distributed Computing*, pp. 329–336.

[LO86] Leland (Will E.) et Ott (Teunis J.). – Load-balancing and process behavior. *ACM SIGMETRICS Performance Evaluation Review*, vol. 14, n° 1, 1986, pp. 54–69.

[LRCM95] Luque (E.), Ripoll (A.), Cortès (A.) et Margalef (T.). – A distributed diffusion method for dynamic load balancing on parallel computers. 1995. – http://www.cpc.wmin.ac.uk/Copernicus/task4.html.

[LS96] Luce (Frédéric) et Stamm (Guillaume). – *Etude de l'équilibre entre le temps de calcul et le temps de répartition de la charge de travail en architecture parallèle.* – Rapport de projet *Modélisation Mathématique*, Ecole Nationale Supérieure des Mines de St. Etienne, 1996.

[LT94] Lu (Hongjun) et Tan (Kian-Lee). – Load-balanced join processing in shared-nothing systems. *Journal of Parallel and Distributed Computing*, vol. 23, 1994, pp. 382–398.

[man95a] *ANSI C Language and Libraries Reference Manual.* – SGS-Thomson Microelectronics, mai 1995.

[man95b] *ANSI C Toolset User Guide.* – SGS-Thomson Microelectronics, mai 1995.

[man95c] *Toolset Reference Manual.* – SGS-Thomson Microelectronics, mai 1995.

[MDLT96] Melab (N.), Devesa (N.), Lacouffe (M. P.) et Toursel (B.). – *Adaptive Load Balanging and Irregular Applications.* – Rapport technique, Laboratoire d'Informatique Fondamentale de Lille, juin 1996. AS-173.

[MG94] Montheillet (Frank) et Gilormini (Pierre). – Prévision du comportement mécanique d'un mélange de deux phases viscoplastiques linéaires à l'aide d'un modèle du type automate cellulaire. *C. R. Acad. Sci. Paris, T. 319, Série 2*, 1994, pp. 483–490.

[MG96] Montheillet (Frank) et Gilormini (Pierre). – Predicting the Mechanical Behavior of Two–Phase Materials with Cellular Automata. *Int. Journal of Plasticity*, vol. 12, n° 4, 1996, pp. 561–574.

[Mig92] Miguet (Serge). – *Redistribution élastique pour équilibrage de charge en imagerie*, chap. 14, pp. 229–240. – in M. Cosnard and M. Nivat and Y. Robert (ed.) *Algorithmique parallèle*, Masson, Paris, 1992.

[MJ96] Montheillet (F.) et Jonas (J. J.). – *Recrystallization, Dynamic.* In *Encyclopedia of Applies Physics*, pp. 205–225. – VCH Publishers, 1996volume 16.

[MLT82] Ma (P.-Y. R.), Lee (E. Y. S.) et Tsuchiya (M.). – A task allocation model for distributed computing systems. *IEEE Transactions on Computers*, vol. C-31, n° 1, janvier 1982, pp. 41–47.

[MM89] Magirou (V. F.) et Milis (J. Z.). – An algorithm for the multiprocessor assignement problem. *Operations Research Letters*, vol. 8, 1989, pp. 351–356.

[MR91] Marinescu (Dan C.) et Rice (John R.). – Synchronization and load imbalance effects in distributed memory multi-processor systems. *Concurrency: Practice and Experience*, vol. 3, n° 6, décembre 1991, pp. 593–625.

[MRW93] Mehrotra (Kishan), Ranka (Sanjav) et Wang (Jhy-Chun). – A probabilistic analysis of a locality maintaining load balancing algorithm. *In: Proceedings of the 7-th International Parallel Processing Symposium*, pp. 369–373.

[MSK87] Muhlenbein (H.), Schleuter (M. Gorges) et Kramer (O.). – New solutions to the mapping problem of parallel systems: The evolution approach. *Parallel Computing*, 1987, pp. 269–279.

[MT91a] Muntean (T.) et Talbi (E.-G.). – Methodes de placement statique des processus sur architectures paralleles. *T.S.I.*, vol. 10, n° 5, 1991, pp. 355–373.

[MT91b] Muntean (T.) et Talbi (E.-G.). – A paralle genetic algorithm for process-processors mapping. *In: Internationnal Conference on High Speed Computing*, éd. par Durand (M.) et Dabaghi (F. El). pp. 71–82. – North Holland.

[Müh89] Mühlenbein (H.). – Parallel genetic algorithms, population genetics and combinatorial optimization. *In: Parallelism, Learning, Evolution*, pp. 398–406.

[MZ95] Muniz (F. J¿) et Zaluska (E. J.). – Parallel load-balancing : An extension to the gradient model. *Parallel Computing*, vol. 21, 1995, pp. 287–301.

[NC88] Nishizeki (T.) et Chiba (N.). – *Planar Graphs: Theory and Algorithms.* – North-Holland, 1988.

[Neu66] Neumann (J. V.). – *Theory of self reproducing automata.* – University of Illinois Press, 1966.

[Nic92] Nicol (David M.). – Communication efficient global load balancing. *In: Scalable High Performance Computing Conference.*

[Nor93] Norre (Sylvie). – *Problèmes de placement de tâches: méthodes stochastiques et évaluation des performances.* – Thèse de PhD, Université Blaise Pascal, Clermont–Ferrand, 1993.

[NV92] Nix (Allen E.) et Vose (Michael D.). – Modeling genetic algorithms with Markov chains. *Annals of Mathematics and Artificial Intelligence*, vol. 5, 1992, pp. 79–88.

[NXG85] Ni (Lionel M.), Xu (Chong-Wei) et Gendreau (Thomas B.). – Distributed drafting algorithm for load balancing. *IEEE Transactions on Software Engineering*, vol. SE-11, n° 10, octobre 1985, pp. 1153–1161.

[Pel95] Pellegrini (François). – *Application de méthodes de patition à la résolution de problèmes de graphes issus du parallélisme.* – Thèse de PhD, Université Bordeuax I, janvier 1995.

[PR93] Porto (Stella C. S.) et Ribeiro (Celso C.). – *A Tabu Search Approach to Task Scheduling on Heterogeneous Processors under Precedence Constraints.* – in *Monografias en Ciéncia da Computaçao, no. 03/93*, 1993.

[PR96] Pellegrini (F.) et Roman (J.). – SCOTCH: A software package for static mapping by dual recursive bipartitioning of process and architecture graphs. *Lecture Notes in Computer Science*, vol. 1067, 1996, pp. 493–??

[PTS88] Pulidas (Spiridom), Towsley (Don) et Stankovic (John A.). – Imbedding gradient estimators in load balancing algorithms. *In: 8th International Conference on Distributed Computing Systems*, pp. 482–490.

[PU86] Peleg (David) et Upfal (Eli). – The token distribution problem. *In: Symposium on Foundations of Computer Science*, pp. 418–427.

[Rom97] Roman Alonso (Graciela). – *Contribution à l'étude du placement dynamique sur machines parallèles de type MIMD.* – Thèse de PhD, Université de Technologie de Compiègne, juin 1997.

[Ros95] Rosenthal (Jeffrey S.). – Convergence rates for Markov chains. *SIAM Review*, vol. 37, n° 3, septembre 1995, pp. 387–405.

[RSAU91] Rudolph (L.), Slivkin-Allalouf (M.) et Upfal (Eli). – A simple load balancing scheme for task allocation in parallel machines. *In: 3-rd Annual ACM Symposium on Parallel Algorithms and Architecture*, pp. 237–245.

[Rud94] Rudolph (Günter). – Convergence analysis of canonical genetic algorithms. *IEEE Transactions on Neural Networks*, vol. 5, n° 1, janvier 1994, pp. 96–101.

[RVV94] Roch (J.-L.), Vermeerbergen (A.) et Villard (G.). – *A new load-prediction scheme based on algorithmic cost function.* – Rapport technique, LMC–IMAG, 1994.

[Sal90] Saletore (Vikram A.). – A distributed and adaptive dynamic load balancing scheme for parallel processing of medium–grain tasks. *In: Distributed Memory Computing Conference*, pp. 994–999.

[Sap90] Saporta (G.). – *Probabilités. Analyse de données et statistique.* – Editions Technip, 1990.

[SB78] Stone (H.S.) et Bokhari (S. H.). – Control of distribted processes. *Computer*, vol. 11, n° 7, juillet 1978, pp. 97–106.

[SE87a] Sadayappan (P.) et Ercal (F.). – Cluster-partitioning approaches to mapping parallel programs onto a hypercube. *In: Supercomputing '87.* pp. 475–497. – LNCS.

[SE87b] Sadayappan (P.) et Ercal (F.). – Nearest-neighbor mapping of finite element graphs onto processor meshes. *IEEE Transactions on Computers*, vol. C-36, n° 12, décembre 1987, pp. 1408–1424.

[Sin87] Sinclair (J. B.). – Efficient computation of optimal assignments for distributed tasks. *Journal of Parallel and Distributed Computing*, vol. 4, 1987, pp. 342–362.

[SKS92] Shivaratri (Miranjan G.), Krueger (Phillip) et Singhal (Mukesh). – Load distributing for locally distributed systems. *IEEE Computer*, vol. 25, n° 12, décembre 1992, pp. 33–44.

[SRG94] Singh (Jaswinder Pal), Rothbegr (Edward) et Gupta (Anoop). – Modeling communication in parallel algorithms: A fruitful interaction between theory and systems. *6-th Annual ACM Symposium on Parallel Algorithms and Architectures*, 1994, pp. 189–199.

[SS94] Subramanian (R.) et Scherson (I. D.). – An analysis of diffusive load–balancing. *In: 6-th Annual ACM Symposium on Parallel Algorithms and Architectures*, pp. 220–225.

[Sto77] Stone (Harold S.). – Multiprocessor scheduling with the aid of network flow algorithms. *IEEE Transactions on Software Engineering*, vol. SE-3, n° 1, janvier 1977, pp. 85–93.

[SU87] Shamir (Eli) et Upfal (Eli). – A probabilistic approach to the load-sharing problem in distributed systems. *Journal of Parallel and Distributed Computing*, vol. 4, 1987, pp. 521–530.

[Tai93] Taillard (E. D.). – *Recherches itératives parallèles.* – Thèse de PhD, Ecole polytechnique Fédérale de Laussane, 1993.

[Tal93] Talbi (E.-G.). – *Allocation de processus sur les architectures parallèles a mémoire distribueé.* – Thèse de PhD, INPG, 1993.

[Tal94] Talbi (El-Ghazali). – Configuration automatique d'une architecture parallèle. *Calculateurs Parallèles*, vol. 21, Mars 1994, pp. 7–27.

[Tal97] Talbi (E.G.). – Une taxonomie des algorithmes d'allocation dynamique des processus dans les systèmes parallèles et distribués. *In: ISYPAR'97, 2-ème Ecole d'Informatique des Systèmes Parallèles et Répartis*, pp. 137–164.

[Tar94] Tarnvik (Erik). – Dynamo - a portable tool for dynamic load balancing on distributed memory multicompters. *Concurrency: Practice and Experience*, vol. 6, n° 8, décembre 1994, pp. 613–639.

[Taw91] Tawbi (Nadia). – *Parallélisation automatique : estimation des durées d'exécution et allocation statique de processeurs.* – Thèse de PhD, Université Paris 6, 1991.

[TP94] Tron (Cécile) et Plateau (Brigitte). – Modelling of communication contention in multiprocessors. *In: Proceedings of the 7th International Conference on Modelling Techniques and Tools for Computer Performance Evaluation - Vienna, Austria*, éd. par Haring (Günter) et Kotsis (Gabriele). pp. 406–424. – Berlin, mai 1994.

[TT85] Tantawi (Asser N.) et Towsley (Don). – Optimal static load balancing in distributed computer. *Journal of the ACM*, vol. 32, n° 2, avril 1985, pp. 445–465.

[Tuk77] Tukey (John W.). – *Exploratory Data Analysis.* – Addison-Wesley, 1977.

[VLL90] Veltman (B.), Lageweg (B. J.) et Lenstra (J. K.). – Multiprocessor scheduling with communication delays. *Parallel Computing*, vol. 16, 1990, pp. 173–182.

[VR95] Van Driessche (R.) et Roose (D.). – An improved spectral bisection algorithm and its application to dynamic load balancing. *Parallel Computing*, vol. 24, 1995, pp. 29–48.

[vTW84] van Tilborg (André M.) et Whittie (Larry D.). – Wave scheduling – decentralized scheduling of task forces in multicomputers. *IEEE Transactions on Computers*, vol. C-33, n° 9, septembre 1984, pp. 835–844.

[Whi93] Whitley (Darrell). – *A Genetic Algorithm Tutorial.* – Technical report CS-93-103, Colorado State University, novembre 1993.

[Wil95] Wilson (Gregory V.). – *Practical Parallel Programming.* – MIT, 1995.

[WLR93] Willebeek-LeMair (M. H.) et Reeves (A. P.). – Strategies for dynamic load balancing on highly parallel computers. *IEEE Transactions on Parallel and Distributed Systems*, vol. 4, n° 9, septembre 1993, pp. 979–993.

[WM88] Wayne Bollinger (S.) et Midkiff (Scott F.). – Processor and link assignment in multicomputers using simulated annealing. *In: International Conference on Parallel Processing*, pp. 1–7.

[WM93] Woodside (C. Murray) et Monforton (Gerald G.). – Fast allocation of processes in distributed and parallel systems. *IEEE Transactions on Parallel and Distributed Systems*, vol. 4, n° 2, February 1993, pp. 164–174.

[WPG91] Wilkstrom (Milton C.), Prabhu (G. M.) et Gustafson (John L.). – Myths of load balncing. *In: Proceeding of Parallel Computing '91*, pp. 531–549.

[XH91] Xu (Jian) et Hwang (Kai). – Mapping rule-based systems onto multicomputers using simulated annealing. *Journal of Parallel and Distributed Computing*, vol. 13, 1991, pp. 442–454.

[XH93] Xu (Jian) et Hwang (Kai). – Heuristic methods for dynamic load balancing in a message–passing multicomputer. *Journal of Parallel and Distributed Computing*, vol. 18, 1993, pp. 1–13.

[XL92] Xu (C. Z.) et Lau (F. C. M.). – Analysis of the generalized dimension exchange method for dynamic load balancing. *Journal of Parallel and Distributed Computing*, vol. 16, 1992, pp. 385–392.

[XL93] Xu (C. Z.) et Lau (F. C. M.). – Optimal parameters for load balancing using the diffusion method in a k-ary n-cube networks. *Information Processing Letters*, vol. 47, 1993, pp. 181–187.

[XL95] Xu (C. Z.) et Lau (F. C. M.). – The generalized dimension exchange method for load balancing in$k$-ary n-cubes and variants. *Journal of Parallel and Distributed Computing*, vol. 24, 1995, pp. 72–85.

[You96] Young (Gilbert H.). – Complexity of task assignement in distributed systems. *In: PDPTA '96 International Conference*, pp. 779–782.

[Zat85] Zatti (Stephano). – *A multivariable information scheme to balance the load in a distributed system.* – Report No. UCB/CSD 85/234, Computer Science Division, University of California Berkeley, mai 1985.

[ZZWD93] Zhou (S.), Zheng (X.), Wang (J.) et Delisle (P.). – Utopia: a Load Sharing Facility for Large, Heterogenous Distributed Computer Systems. *Software – Practice and Experience*, vol. 23, n° 11, décembre 1993, pp. 1305–1336.

Bibliographie personnele

- **Publications dans des revues avec comité de séléction**

  - M. Juganaru, I. Sakho Répartition dynamique de données régulières pour des machines MIMD homogènes à mémoire distribuée. *Calculateurs parallèles*, 8(1):89–102, 1996.
  - M. Juganaru, I. Sakho, L. Carraro. A probabilistic complexity analysis based load balancing algorithm. In *The 11-th International Conference on Mathematical and Computer Modelling and Scientific Computing*, Washington, 31 mars - 3 avril 1997. A paraître dans *Mathematical Modelling and Scientific Computing*, Vol. 8, 1997

- **Conférences internationales avec comité de lecture**

  - Mihaela Juganaru, Ibrahima Sakho. Un algorithme distribué de régulation de charge sur un arbre. In *RenPar'7*, Mons, page 245, 30 mai - 2 juin 1995.
  - Mihaela Juganaru, Ibrahima Sakho, Laurent Carraro. Une analyse probabiliste de la complexité en temps d'un algorithme de régulation de charge. In *RenPar'8*, Bordeaux, page 209, 22 - 24 mai 1996.
  - M. Juganaru, C. Maurice, I. Sakho, F. Montheillet. A topology based model for large–strain deformation of polycrystals. In *International Conference on The Quantitative Description of Materials Microstructure*,Warsaw, 16 - 19 April 1997, Polland.
  - M. Juganaru, C. Maurice, I. Sakho, F. Montheillet. Parallel Implementation of a Large-Strain Polycrystal Deformation Model. In *International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'97)*, Las Vegas, 30 juin - 3 juillet 1997.
  - M. Juganaru, I. Sakho, C. Maurice, F. Montheillet. A Parallel Simulation of a Quantitative Large-Strain Polycrystal Deformation. In *Euro-Par'97 Conference*, Passau, 26 - 29 août 1997.

- **Conférences nationales avec comité de lecture**

  - M. Juganaru, I. Sakho. Régulation de charge dans les architectures MIMD à mémoire distribuée : un exercice d'équilibrage dynamique de données. In *Journées de Recherche sur le Placement Dynamique et la Répartition de Charge : Application aux Systèmes Répartis et Parallèles*, Paris, pages 35–38, 11-12 mai 1995.

- Mihaela Juganaru, Ibrahima Sakho, Laurent Carraro. Une analyse probabiliste de la complexité en temps d'un algorithme de régulation de charge. In *Ecole Française de Parallélisme, Réseaux et Systèmes. Placement Dynamique et Répartition de Charge : application aux systèmes parallèles et répartis*, pages 213–218, Presqu'île de Giens, 1–5 Juillet 1996.
- M. Juganaru, I. Sakho, L. Carraro. Une approche probabiliste d'estimation de la complexité en temps d'une stratégie de régulation de charge. In *Atelier d'Evaulation de Performances*, Versailles, 25 - 28 novembre 1996.

o **Rapports de recherche**

- Mihaela Juganaru, Ibrahima Sakho. A solution for regular data load balancing in distributed memory MIMD computers. Rapport de recherche, SIMADE, RR 95.11, Ecole Nationale Supérieure des Mines de St. Etienne, France, Novembre 1995.
- M. Juganaru, I. Sakho, L. Carraro. A time-complexity-analysis based load balancing algorithm for regular load. Rapport de recherche, SIMADE, RR 97.1, Ecole Nationale Supérieure des Mines de St. Etienne, France, Janvier 1997.

Table des figures

Liste des tableaux

Résumé : La résolution du problème d'allocation de charge représente un enjeu important dans l'exploitation des machines parallèles.

Nous faisons d'abord une étude bibliographique de ce problème dans le cadre des architectures à mémoire distribuée en mettant l'accent sur l'allocation dynamique, plus précisément sur l'équilibrage et la régulation de charges régulières.

Une stratégie originale de régulation basée sur un calcul de préfixe généralisé est proposée. Elle s'avère à la fois correcte, exacte et indépendante du réseau d'interconnexion de processeurs. Un noyau de régulation de charge basé sur cette stratégie est développé. Nous poursuivons ensuite avec une analyse de son temps total d'exécution. Nous trouvons qu'une loi de probabilité de Gumbel modélise le temps maximal d'exécution. A partir de ce résultat nous inférons des politiques d'initiation et de décision pour la mise en œuvre de la stratégie proposée. L'algorithme de régulation ainsi obtenu est donc efficace.

Une application de simulation des phénomènes mécaniques, déformation-recristallisation à chaud des agrégats polycristallins, est développée. Pour cette application dynamique nous utilisons le noyau de régulation de charge avec les politiques d'initiation et décision proposés. L'algorithme complet s'avère en pratique correct, stable et efficace.

Mots clé : Machine parallèle, Allocation de charge, Allocation dynamique, Algorithme de régulation, Calcul de préfixe, Loi de Gumbel, Déformation à chaud des polycristallins, Recristallisation dynamique continue.

Abstract : Solving the load allocation problem is an important issue for using and operating parallel machines.

We start with a state of arts of this problem for distributed memory architectures; we focus on dynamic allocation, more exactly on load balancing for regular load.

An original load balancing strategy based on a generalized prefix calculation is proposed. This strategy is correct, exact and independent on the processors interconnexion network. A load balancing kernel is implemented. We continue our study with an analysis of the execution time of this strategy. The main result is that a Gumbel distribution law modelizes the maximal execution time. Based on this results we infer a decision policy for initiating and running the load balancing process. So the proposed load balancing algorithm is efficient.

An application for mechanical process simulation (large-strain deformation and recrystallization) is developed. For this dynamic application we use our load balancing kernel and the initiation and decision policies. In practice the load balancing algorithm is always correct, stable and efficient.

Keywords : Parallel machine, Load allocation, Dynamic allocation,Load balancing algorithm, Prefix calculation, Gumbel law, Large-strain polycrystal deformation, Continuous dynamic recrystallization.