Contributions pour l'utilisation des machines parallèles, réparties et hétérogènes à travers le prisme de la gestion des données partagées

Loïc Cudennec

MINISTÈRE DES ARMÉES
Liberté
Égalité
Fraternité

DGA
DIRECTION GÉNÉRALE DE L'ARMEMENT

UNIVERSITÉ DE RENNES 1

Thèse d'Habilitation à Diriger les Recherches

présentée à

L'Université de Rennes 1

spécialité

Informatique

soutenue par

Loïc Cudennec

le 28 septembre 2022

Contributions pour l'utilisation des machines parallèles, réparties et hétérogènes à travers le prisme de la gestion des données partagées

Jury

Rapporteurs		
Camille	Coti	Professeure, Université du Québec à Montréal
Frédéric	Desprez	Directeur de recherche, INRIA
Raymond	Namyst	Professeur, Université de Bordeaux
Examinateurs		
Michaël	Krajecki	Professeur, Université de Reims Champagne-Ardennes
Vania	Marangozova-Martin	Maîtresse de Conférence, Université Grenoble Alpes
Christine	Morin	Professeure, Université de Rennes 1

Résumé

Ces deux dernières décennies ont été marquées par des évolutions technologiques majeures dans le domaine des composants électroniques et des moyens de calcul informatiques. Premièrement, la miniaturisation de la gravure des transistors a permis d'offrir des capacités de calcul toujours plus importantes pour un facteur de forme équivalent et une enveloppe thermique en baisse. Dès 2007, les objets connectés, et notamment les smartphones deviennent des biens de consommation courants, accélérant l'innovation d'usage dans le quotidien, l'industrie et la défense. Deuxièmement, les limites physiques de la miniaturisation poussent les constructeurs à proposer de nouvelles architectures de calcul, privilégiant le parallélisme massif ou la spécialisation des unités de traitement. En 2009 la course est lancée pour concevoir des processeurs embarquant des milliers de coeurs de calcul. Quelques années plus tard, des accélérateurs spécifiques aux tâches d'apprentissage machine sont intégrés dans les centres de calcul et les systèmes embarqués, pour traiter de grands volumes de données et déployer des tâches complexes sur des objets communicants.

Une conséquence directe liée à cette grande diversité d'architectures de calcul est la complexification du modèle de programmation. Désormais, il est courant de recourir à un langage, une interface ou un système d'exploitation spécifique à chaque architecture. Ainsi, un programme écrit pour un processeur classique (CPU) devra être ré-écrit pour exploiter efficacement une nouvelle architecture telle qu'un accélérateur graphique (GPU), un accélérateur reprogrammable (FPGA) ou un processeur massivement parallèle (many-coeur). Cette opération est bien souvent laissée à la charge du développeur. Si des initiatives de normalisation et d'unification de modèles de programmation existent, elles se heurtent rapidement à la problématique de la performance face à l'hétérogénéité grandissante des systèmes.

Cette thèse d'Habilitation à Diriger des Recherches revient sur une dizaine d'années de contributions théoriques et pratiques aux systèmes distribués, parallèles, hétérogènes et embarqués. Le fil rouge de ces travaux est la gestion des données partagées, ou comment rendre les accès transparents pour le développeur d'application tout en restant indépendant de l'architecture matérielle. Les domaines d'application sont les processeurs massivement parallèles, les serveurs de calcul hétérogènes et les nouvelles architectures dédiées aux applications d'intelligence artificielle. Un ensemble d'outils pour l'optimisation des systèmes et l'aide à la décision sont présentés permettant une gestion efficace des ressources tant sur le plan de l'intensité calculatoire que de la consommation d'énergie. Enfin, des perspectives de recherche sont proposées, ancrées dans la profonde mutation en besoins de puissance de calcul et de maîtrise de l'énergie requis par le développement de l'intelligence ambiante.

Remerciements

Je tiens à remercier chaleureusement Camille Coti, Frédéric Desprez, Michaël Krajecki, Vania Marangozova-Martin, Christine Morin et Raymond Namyst pour avoir accepté de participer à mon jury d'HDR. Merci à eux d'avoir pris le temps d'étudier ce manuscrit, malgré la petite taille de la police de caractère :)

Maintenir un cap et construire une activité de recherche cohérente n'est pas chose aisée. Pour cela il faut établir un subtil mélange constitué de persévérance individuelle et d'émulsion collective. Je remercie donc avec une grande gratitude mes amis et collègues du Commissariat à l'Énergie Atomique et aux Énergies Alternatives ainsi que de la Direction Générale de l'Armement pour toutes les idées que nous avons pu échanger et qui constituent le fondement d'un travail de recherche. Je remercie tout particulièrement Caaliph Andriamisaina, Pascal Aubry, Selma Azaiez, Sergiu Carpov, Cyril Faure, François Galéa, Thierry Goubier, Julien Hervé, Hana Krichen, Stéphane Louise, Oana Stan, Thibaud Tortech et Kods Trabelsi avec lesquels j'ai partagé une dizaine d'années d'activités foisonnantes, toujours passionnantes et synonymes de bonne humeur et de bienveillance !

Je remercie tout spécialement Safae Dahmani et Erwan Lenormand que j'ai encadrés en thèse et qui poursuivent de belles carrières dans le monde de la recherche. Les travaux présentés ici sont bien évidemment aussi le fruit de leur créativité ! Je souhaite autant de réussite à Cédric Prigent que je co-encadre avec Gabriel Antoniu et Alexandru Costan.

J'exprime ma grande reconnaissance à ceux qui ont largement favorisé ce parcours, en particulier Renaud Sirdey pour être venu dès ma soutenance de thèse en 2009 me proposer de travailler avec lui au CEA, Guy Gogniat et Henri-Pierre Charles pour les collaborations fructueuses lors de l'encadrement de thèses, Dominique Chauveau pour son accueil remarquable à la DGA et son soutien déterminant à la réalisation de cette HDR. Merci aussi à Marie-Aline Cavarroc, Arnaud Ramey et Eric Moliné pour leur accueil dans leurs unités respectives, et le soutien concret dont ils ont fait preuve. Je souhaite aussi remercier Karine Jones pour son aide précieuse dans la préparation de la soutenance.

Enfin, un tel projet déborde inévitablement dans la vie privée et je remercie ma famille qui a été d'un grand soutien et une grande source d'encouragements tout au long de ce chemin.

> A distributed system is one in which the failure of a computer you didn't even know existed can render your own computer unusable [Lamport, 1987].

Table des matières

Chapitre 1

Introduction

L'histoire de l'informatique est marquée par diverses vagues de systèmes centralisés, parallèles et distribués. Depuis les années 1950, plusieurs architectures se sont ainsi succédées ou ont cohabité : les terminaux connectés à un ordinateur central, les ordinateurs personnels mis en réseau, les grappes et grilles de calculateurs, le modèle centralisé de l'informatique dans les nuages ou encore l'aspect massivement décentralisé de l'internet des objets et de l'ubiquité numérique. Ces solutions architecturales sont dirigées par les applications, les usages ainsi que les avancées technologiques. En premier lieu, nous pouvons observer des avancées technologiques importantes et rapides depuis l'avènement de la microélectronique, bousculant le domaine d'application de l'informatique numérique. Principalement orientée vers le calcul scientifique à ses débuts, l'informatique est aujourd'hui ancrée dans le quotidien, supportée par des centres de données gigantesques et la myriade annoncée d'équipements embarqués exécutant des applications de communication, de mobilité et d'assistance basées sur l'intelligence artificielle. Dans ce contexte, il n'est pas étonnant de constater que les architectures de calcul ont évolué significativement pour répondre aux objectifs et contraintes du moment.

Ces évolutions concernent plusieurs points : un premier point est le recours au **parallélisme** massif dans les unités de traitement. Cette évolution, popularisée avec l'arrivée des processeurs multi-cœurs au début des années 2000 est devenue courante dans tous les domaines applicatifs, de l'accélérateur graphique dédié au calcul haute-performance au processeur basse-consommation équipant les *smartphones*. Un deuxième point est la **distribution** des applications sur plusieurs nœuds de calcul. La mutualisation de ressources distantes interconnectées par un réseau est une technique largement éprouvée depuis les premières expérimentations menées sur les réseaux d'entreprises et les instituts de recherche dans les années 1980 puis généralisée avec le déploiement d'internet et l'interconnexion des centres de calcul. Un troisième point est l'**hétérogénéité** des architectures de calcul qui consiste à agréger des unités de traitement suffisamment différentes pour nécessiter un effort de programmation spécifique. L'hétérogénéité est actuellement en plein essor, poussée par les besoins en accélération de certains noyaux de calcul liés à l'intelligence artificielle.

Ces trois propriétés, le parallélisme des tâches, la distribution des calculs et l'hétérogénéité des unités de traitement ont des implications fortes sur l'organisation des calculs et la gestion des données partagées. Si ces problématiques sont largement étudiées par la communauté scientifique depuis de nombreuses années, l'émergence d'architectures combinant les trois propriétés nécessitent de nouvelles approches et innovations. C'est notamment le cas dans le domaine de l'embarqué, par exemple pour l'automobile avec les véhicules autonomes ou encore les satellites avec le *New Space*. Ces systèmes connaissent ces dernières années une évolution majeure sur le plan de l'électronique et de l'informatique : à l'origine pensés de manière monolithique, c'est-à-dire avec une architecture de calcul mono-cœur éprouvée pour des raisons de sécurité et de déterminisme à l'exécution, ils sont désormais des exemples d'intégration de nouvelles technologies sur plusieurs plans. Sur le plan du parallélisme, avec le recours à des processeurs multi-cœurs basse-consommation, sur le plan de la distribution des calculs avec l'interconnexion par des réseaux embarqués ou des réseaux sur puce de différentes unités de traitement spécialisées, et enfin sur le plan de l'hétérogénéité avec l'intégration d'accélérateurs graphiques, de processeurs many-cœurs ou de processeurs à la logique programmable. Ainsi pouvons-nous trouver des accélérateurs Nvidia dans les voitures pour des tâches liées à la conduite autonome ou au divertissement, et des FPGA Xilinx dans les satellites pour du traitement du signal ou de la reconnaissance d'objets.

Ces nouveaux systèmes hétérogènes ont en commun de profiter d'un changement de méthode de conception en ayant recours à des composants sur étagère. Ces produits à l'origine conçus pour des équipements grand public tels que les ordinateurs de bureau ou les *smartphones*, offrent un coût d'intégration significativement plus faible qu'une architecture dédiée produite en petit volume. Ils fournissent de plus bien souvent un environnement de développement mature et continuellement amélioré par les retours des utilisateurs. Ceci facilite leur intégration dans des projets plus flexibles, par exemple menés en suivant les méthodes agiles. En contrepartie, la multiplicité des architectures proposées rend l'établissement de normes de programmation illusoire et l'intégration de plusieurs unités différentes dans un même système conduit rapidement à un casse-tête de génie logiciel.

Pourtant, il existe un enjeu réel à fournir des méthodes et outils permettant une exploitation efficace de ces nouvelles architectures. La première raison est la **performance calculatoire**, dirigée par le besoin d'effectuer des calculs au plus proche des utilisateurs, notamment pour gagner en réactivité et rationaliser les communications face au déluge

de données. Une deuxième raison est la nécessité de maîtriser la **consommation d'énergie** afin de gagner en autonomie pour les systèmes fonctionnant sur batterie, ou tout simplement pour concevoir des systèmes plus respectueux des normes environnementales. Et force est de constater que l'aspect énergétique est devenu aussi important que la puissance de calcul ces dix dernières années, comme en témoigne la création des listes du GREEN500, mettant en valeur les super-calculateurs les plus efficaces sur le plan du **compromis calculatoire et énergétique**.

Travaux présentés dans ce manuscrit. La problématique principale étudiée réside dans la capacité à fournir une abstraction logicielle simple permettant l'exploitation efficace d'une plateforme matérielle complexe. Le fil rouge de ce manuscrit est la recherche du compromis entre la performance calculatoire et la consommation d'énergie pour les architectures massivement parallèles, distribuées et hétérogènes. Les travaux présentés interviennent sur deux plans : le modèle de programmation et le modèle de gestion des données. Ces deux approches sont en réalité intrinsèquement liées : le modèle de programmation détermine souvent la manière dont les données sont accédées ou tout du moins en limite les possibilités. Plusieurs exemples concrets sont étudiés dans ce manuscrit : un modèle de programmation flot-de-données pour un processeur many-cœur, un outil de décision pour les protocoles de cohérence des caches coopératifs et une mémoire virtuellement partagée logicielle pour architectures hétérogènes distribuées.

La réalisation de ces contributions s'est déroulée de 2009 à 2019 au sein de l'institut LIST du Commissariat à l'Énergie Atomique et aux Énergies Alternatives (le CEA) dans un contexte à cheval entre les activités d'un laboratoire de recherche académique et un département de R&D industriel. L'une des missions du CEA est d'apporter un soutien à l'industrie à travers le transfert de briques technologiques innovantes. Les thématiques traitées sont donc largement dirigée par les besoins des industriels, la part du financement étatique du CEA LIST étant très faible. Dès lors, conduire une activité de recherche consolidée sur plusieurs années n'est pas la norme et demande des efforts de justification spécifiques à l'institut. Pour autant, l'histoire du CEA et la variété de ses missions dans tous les domaines de la science conduisent à une multidisciplinarité singulière qui est une incroyable force pour imaginer de riches collaborations transverses.

Les contributions suivantes sont organisées en deux chapitres. Le chapitre 2 présente les travaux menés entre 2009 et 2015 pour l'exploitation des architectures many-cœurs. Le chapitre 3 présente l'ensemble des activités menées entre 2015 et 2020 autour de la conception d'une mémoire partagée logicielle pour les architectures hétérogènes. Quelques contributions annexes concernant le déploiement d'applications sur des chaînes de production industrielles ne sont pas présentées ici.

Programmabilité des processeurs many-cœurs. En 2009 est créé le laboratoire commun CEA-Kalray dont l'objectif est de proposer un environnement de programmation pour la puce many-cœur MPPA-256 et d'influencer l'architecture matérielle. Nous proposons un modèle de programmation flot de données basé sur une variation du modèle CSDF (*Cyclo-Static Data-Flow*) dans lequel des agents communiquent à travers des ports et des liens de communication, suivant une spécification des échanges fournie par l'application. Cette particularité permet d'effectuer de nombreuses vérifications lors de la compilation, notamment l'absence d'interblocages, un cadencement correcte des tâches ou encore un dimensionnement possible des tampons de communication. Le langage de programmation associé, Σ-C, est supporté par une chaîne de compilation expérimentale qui a été transférée et industrialisée dans l'environnement de développement fourni par Kalray. Pour réaliser cette chaîne de compilation, de nombreux travaux ont été engagés par une équipe d'une dizaine de personnes pendant 3 ans. Dans ce manuscrit, après un état de l'art sur les many-cœurs et une présentation succincte du langage Σ-C, nous décrivons plus précisément deux étapes de la chaîne de compilation : 1) la compilation du parallélisme qui consiste à instancier, vérifier et optimiser le graphe de communication flot-de-données et 2) la réorganisation des accès aux données qui consiste à gérer les tampons partagés implémentant les canaux de communication entre agents. Enfin, pour conclure sur l'utilisation de la chaîne de compilation Σ-C, une troisième contribution est décrite traitant de l'exécution de programmes chimiques.

Un deuxième axe de travail visant à faciliter l'exploitation des many-cœurs porte sur la cohérence des caches. Les processeurs dotés d'un grand nombre de cœurs (plus d'une centaine) n'offrent pas de système matériel de cohérence des caches, ce qui empêche le déploiement d'applications écrites en mémoire partagée à l'échelle de la puce et contraint le développeur à exprimer à la main les transferts de données dans le code. Dans le manuscrit nous présentons des travaux permettant de déployer des protocoles de cohérence adaptés à ces grilles de calculateurs miniaturisées. Pour cela, plusieurs contributions sont présentées : 1) trois protocoles de cohérence des caches reposant sur l'aspect coopératif des cœurs voisins, 2) deux outils pour l'évaluation des protocoles de cohérence des caches, l'un basé sur une méthode analytique et le second sur un simulateur de réseaux sur puce et 3) un outil d'aide à la décision pour le choix et la configuration des protocoles de cohérence basé sur une méthode populationnelle.

Programmabilité des architectures hétérogènes. Entre 2014 et 2018 les projets européens H2020 FiPS et M2DC dans lesquels le CEA LIST est impliqué visent à concevoir une architecture micro-serveur, c'est-à-dire un serveur de calcul modulaire dont le matériel peut être configuré avec des processeurs et des accélérateurs hétérogènes. Les architectures ainsi conçues, commercialisées par Christmann en Allemagne, reflètent la tendance des systèmes intégrés actuels, à la frontière des centres de calcul et des systèmes embarqués. Les micro-serveurs ne disposent pas d'une mé-

moire centrale : les extensions supportant les unités de traitement sont interconnectées par un réseau intégré de type Ethernet. Les travaux présentés dans ce manuscrit concernent la conception et le déploiement d'une mémoire virtuellement partagée logicielle sur les micro-serveurs afin d'en abstraire la complexité matérielle pour le placement des tâches sur les ressources et la gestion des données partagées entre les nœuds.

Dans ce manuscrit, les travaux autour de la mémoire partagée sont présentés en deux parties : une première partie concerne la conception et l'efficacité du système. Elle comprend la présentation d'un modèle de programmation hybride inspiré de l'approche événementielle ainsi qu'une comparaison des performances d'une application écrite sur la mémoire partagée avec deux intergiciels populaires de passage de message, MPI et ZeroMQ. Une deuxième partie traite de la dépense d'énergie et des méthodes pour la maîtrise de la consommation sur une architecture hétérogène. Cette partie présente plusieurs contributions : 1) un outil d'aide à la décision pour le choix d'une topologie logique et le dimensionnement de la mémoire partagée basé sur des méthodes d'exploration de solutions par voisinage et menant à un compromis entre performance calculatoire et dépense énergétique, 2) la réduction de la consommation d'énergie dans les communications MPI avec une application à la mémoire partagée et 3) l'intégration d'accélérateurs matériels reprogrammables dans la mémoire partagée avec une application dans le cadre du projet H2020 LEXIS pour la mise en place de systèmes d'alerte aux tsunamis.

□

Chapitre 2

Gestion des ressources à l'échelle des processeurs many-cœurs

Les processeurs many-cœurs, aussi appelés MPPA (*Multi Purpose Processor Array*), sont composés de plusieurs dizaines à plusieurs milliers de cœurs de calcul réunis sur une puce et interconnectés par un réseau sur puce NoC (*Network on Chip*). À l'instar des processeurs plus classiques les many-cœurs sont des processeurs dont les cœurs peuvent effectuer des opérations génériques et ce de manière indépendante. Ceci les différencie par exemple des accélérateurs de type GPGPU (*General Purpose computing on Graphics Processing Unit*) pour lesquels les calculs sont orchestrés par une machine hôte et dont le flot de contrôle d'une unité de calcul est possiblement contraint par celui des unités voisines, notamment pour les instructions conditionnelles et de branchement. Les many-cœurs sont caractérisés par des capacités de calculs importantes dès lors que le parallélisme inhérent à l'architecture est efficacement exploité. De part leur conception basée sur l'accumulation d'éléments simples de calcul, de stockage et de communication, les many-cœurs offrent un excellent rapport entre la puissance de calcul et l'énergie consommée. Ces caractéristiques répondent à des contraintes liées aux contextes du HPC (*High-Performance Computing*) et des systèmes embarqués, permettant de nouvelles applications et le développement d'un contexte émergent, le HPEC (*High-Performance Embedded Computing*).

Le renouveau des puces massivement parallèles. Au milieu des années 2000, le monde de l'embarqué connaît un nouvel essor avec le développement de processeurs basse-consommation. Ces processeurs présentent un jeu d'instruction simplifié et des fréquences de fonctionnement plus faibles, en dessous de 700 MHz, là où les processeurs plus traditionnels font la course au delà des 3 GHz. En 2004, le processeur ARM1176JZF-S mono-cœur [arm Limited, 2004] doté du jeu d'instructions 32 bits ARMv6 propose une puissance de calcul suffisante pour exécuter un système d'exploitation complet, pour une consommation énergétique instantanée d'environ 1 W [Astudillo-Salinas et al., 2016]. Ce processeur équipe le premier Iphone 2G en 2007 [Patterson, 2011] et le premier Raspberry Pi en 2012 [Raspberry PI Foundation, 2017]. Ces deux équipements ont alors une influence majeure sur l'électronique grand public dans les télécommunications mobiles et dans le développement et le prototypage de systèmes embarqués, en robotique notamment. Pour répondre aux problématiques de l'embarqué, la tendance est alors de multiplier le nombre de cœurs basse-consommation ou de proposer des cœurs de calcul simplifiés. Par exemple en se basant sur des processeurs de générations plus anciennes telles que le premier Pentium P54C conçu au milieu des années 90 et dont le circuit est à la base des cœurs de calcul du Xeon Phi d'Intel [Barlas, 2015] en 2010. Ces circuits sont miniaturisés grâce à l'utilisation de technologies de gravure plus fines, ce qui permet de colocaliser un nombre important de cœurs sur une même puce et de réduire la puissance nécessaire au fonctionnement du processeur afin de réaliser des économies d'énergie. La puissance de calcul repose alors sur la multiplication des cœurs de calcul embarqués sur la puce.

L'approche n'est pas nouvelle : au début des années 1980 la capacité d'implantation des transistors sur une puce atteint une limite, freinant la conception d'architectures plus complexes. Le *Transputer* [Wikipedia, 2021] développé par la société Inmos au Royaume-Uni répond à cette problématique. Le principe est de se baser sur un processeur générique capable d'être directement interconnecté à d'autres processeurs sur une carte. Les Transputers sont intégrés dans une matrice pour former une architecture parallèle relativement intégrée pour l'époque. Cette organisation nécessite l'emploi d'un paradigme de programmation parallèle basé sur CSP (*Communicating Sequential Processes*) implémenté avec le langage *occam*, ce qui implique potentiellement la réécriture ou le portage des applications et qui n'est pas sans conséquences pour l'adhésion à un tel écosystème. L'évolution technologique dans le domaine de la microélectronique rend de nouveau possible la course à l'intégration des composants et limite l'intérêt pour le Transputer. Cependant, de nombreuses architectures bénéficieront des éléments de conception du Transputer, tels que l'accumulation d'unités de calcul, le réseau intégré et la gestion du multitâche au niveau matériel et logiciel.

Entre les années 1980 et 2010, les architectures parallèles et les systèmes répartis ont bénéficié de plusieurs engouements : les *clusters Beowolf*, des grappes de calculateurs fédérant des ordinateurs de bureau inutilisés dans les années 80, les grappes de calcul HPC dans les années 90, les grilles de calculateurs telles que définies par Foster [Foster and

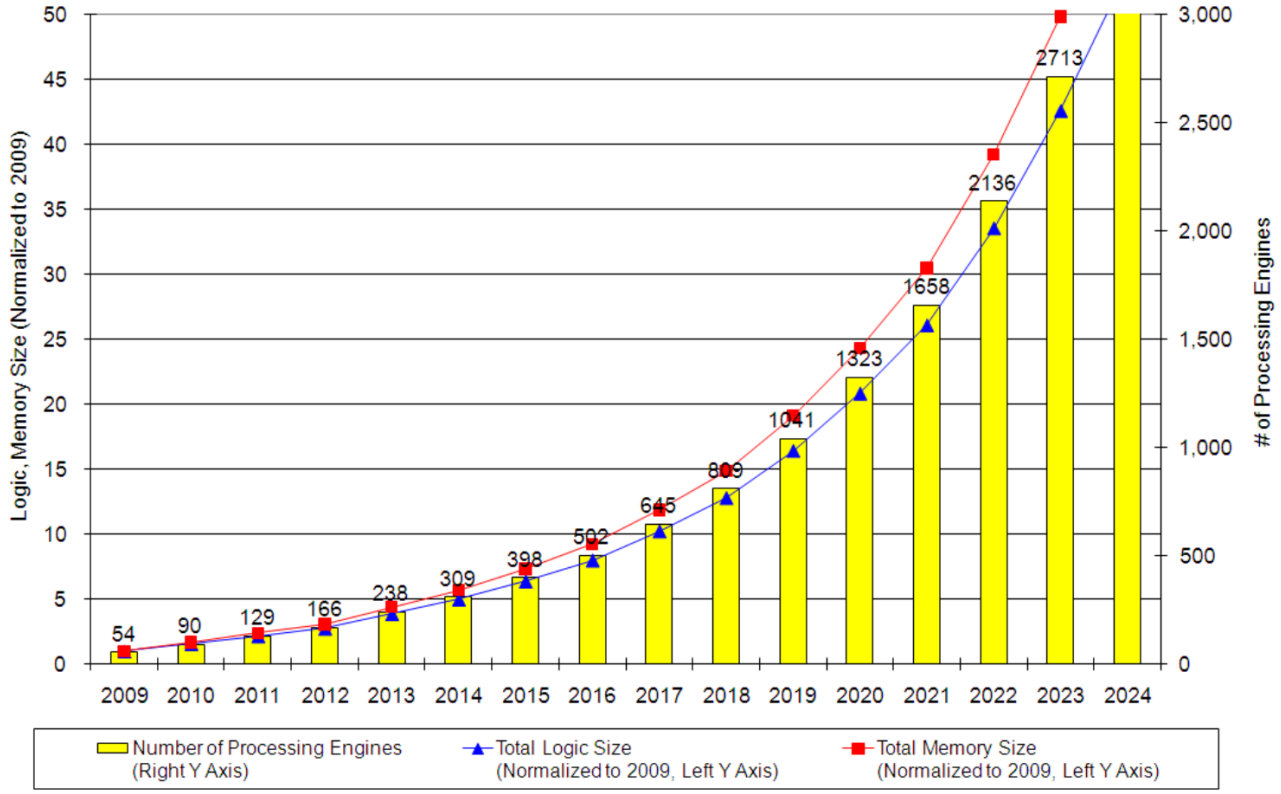


FIGURE 2.1 – Prévisions pour l'évolution de la complexité des composants dans les systèmes embarqués grand public (*SOC Consumer Portable Design Complexity Trends* [ITRS, 2011] proposé par l'*International Technology Working Group* en 2010 (mis à jour en 2011). Figure issue du rapport.

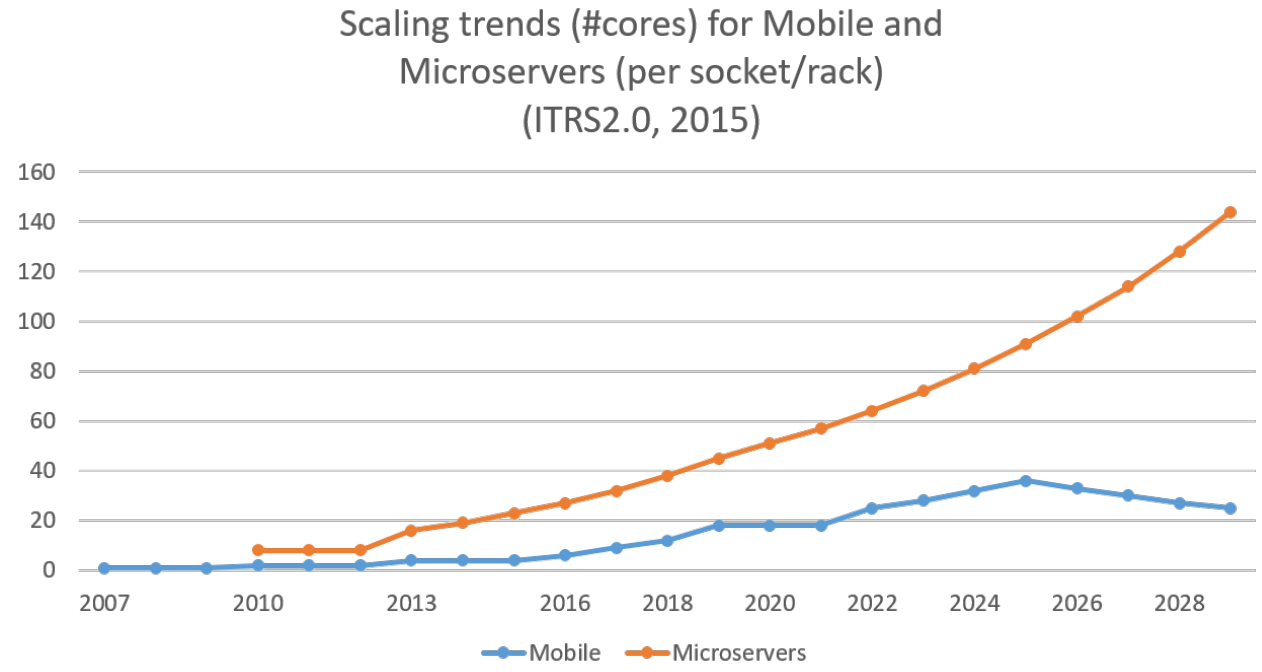


FIGURE 2.2 – Prévisions pour l'évolution du nombre d'unités de calcul pour les équipements embarqués et les microserveurs (*SOC Consumer Portable Design Complexity Trends* [ITRS2.0, 2015] proposé par l'*International Technology Working Group* en 2015. Données compilées à partir du rapport.

Kesselman, 2004] dans les années 2000, l'émergence des processeurs multi-cœurs au milieu des années 2000 et les GPU (*Graphics Processing Unit*) appliqués au rendu graphique et au calcul numérique (GPGPU).

À la fin des années 2000, alors que la conception des processeurs génériques se heurte aux limites atteintes par l'augmentation de la fréquence d'horloge et la dissipation thermique, les processeurs many-cœurs apparaissent comme une architecture naturelle, faisant une nouvelle fois appel au parallélisme massif pour pallier les problèmes rencontrés par l'approche plus centralisée. Les prévisions émises par l'*International Technology Working Group* en 2010 dont est issue la figure 2.1 anticipent une croissance géométrique du nombre de cœurs de calcul dans les puces embarquées : de quelques dizaines de cœurs en 2010 à plus d'un millier de cœurs en 2020. Si ces prévisions s'avèrent aujourd'hui largement surestimées dans le domaine de l'électronique grand public, elles témoignent d'un engouement important pour cette dynamique de parallélisation massive des processeurs. Une version plus tempérée de ce rapport est proposée en 2015 (les résultats sont compilés dans la figure 2.2), dans lequel les many-cœurs ne sont plus pressentis pour équiper les systèmes embarqués grand public, mais plutôt les architectures de type micro-serveur hétérogènes, comme développé dans le Chapitre 3. Le nombre de cœurs reste cependant modeste, ne dépassant pas la centaine à l'horizon 2025. Les many-cœurs de plus grande taille ne disparaissent pas pour autant et se retrouvent dans des domaines spécifiques : les super-calculateurs et les systèmes embarqués autonomes.

Panorama des many-cœurs. La table 2.1 présente plusieurs processeurs many-cœurs conçus depuis le milieu des années 2000. Cette table a été construite en collectant les informations librement accessibles, publiées dans des conférences et journaux, sous forme de fiches techniques ou de sites collaboratifs. Elle permet de dégager plusieurs tendances concernant les choix de conception des many-cœurs sur les critères de taille de circuit, de puissance et de consommation, ainsi que de présenter les principaux acteurs de cet écosystème technologique. Une première observation concerne le nombre de cœurs : pendant la première décennie la stratégie consiste à explorer des architectures comportant un grand nombre de PE (*Processing Element*) (les cœurs de calcul). De quelques dizaines de cœurs pour les premières architectures proposées par Tilera et Intel en 2007, à plus de 2000 cœurs pour le processeur conçu par PEZY Computing en 2017. Il semble que la course au nombre de PE marque ensuite une pause : les constructeurs se concentrent sur les unités d'accélération spécialisées pour certains noyaux de calcul rencontrés dans les applications en IA (Intelligence Artificielle), tels que les convolutions et la rétropropagation de gradients pour l'apprentissage automatique (ML (*Machine Learning*)). Par exemple l'évolution du processeur MPPA de Kalray montre une diminution importante du nombre de cœurs entre Bostan en 2015 (256 cœurs) et Coolidge en 2020 (80 cœurs, mais auxquels viennent s'adosser des accélérateurs vectoriels pour le calcul sur les *tenseurs* utilisés en ML).

Une deuxième observation concerne les objectifs en terme de performance par unité d'énergie consommée, ce qui détermine si le processeur peut être intégré dans un système embarqué, contraint par la dépense énergétique, ou si il est destiné à être utilisé dans un centre de calcul. Les deux tendances sont représentées, avec d'une part les puces consommant entre 100 et 400 W conçues pour le contexte HPC et d'une autre part les puces consommant quelques watts à quelques dizaines de watts susceptibles d'être embarquées dans des équipements tels que les véhicules autonomes, les drones ou les satellites. Par exemple la puce Epiphany IV de la *fabless* Adapteva est constituée de 64 PE pour une consommation instantanée de seulement 2 W.

Enfin, une troisième observation porte sur le choix d'offrir ou non un mécanisme de cohérence matérielle entre les mémoires des PE (les caches, les scratchpads), indiqué dans la colonne CC. La présence de cohérence matérielle détermine si il est possible de déployer des applications reposant sur le paradigme de programmation en mémoire partagée, comme il est possible de le faire avec les *thread POSIX* sur un processeur ou sur un multiprocesseur NUMA (*Non-Uniform Memory Architecture*). Les premiers many-cœurs offrent généralement des mécanismes de cohérence des caches, reposant sur des NoC dédiés : par exemple une superposition de 6 réseaux de type *mesh* (en grille) pour interconnecter les cœurs dans l'architecture du TILE64, dont 2 réseaux sont réservés à la gestion de la cohérence des caches. Un autre exemple est le réseau sur puce en anneau présent dans les Xeon Phi, basé sur la circulation d'un *token* (jeton). La topologie de ces réseaux est une propriété importante à prendre en compte pour exploiter correctement les cœurs de calcul, avec pour objectif de limiter le nombre de *hops* entre la source et la destination et éviter le phénomène de contention sur des routeurs. Avec la multiplication du nombre de cœurs au milieu des années 2010, les mécanismes de CC sont rarement proposés. La CC est réintégrée dans les puces modernes dès lors que le nombre de cœurs ne dépasse pas la centaine, une limite pour le passage à l'échelle des mémoires virtuellement partagées déjà évoquée 35 ans auparavant par [Li, 1988] avec les grappes de calculateurs. Ce choix technique est aussi dirigé par les besoins d'abstraction du matériel pour les applications et intergiciels dédiés à l'intelligence artificielle, jusqu'alors développés pour les architectures parallèles plutôt que réparties.

Stratégies pour passer à l'échelle. La course au nombre de cœurs embarqués s'accompagne de problématiques classiques de passage à l'échelle concernant les entrées-sorties du cœur de calcul, le réseau sur puce, la mémoire locale au cœur, la mémoire centrale à la puce, l'alimentation électrique et même la synchronisation du front d'horloge [Teixeira Monteiro, 2016] ! Dans ces conditions, les plus gros many-cœurs proposent rarement un mécanisme matériel pour la cohérence de cache et reposent sur deux principales techniques : 1) une architecture hiérarchique composée d'un

Année	Puce	Concepteur Référence	Cœurs	MHz	Perf.	W	CC	Nat.
2007		Intel Corp. [Vangal et al., 2008]	80	4270	1000 SP	97	✓	
2007	TILE64	Tilera Corp. [Bell et al., 2008]	64	750	144 GOPS	10,8	✓	
2010	MIC Knights Ferry	Intel Corp. [Wikipedia, 2020]	32	1200	750 SP	300	✓	
2011		Intel Corp. [Howard et al., 2011]	48	1000		125	✓	
2012	Xeon Phi Knights Corner	Intel Corp. [Wikipedia, 2020]	61	1200	1200 DP	300	✓	
2012	PEZY-1	PEZY Computing [WikiChip, 2020]	512	533	533 SP 266 DP	35		
2013	TILE-Gx72	Tilera Corp. [Matthew Mattina, 2013]	72	1200		75	✓	
2013	MPPA Andey	Kalray Inc. [de Dinechin et al., 2013a]	256	400	211 SP 70 DP	12		
2013	Epiphany-IV E64G401	Adapteva Inc. [Adapteva Inc., 2017]	64	800	102 SP	2		
2014	PEZY-SC	PEZY Computing [PEZY Computing, 2020]	1024	733	3000 SP 1500 DP	100		
2015	MPPA-2 Bostan	Kalray Inc. [de Dinechin, 2015]	256	800	845 SP 422 DP	20		
2016	Xeon Phi Knights Landing	Intel Corp. [Wikipedia, 2020]	72	1500	5300 SP 2700 DP	260	✓	
2016	Sunway SW26010	ICC (Shanghai) [Dongarra, 2016]	260	1450	3062 DP	374		
2016	Epiphany-V	Adapteva Inc. [Richie and Ross, 2018]	(*1024*)	(*500*)		(*10*)		
2017	PEZY-SC2	PEZY Computing [WikiChip, 2020]	2048	1000	8192 SP 4096 DP	200		
2017	KiloCore	U.C. Davis [Bohnenstiehl et al., 2017]	1000	115	1,78 TIPS	40		
2017	NEO	REX Computing Inc. [Sebexen and Sohmers, 2015]	(*256*)		(*512 DP*)	(*8*)		
2017	Matrix-2000	NUDT [WikiChip, 2017]	128	1200	4920 SP 2460 DP	240	✓	
2017	Xeon Phi Knights Mill	Intel Corp. [Domke et al., 2019]	72	1500	13800 SP 1700 DP	320	✓	
2019	EPYC 7742	AMD Inc. [AMD Inc., 2019]	64	3400	(*1360 DP*)	225	✓	
2020	INTACT TSARLET	CEA-Leti [Vivet et al., 2020]	96	1150	220 GOPS	(*23*)	✓	
2020	MPPA-3 Coolidge	Kalray Inc. [Serra, 2020]	80	1200	1150 SP 384 DP	35	✓	
2020	Prodigy T16128	Tachyum Inc. [Tachyum, 2020]	(*128*)	(*4000*)	(*16000 DP*)		✓	
2020	MN-Core	Preferred Networks [Preferred Networks, 2020]	512		(*131000 SP*) (*32800 DP*)	(*500*)		
TBA	PEZY-SC3	PEZY Computing [WikiChip, 2020]	(*8192*)	(*1333*)		(*400*)		

TABLE 2.1 – Quelques architectures many-cœurs conçues depuis le début des années 2000. La performance est donnée telle qu'indiquée par les concepteurs ou dans des évaluations académiques. Les unités de mesure pour la performance sont le SP pour GFLOPS (nombre d'opérations en virgule flottante par seconde, en million 10^6) Simple Précision, le DP pour GFLOPS Double Précision, le GOPS (nombre d'opérations par seconde, en million 10^6) et le TIPS pour un million de MIPS (nombre d'instructions par seconde, en million 10^6) (10^{12} IPS (nombre d'instructions par seconde)). La colonne CC (Cohérence de Cache) indique si le processeur est doté d'un système de maintient de la cohérence des caches ou des *scratchpads* entre les cœurs. Les valeurs entre parenthèses sont à prendre avec précaution.

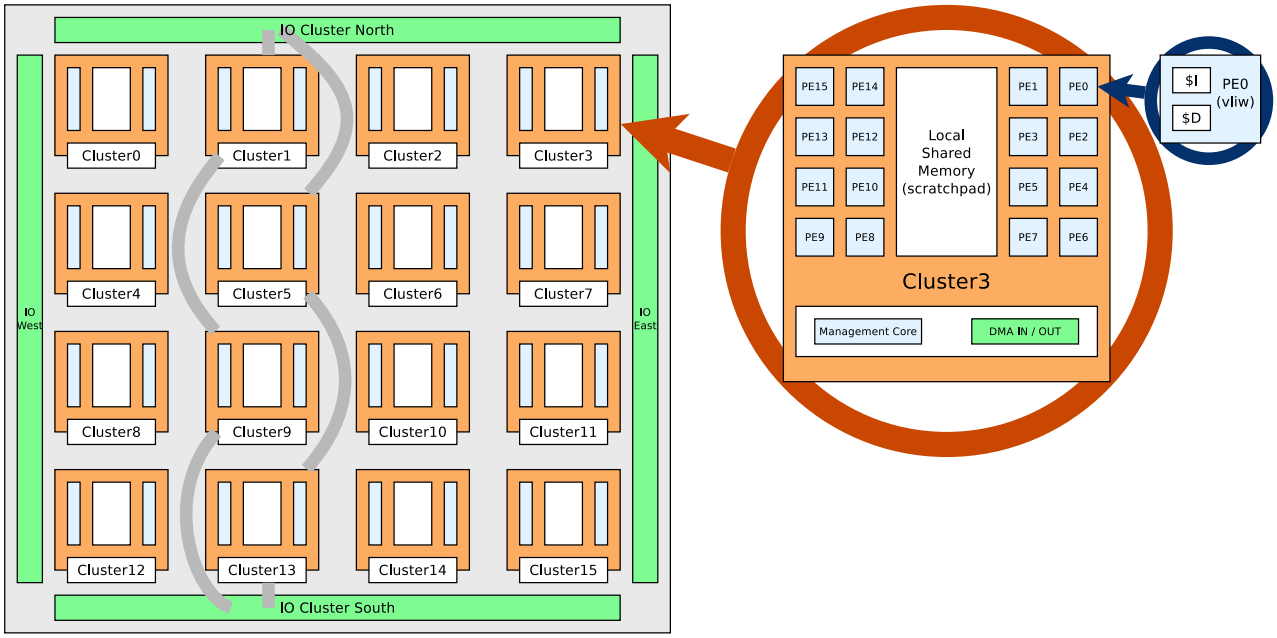


FIGURE 2.3 – Architecture de la puce Kalray MPPA Andey : une organisation hiérarchique composée de 16 clusters de 16 PE interconnectés par un NoC dont la topologie est un tore 2D entrelacé (pour des raisons de lisibilité le NoC est représenté pour une colonne Nord-Sud, le maillage complet concerne toutes les colonnes et toutes les lignes Est-Ouest).

ensemble de *clusters* de PE. Par exemple la puce MPPA Andey de Kalray propose 256 cœurs organisés en 16 *clusters* de 16 cœurs (Figure 2.3). Dans chaque *cluster*, une mémoire physiquement partagée (*scratchpad*) permet aux processus s'exécutant sur les cœurs du *cluster* d'allouer de la mémoire et de communiquer par le moyen de cet espace partagé. En revanche, deux cœurs localisés dans des *clusters* différents doivent faire appel à une bibliothèque de MP (Passage de Message) spécifique pour communiquer. 2) Une deuxième technique pour le passage à l'échelle consiste à simplifier l'architecture globale, comme c'est le cas pour les puces Epiphany d'Adapteva, organisées en un tuilage de PE, chaque tuile comportant un cœur de calcul, une petite mémoire locale et des connexions aux tuiles voisines, ce qui n'est pas sans rappeler le Transputer précédemment évoqué. L'évolution des architectures montre que la cohérence de cache matérielle est souvent présente dans les nouvelles puces, ce qui peut s'expliquer par une demande dirigée par l'applicatif, un nombre de cœurs *raisonnable* (une centaine) et l'arrivée de nouvelles technologies pour l'intégration des NoC telles que le *3D stacking* (l'empilement sur la puce des couches contenant les cœurs, les mémoires et le NoC favorisant les échanges rapides verticaux en plus des communications horizontales classiques). Ces choix architecturaux ont des implications fortes sur les modèles de programmation comme nous le verrons dans la suite de ce document.

Le many-cœur d'aujourd'hui est-il le multi-cœur de demain ? Le TOP500 [TOP500, 2020] propose le classement biannuel des super-calculateurs les plus puissants selon les performances mesurées par le *benchmark* LINPACK [Dongarra, 1987]. Un second classement, le GREEN500, permet de prendre en compte la consommation d'énergie requise pour exécuter le *benchmark* et ainsi encourager la conception de machines offrant un meilleur compromis entre la performance calculatoire et la dépense énergétique. Depuis les années 2010, les machines classées dans le TOP500 présentent deux types de nœud de calcul : 1) un nœud composé d'un processeur multi-cœur et d'un GPGPU (une grande majorité des machines) et 2) un nœud composé d'un many-cœur ou composé d'un multi-cœur et d'un many-cœur. Les figures 2.4 et 2.5 permettent de mettre en valeur la place des many-cœurs dans les deux classements respectifs. Ces figures représentent le nombre maximal de cœurs présents dans un processeur (processeur hôte ou accélérateur) pour les dix premiers systèmes des classements respectifs. Elles ont été créées en analysant les listes mises à disposition sur le site du TOP500. La figure 2.4 montre l'introduction des premiers many-cœurs dans le TOP500 en novembre 2012 avec le système Stampede (U.Texas) classé 7^{e} et composé des premiers Intel Xeon Phi SE10P à 60 cœurs. Par la suite, plusieurs systèmes intègrent des many-cœurs, souvent développés spécifiquement pour les super-calculateurs (PEZY, Sunway, Matrix-2000), et aussi parfois pour des activités liées à la défense. Cette figure montre aussi l'évolution du nombre de cœurs dans les processeurs hôtes : de 2 à 4 cœurs tout au plus en 2009, exception faite du processeur IBM PowerXCell 8i à 9 cœurs qui équipe le premier super-calculateur du TOP500 au *Los Alamos National Laboratory*, à une vingtaine de cœurs en 2020 dans le top 10. Cette augmentation atteint en 2021 jusqu'à 48 cœurs pour le processeur Arm/Fujitsu A64FX [Okazaki et al., 2020] et 64 cœurs pour le processeur AMD EPYC 7742 qui équipent respectivement les super-calculateurs Fugaku du RIKEN (première place) et Selene de Nvidia (5^{e} place).

FIGURE 2.4 – Nombre de cœurs embarqués par le processeur généraliste (multi-cœur ou many-cœur) le plus important présent sur les nœuds de calcul des systèmes classés dans les 10 premières places du TOP500, de juin 2009 à juin 2022.

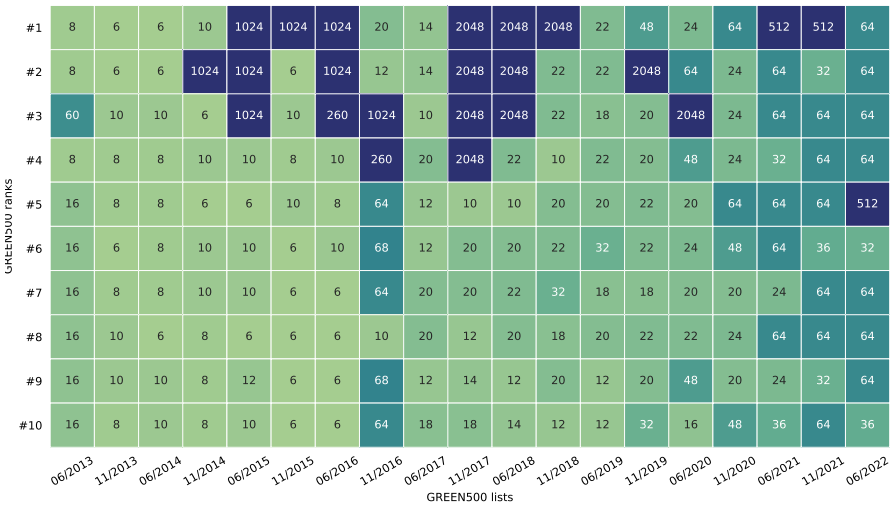


FIGURE 2.5 – Nombre de cœurs embarqués par le processeur généraliste (multi-cœur ou many-cœur) le plus important présent sur les nœuds de calcul des systèmes classés dans les 10 premières places du GREEN500, de juin 2013 à juin 2022.

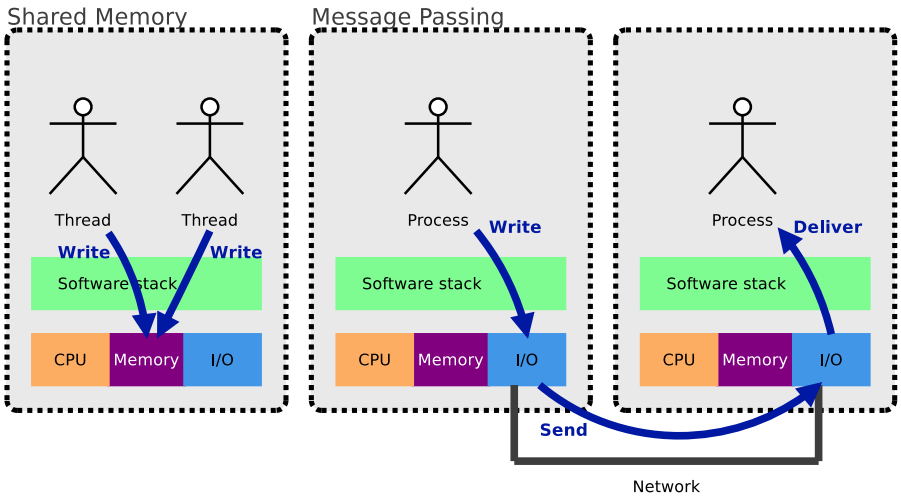


FIGURE 2.6 – Modèles de programmation en mémoire partagée (gauche) et par passage de messages (droite).

L'évolution des architectures de calcul dans le HPC montre une convergence entre les processeurs multi-cœurs haute-performance "classiques" et les processeurs many-cœurs.

Cette convergence est d'autant plus prononcée que les processeurs traditionnels se heurtent depuis les années 2010 à la fin de la loi de Dennard [Dennard et al., 1974]. Cette loi énonce que la miniaturisation des transistors permet de multiplier leur nombre sur une puce et d'augmenter la puissance de calcul tout en conservant la même consommation électrique. Alors que les procédés de fabrication des circuits descendent sous les 3 nm (3×10^{-9} m, en comparaison du diamètre estimé d'un atome d'hélium qui est de $5{,}96 \times 10^{-10}$ m), les courants de fuite induits par les petites échelles ne permettent plus de dissiper correctement la chaleur et de conserver des fréquences de fonctionnement élevées. L'histoire se répète : les moyens de calculs deviennent massivement parallèles et les processeurs héritent des travaux effectués pour les many-cœurs. Le processeur many-cœur emblématique Intel Xeon Phi est en fin de production [Cutress and Shilov, 2019] alors que les processeurs multi-cœurs AMD EPYC proposent désormais autant de cœurs de calcul que les many-cœurs de l'époque. Dès lors, les processeurs généralistes rencontrent les mêmes problématiques de programmabilité que les puces massivement parallèles.

Problématique de la programmabilité. La complexité architecturale des many-cœurs provient de l'accumulation et de la hiérarchisation des briques de calcul et de communication. Concevoir un programme nécessite bien souvent une connaissance minimale de l'architecture cible, ce qui contraint ou détermine le choix du modèle de programmation. Ainsi, il est possible de monter en complexité, du programme séquentiel, au programme parallèle, au programme réparti jusqu'aux approches hybrides mêlant les différents paradigmes de programmation.

La figure 2.6 illustre deux principaux modèles de programmation : la mémoire partagée à gauche, dans lequel un espace est alloué dans une mémoire physiquement partagée par des tâches (dans cet exemple, deux fils d'exécution, des *threads*). Une donnée partagée est identifiée par une adresse dans l'espace logique, traduite par un emplacement dans la mémoire physique. Les accès concurrents sont alors arbitrés localement par le matériel et/ou la pile logicielle. Cette méthode permet d'écrire des programmes parallèles avec une représentation des données partagées intuitive. Les tâches sont déployées sur une machine dite NUMA, sur laquelle la proximité physique des unités de calcul et des mémoires permettent des temps d'accès faibles (entre la nanoseconde 10^{-9} s et la microseconde 10^{-6} s). Cependant, les machines NUMA sont aujourd'hui limitées à quelques dizaines ou centaines de cœurs, avec un TCO (*Total Cost of Ownership*) important lorsque le nombre de cœurs est tel qu'il nécessite des architectures dédiées, bien loin des standards implémentés pour les machines grand public.

Le paradigme de programmation par passage de message illustré à droite de la figure 2.6 peut être utilisé en l'absence d'une mémoire physiquement partagée : c'est-à-dire lorsqu'il n'est pas possible d'allouer un espace logique visible et accessible par les deux tâches. Une donnée partagée est identifiée par un ou plusieurs messages qui sont transmis entre plusieurs tâches. Ces messages peuvent être transmis en empruntant des liens de communication potentiellement complexes en fonction de la topologie réseau tissée entre les deux processus, avec des temps d'accès relativement élevés, entre la microseconde et la milliseconde. Ils peuvent aussi être tout simplement transmis en utilisant un tampon partagé si les deux processus sont localisés sur une machine disposant d'une mémoire physiquement partagée, comme nous le verrons dans la section 2.1.3. Le passage de message permet d'écrire des programmes répartis, faisant fi de l'architecture sous-jacente, ce qui permet d'exploiter des grappes de calculateurs dont le coût unitaire est beaucoup moins élevé qu'une grosse machine NUMA et de passer plus facilement à l'échelle pour le nombre d'unités de traitement. Cependant, le partage de données est plus complexe à gérer dans la partie applicative car il repose sur un mécanisme différent, impliquant de traduire un accès en une suite de messages. Les accès aux données partagées reposent par

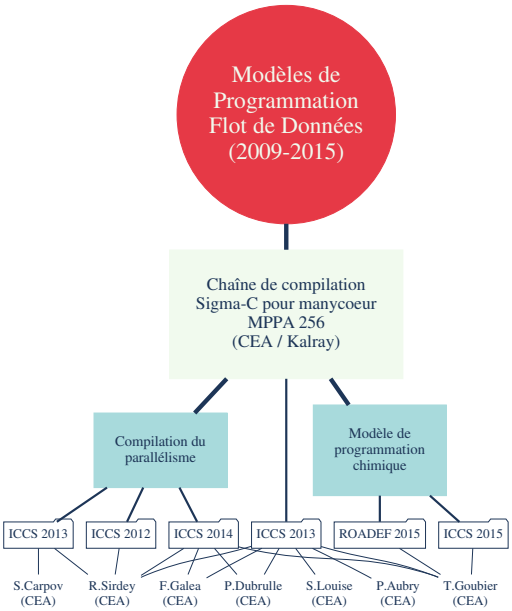
ailleurs sur des fondements théoriques différents : sur une machine NUMA il est possible d'utiliser une horloge globale pour séquentialiser les accès en se basant sur un horodatage simple. Ce n'est pas le cas dans un système distribué pour lequel il n'existe pas une telle horloge globale, mais plutôt une multitude d'horloges locales non synchronisées. Dès lors, l'arbitrage sur la séquentialisation des accès repose sur des dépendances causales et l'utilisation d'horloges vectorielles telles que décrites par Lamport [Lamport, 1978], Mattern [Mattern, 1988] et Raynal [Raynal, 1987].

Dans le contexte des many-cœurs, le paradigme de programmation en mémoire partagée facilite le portage d'applications existantes, le développement et la mise au point de nouvelles applications, favorisant l'adoption du processeur et de son environnement de développement (IDE (*Integrated Development Environment*)) par la communauté scientifique académique et industrielle. Dans le cas contraire, en l'absence de mémoire partagée, le développement des applications peut s'effectuer selon trois méthodes : 1) reposer sur le paradigme de programmation par passage de messages, soit en utilisant des intergiciels classiques tels que MPI et ZeroMQ, soit en utilisant un intergiciel propriétaire de MP fourni par le constructeur de la puce. Cette solution implique souvent de spécialiser le code à l'architecture de la puce afin d'en exploiter correctement les ressources. 2) Utiliser une MVP pour fournir une couche d'abstraction logicielle offrant les mécanismes de cohérence de cache, ce qui peut s'avérer coûteux pour les performances mais permet de déployer des applications écrites pour la mémoire partagée de façon relativement transparente. Enfin 3) utiliser des modèles de programmation spécifiques tels que le flot de données (*dataflow*), offrant une couche d'abstraction de la puce, mais au prix de se conformer à l'organisation des tâches et des données imposées par le modèle choisi.

Plusieurs initiatives ont été proposées pour abstraire la complexité matérielle des many-cœurs, en offrant aux applications une vision logique centralisée de la puce. C'est le cas par exemple du SSI (*Single System Image*) ALMOS [Almaless, 2014], un système d'exploitation conçu pour le many-cœur de recherche TSAR [LIP6, 2009] développé au laboratoire d'informatique de Paris 6 (Lip6). Tout comme les SSI habituellement déployés sur des grappes de calculateurs, ALMOS offre une MVP ainsi qu'un ordonnanceur de processus donnant l'illusion d'être exécuté sur une machine NUMA. Une autre initiative vers la conception d'un SSI [Penna et al., 2019] propose une HAL (*Hardware Abstraction Layer*) pour la gestion des cœurs et de la mémoire pour le MPPA Bostan. Un troisième exemple est le développement d'une MVP logicielle sur la puce Epiphany [Richie et al., 2017]. Ces projets montrent l'importance de proposer un environnement logiciel suffisamment haut-niveau afin de masquer la complexité architecturale des many-cœurs. Dans ce contexte, j'ai participé à plusieurs projets et proposé des contributions pour faciliter le déploiement d'applications.

Contributions pour la programmabilité des processeurs many-cœurs. En 2008, plusieurs projets académiques et industriels sont lancés afin de concevoir des processeurs many-cœurs. Parmi ceux-ci, une initiative du CEA et de la startup Kalray vise à proposer une architecture pouvant accueillir de 256 à 1024 coeurs. Je rejoins en 2009 le laboratoire commun CEA-Kalray (2009-2012) dont l'objectif est de proposer un environnement de programmation de cette puce et d'influencer l'architecture matérielle. Nous proposons un modèle de programmation flot de données basé sur une variation du modèle CSDF dans lequel des agents communiquent à travers des ports et des liens de communication, suivant une spécification des échanges fournie par l'application. Cette particularité permet d'effectuer de nombreuses vérifications lors de la compilation, notamment l'absence d'interblocages, un cadencement correcte des tâches ou encore un dimensionnement possible des tampons de communication. Le langage de programmation associé, Σ-C [Goubier et al., 2011], est supporté par une chaîne de compilation *source-to-source* [Aubry et al., 2013] qui a été transférée et industrialisée par Kalray. Dans le laboratoire commun, j'ai été responsable des tests unitaires, des tests d'intégration et de la livraison de la chaîne de compilation Σ-C à Kalray. J'ai aussi contribué à la conception de différentes étapes de la chaîne de compilation décrites dans la suite de ce manuscrit, ce qui représente environ 45 000 lignes de code C sur 245 000 et 6 publications dans des conférences avec actes.

En 2012 je renoue avec certaines thématiques explorées pendant ma thèse réalisée à l'INRIA sur la cohérence des données et les MVP pour grilles de calculateurs. Je propose de lancer une activité sur la mémoire (virtuellement) partagée dans le laboratoire. L'idée directrice est de proposer des protocoles de cohérence adaptés pour les architectures many-cœurs et leurs applications, puis de faciliter le choix et la configuration en fournissant des outils d'aide à la décision. En collaboration avec des collègues, des doctorants et des stagiaires que j'encadre, nous proposons trois protocoles de cohérence basés sur la coopération entre les caches des PE. Une chaîne de compilation est spécifiée, dont le but est d'analyser un code source parallèle et d'aider à déterminer quelles sont les configurations de protocoles de cohérence les plus adaptées pour chaque variable partagée. Plusieurs briques de cette chaîne de compilation sont réalisées dans le cadre de la thèse de Safae Dahmani [Dahmani, 2015] soutenue en 2015, que j'ai proposée et co-encadrée avec Guy Gogniat de l'UBS. Ces travaux se traduisent notamment par une douzaine de publications dans des conférences avec actes et un brevet.



2.1 Σ-C : un langage flot-de-données cyclo-statique

Σ-C [Goubier et al., 2011] est un langage flot-de-données proposé par le CEA pour les architectures processeur massivement parallèles. À l'origine conçu dans le cadre d'une collaboration avec la compagnie *fabless*[1] Kalray pour la puce MPPA-256 (une description de la puce se trouve dans la figure 2.3), Σ-C est en fait un langage dédié (un DSL (*Domain Specific Language*)) mais non spécialisé à une architecture particulière. C'est la chaîne de compilation (le compilateur) conçu pour Σ-C qui permet de cibler différentes architectures d'exécution : par exemple un binaire pour un processeur classique x86 (un programme parallèle utilisant les *Posix threads*), un binaire pour un simulateur d'architecture (un programme lié avec l'API (*Application Programming Interface*) d'un ISS (*Instruction Set Simulator*)) ou un binaire ciblant une architecture many-cœur comme le MPPA. Le compilateur est aussi capable d'effectuer de nombreuses vérifications, les principales étant la cohérence du graphe de communications, la vivacité à l'exécution, le débit de traitement et dans certains cas la latence bout en bout. Le développement de Σ-C répond à un besoin particulier : comment exploiter une puce massivement parallèle, ne disposant pas de mémoire partagée entre les clusters, en respectant des contraintes d'embarquabilité, notamment pour l'empreinte mémoire ? Le flot-de-données apporte un cadre formel pour des applications déployées sur le MPPA, nécessairement distribuées entre clusters, et ce cadre tend vers un déterminisme d'exécution (pouvoir reproduire des exécutions avec un entrelacement des évènements identique) qui est un prérequis pour cibler des applications embarquées pseudo temps-réelles (traitement de flux de données, traitement d'image pour véhicules autonomes…).

Σ-C a permis de programmer le premier processeur MPPA conçu par Kalray et de démontrer la performance calculatoire et énergétique de l'approche : En 2012 Kalray annonce que le MPPA-256 (Andey) est capable d'effectuer un encodage vidéo H264 aussi rapidement qu'un processeur Intel i7 de la même époque, mais avec une consommation d'énergie entre 6 et 20 fois moindre [de Dinechin et al., 2013a] ! Ceci constitue une véritable avancée pour un processeur générique, là où ce genre de résultat est habituellement obtenu par des accélérateurs spécialisés tels que les ASIC (*Application-Specific Integrated Circuit*), les GPGPU ou les FPGA. Ceci valide aussi les choix et la qualité de la R&D que nous avons menés dans le laboratoire commun du CEA LIST. Mais avant de décrire les contributions effectuées sur le modèle de calcul et la chaîne de compilation, revenons sur les grands principes du flot-de-données.

Dans un modèle de programmation flot-de-données, les calculs sont exprimés par un ensemble d'acteurs dotés de canaux de communication. L'ensemble forme un graphe orienté dans lequel les nœuds représentent les traitements et les liens représentent les flux de données. Dans les années 1970, Gilles Kahn propose les KPN (*Kahn Process Network*) [Kahn, 1974], un modèle de calcul distribué dans lequel des processus déterministes communiquent via des canaux FIFO (*First In First Out*) de taille infinie. Les langages dérivés des réseaux de processus sont rapidement utilisés dans le contexte du traitement du signal [Lee and Parks, 1995] puis dans d'autres contextes HPC et embarqués, pour le multimédia notamment [Denolf et al., 2007, Michalska et al., 2015]. Ils permettent d'exprimer facilement un programme parallèle dans lequel la problématique de la concurrence des accès aux données est décidée par construction. Un cas particulier du KPN est le CSDF [Bilsen et al., 1996], dans lequel les consommations et les productions de chaque acteur

1. Dans la microélectronique, une compagnie *fabless* conçoit des puces mais ne les *fond* pas, c'est-à-dire qu'elle ne les réalise pas physiquement.

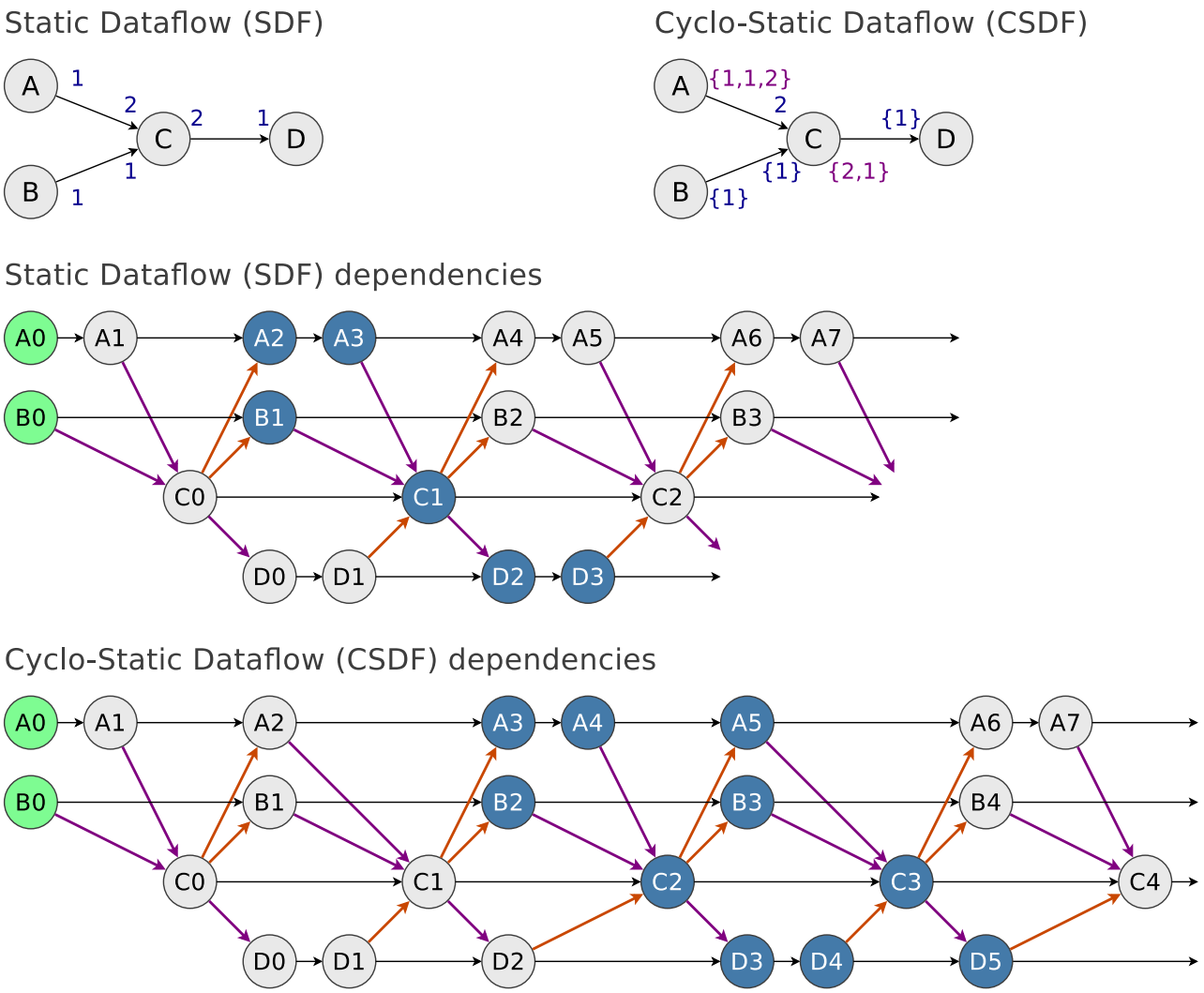


FIGURE 2.7 – Exemples de graphes de communication et de dépendances pour des flots de données SDF (*Static Data-Flow*) et CSDF.

sont explicitement indiquées dans le programme. L'appellation *Cyclo-Static* provient du fait que ces consommations et productions ne sont pas nécessairement les mêmes à chaque exécution de l'acteur, mais elles doivent suivre une séquence prédéfinie qui est donc répétée de manière cyclique.

Dans un langage flot de données, les acteurs peuvent être vus comme des machines à état dont le fonctionnement est le suivant : A) l'acteur attend que les canaux en entrée contiennent le nombre de données spécifiées par le programme. B) lorsque toutes les données sont présentes, elles sont lues de manière atomiques et utilisées pour le calcul (le code utilisateur). Cette phase est appelée *occurrence* (*firing* en anglais). C) les résultats sont écrits de manière atomique sur les canaux de sortie selon la spécification établie dans le programme. Puis l'acteur reboucle sur A. Il en résulte un fonctionnement sans terminaison. Une propriété fondamentale pour un flot de données est la vivacité (*liveness* en anglais) qui assure que chaque acteur est bien exécuté de manière infinie [Benazouz et al., 2013]. Cette notion se rapproche finalement de la problématique de la réception des messages dans les systèmes distribués avec l'emploi du qualificatif *eventually* en anglais qui signifie que le message finira bien par arriver, sans pour autant être en mesure de déterminer une date pour cet évènement. L'un des intérêts du CSDF est de donner un cadre suffisamment contraint, avec une description du comportement des acteurs suffisamment explicite pour construire des preuves sur des propriétés telles que la vivacité, l'absence d'interblocages ou le dimensionnement minimal des canaux de communication.

La figure 2.7 illustre les graphes de communications et de dépendances respectifs de deux exemples, l'un SDF et l'autre CSDF. Ces deux exemples sont composés de 4 acteurs, la structure du graphe de communications est la même mais les productions et les consommations sont différentes. Celles-ci sont indiquées pour chaque canal de communication : la quantité de données produites par l'acteur émetteur est indiqué à l'origine du canal, sur le port de sortie, alors que la quantité de données consommées est indiqué sur le port d'entrée. Dans le cas du SDF, ces quantités sont les mêmes à chaque occurrence de l'acteur. Dans le cas du CSDF, les quantités sont exprimées sous la forme d'un vecteur indiquant la séquence des productions ou des consommations émises ou reçues sur le port de communication. Par exemple l'acteur A produit 1 donnée à la première occurrence, 1 donnée à la deuxième occurrence, 2 données à la

troisième occurrence puis reboucle sur le début de la séquence pour la quatrième occurrence. À partir de ces graphes il est possible de construire le graphe des dépendances entre les occurrences. Pour chaque acteur X, les occurrences sont identifiées par la suite infinie $(X_0, X_1, \ldots)$.

Un graphe de dépendances est construit en reliant les occurrences par des relations de causalité, c'est-à-dire une dépendance reliant les deux évènements. Les occurrences n'ayant aucune dépendance sont les points d'entrée du graphe, elles sont représentées en vert dans la figure 2.7. Ces occurrences sont directement exécutées lors du lancement de l'application. Une relation de causalité évidente est donnée par $X_i \rightarrow X_{i+1}$, qui indique que l'occurrence X_{i+1} ne peut se produire avant X_i. En déroulant symboliquement les productions et consommations spécifiées dans le graphe SDF ou CSDF, il est possible de déterminer les liens de causalité entre les occurrences d'acteurs différents. Ces liens sont de deux types : les dépendances induites par les productions-consommations et les dépendances induites par le dimensionnement des canaux de communication. Pour les productions-consommations, une dépendance $X_i \rightarrow Y_j$ existe si la quantité de données attendue sur le port d'entrée de Y à l'occurrence j a été produite par un nombre suffisant d'occurrences de X se terminant par l'occurrence X_i. Ces dépendances sont représentées en violet sur la figure. Pour le dimensionnement des canaux, une dépendance $X_i \rightarrow Y_j$ existe si l'acteur X a suffisamment consommé de données sur le canal $Y \rightarrow X$ pour que Y puisse de nouveau y produire des données. Ces dépendances sont représentées en orange sur la figure. Dans cet exemple, la taille des canaux de communication correspond à la valeur maximale du nombre de productions ou de consommations. Dans le cas présent, cette approche permet un dimensionnement suffisant des canaux, mais c'est en réalité un cas particulier, comme discuté dans la section 2.1.3.

Le graphe de dépendances est un objet apportant de nombreuses informations sur l'application : il est notamment très utile pour le bon paramétrage de l'environnement d'exécution (le *runtime*). Il permet d'effectuer des vérifications hors-ligne et des optimisations pour le dimensionnement des canaux de communication et le placement-routage de l'application sur la puce. Il est aussi utilisé en-ligne pour diriger l'ordonnancement précalculé des tâches, préparer les contextes des prochaines tâches à exécuter ainsi que les espaces de stockage pour transférer les données sur les canaux de communication. Sous la forme présentée en figure 2.7, le graphe est sans terminaison et il faut donc trouver une représentation finie et compacte pouvant être embarquée à l'exécution. Pour cela, il est possible de calculer le vecteur de répétition. Le vecteur de répétition est un sous-graphe du graphe de dépendances identifié comme un motif pouvant être répété à l'infini. À partir de ce motif il est possible de reconstituer le graphe de dépendances, ou tout du moins le *régime établi* de l'application, la phase d'initialisation (le *bootstrap*) nécessitant une description spécifique. Le vecteur de répétition est indiqué en bleu sur la figure 2.7. Dans Σ-C le vecteur de répétition est encodé sous une forme compacte afin de limiter l'empreinte mémoire une fois chargé sur la puce et une méthode de compression est appliquée sur le modèle de temps logique vectoriel (LVT (*Logical Vector Time*)) [Dubrulle et al., 2012].

La chaîne de compilation Σ-C implémente toutes les étapes nécessaires pour l'analyse d'un programme fourni sous forme de texte, jusqu'à la construction d'un binaire embarquant le code utilisateur, le logiciel système ainsi que les objets décrivant le comportement de l'application. La réalisation de cette chaîne de compilation a nécessité un effort équivalent à une équipe d'une dizaine de personnes sur 3 ans. Un accord de confidentialité et d'exclusivité a été établi entre le CEA et Kalray pour la durée du laboratoire commun, à l'issue duquel il a été possible de communiquer sur les résultats et de publier. Dans ce projet, j'ai effectué plusieurs contributions : l'instanciation du graphe de communication, la compilation et la réduction du parallélisme ou encore le calcul des motifs d'accès aux tampons partagés.

2.1.1 Une chaîne de compilation *source-à-source*

> Aubry, P., Beaucamps, P., Blanc, F., Bodin, B., Carpov, S., Cudennec, L., David, V., Dore, P., Dubrulle, P., de Dinechin, B. D., Galea, F., Goubier, T., Harrand, M., Jones, S., Lesage, J., Louise, S., Chaisemartin, N. M., Nguyen, T., Raynaud, X., and Sirdey, R. (2013). Extended cyclostatic dataflow program compilation and execution for an integrated manycore processor. In Alexandrov, V. N., Lees, M., Krzhizhanovskaya, V. V., Dongarra, J. J., and Sloot, P. M. A., editors, Proceedings of the International Conference on Computational Science, ICCS 2013, Barcelona, Spain, 5-7 June, 2013, volume 18 of Procedia Computer Science, pages 1624–1633. Elsevier.
>
> [Aubry et al., 2013] CEA LIST, Kalray SA.
>
> Contributions | ☐ initiative ■ idées ■ implémentation ☐ expérimentations ■ rédaction ☐ présentation

Écrire un programme flot-de-données est relativement aisé, et peut se faire dans la plupart des langages de programmation, par exemple en C, `Python` ou `Fortran`. La plupart du temps, les méthodes et bonnes pratiques issues du génie logiciel poussent naturellement à structurer le code sous forme de tâches qui communiquent par le biais de messages ou de variables partagées. Alors finalement, pourquoi proposer un nouveau langage ? La question est critique à l'échelle d'un projet de recherche, et encore plus lorsqu'il s'agit d'un transfert technologique vers l'industrie. Pour les développeurs d'applications métier, utiliser un nouveau langage est source de nombreux désagréments : l'apprentissage de la syntaxe, des concepts objets et du modèle d'exécution sous-jacent demande un investissement en temps

(et en argent s'il faut recourir à des formations). Les développeurs perdent de plus l'usage des bibliothèques de code maison constituées et éprouvées dans les projets passés. Proposer un nouveau langage répond donc à un besoin, en faisant le pari que l'investissement concédé par les développeurs pour l'adopter sera récompensé à terme par un code plus qualitatif, de meilleures performances, de nouvelles fonctionnalités ou un cycle de développement raccourci.

Listing 1 –  Un flot de données écrit en Python avec MPI, sous forme d'une chaîne d'agents (pipeline).

```python
#!/usr/bin/env python3
# $ mpirun -np 3 python3 python_dataflow.py

from mpi4py import MPI
from time import sleep

class DataflowAgent:
    def __init__(self, args=None):
        self.comm = MPI.COMM_WORLD
        self.rank = self.comm.Get_rank()
        self.size = self.comm.Get_size()
        print(str(self.rank) + " started")

        # create a pipeline starting from MPI rank 0
        if (self.rank > 0):
            self.input_port = self.rank - 1
        if (self.rank < self.size - 1):
            self.output_port = self.rank + 1

    def firing(self, running=True):
        if (self.rank > 0):
            running = self.comm.recv(source=self.input_port)
            # this is the place to compute something
            sleep(1)
        if (self.rank < self.size - 1):
            self.comm.send(running, dest=self.output_port)
        print(str(self.rank) + " fired with running value " + str(running))
        return running


def main ():
    dataflow_agent = DataflowAgent()
    if (dataflow_agent.rank == 0):
        for i in range(3):
            dataflow_agent.firing(True)
        dataflow_agent.firing(False)
    else:
        while dataflow_agent.firing():
            pass

if __name__  == "__main__":
    main()
```

Prenons par exemple le code `Python` présenté dans le listing 1 et dans lequel un flot-de-données simple est implémenté. Le programme permet de déployer une chaîne de n agents (un *pipeline*). Chaque agent est exécuté par un processus MPI et peut donc communiquer par ce moyen. Un booléen transite tout le long de la chaîne et est utilisé comme drapeau pour la terminaison : si la valeur reçue est `False`, l'agent écrit cette valeur sur son port de sortie et termine son exécution, ce qui permet la terminaison de l'application. Le code est décomposé en trois parties remplissant des fonctions distinctes et pouvant être généralisées à tout programme flot-de-données : 1) Dans la fonction d'initialisation de l'agent `__init__`, les ports de communication sont déclarés et *connectés*. Ici, l'identifiant du processus MPI distant est indiqué dans les variables `input_port` et `output_port`. Cette étape permet de décrire le graphe de communication. 2) La fonction `firing` correspond aux instructions exécutées lors d'une occurrence de l'agent. Elle effectue de manière séquentielle la lecture de la donnée reçue sur le port d'entrée, un éventuel traitement utilisateur (ici un simple appel à la fonction `sleep`) suivi de l'écriture du résultat sur le port de sortie. Enfin 3) des boucles dans la fonction `main` implémentent l'automate distribué en itérant sur les occurrences jusqu'à la terminaison de l'application.

Sur la quarantaine de lignes de code, seule une ligne concerne le code utilisateur, c'est-à-dire directement liée à l'algorithme implémenté pour résoudre un problème (l'instruction `sleep` dans cet exemple). Le reste du programme est ce que l'on appelle un squelette (*code skeleton* en anglais), c'est-à-dire du code permettant de structurer les traitements et les données, bien souvent générique, compilable, mais dont l'exécution ne produit pas de résultat. Écrire un programme flot-de-données à la main consiste donc à écrire du code utilisateur métier et à l'enrober dans un squelette en charge de construire le graphe de communication. Concevoir ce squelette n'est pas anodin car il détermine la qualité du programme en terme de performance calculatoire, notamment par l'emploi du parallélisme de tâches. C'est aussi un élément déterminant sur le plan fonctionnel : que se passe-t-il si des agents ne sont pas correctement connectés, si une connexion est établie par erreur, si les données qui transitent ne sont pas celles attendues, si un agent produit à une cadence plus élevée que le consommateur ? Ces erreurs ne sont pas détectées par un compilateur classique et les problèmes surgissent trop tard, à l'exécution, parfois de manière silencieuse. Détecter à la compilation des erreurs structurelles nécessite une modification lourde du squelette et justifie : 1) la spécification formelle du squelette afin de contraindre le développeur à écrire des programmes corrects par construction et 2) l'utilisation d'outils d'analyse et de vérification afin de fournir la preuve que l'application peut s'exécuter dans un contexte donné.

> Ces deux points correspondent au besoin de proposer un nouveau langage et d'en assurer la compilation pour vérifier un certain nombre de propriétés avant l'exécution.

Dans la littérature plusieurs langages flot-de-données et chaînes de compilation ont été proposés pour programmer des architectures embarquées massivement parallèles, tels que Streamit [Amarasinghe et al., 2005], Brook [Buck et al., 2004], XC [Watt, 2009] ou OpenCL (qui est aujourd'hui largement répandu pour programmer les GPU et les FPGA). À la fin des années 2000 le modèle de calcul le plus proche de Σ-C est Streamit et son compilateur RAW [Gordon et al., 2002]. Comparé à Streamit/RAW le modèle de calcul Σ-C apporte plusieurs fonctionnalités : 1) Σ-C autorise l'ordonnancement dynamique des tâches sur la puce, 2) le NoC du MPPA implique que les communications doivent être implémentées indifféremment sur une mémoire partagée, sur un DMA (*Direct Memory Access*) ou du MP, ce qui n'est pas permis par le logiciel système de Streamit, 3) la granularité des tâches est plus fine que le concept de filtre Streamit et 4) le modèle CSDF permet une plus grande expressivité de part la construction très libre de la topologie du graphe de communication, à l'époque plus élaboré que les séries parallèle-séquentiel de RAW. Σ-C apparaît donc comme un compromis satisfaisant entre d'une part un langage doté d'une grande expressivité, notamment sur la construction du graphe de communication, et d'autre part offrant un cadre de développement suffisamment contraint pour produire des applications correctes par construction et dont certaines propriétés critiques peuvent être vérifiées à la compilation. Quelques années après Σ-C, une extension hiérarchique du modèle SDF est proposée par l'IETR/INSA de Rennes en collaboration avec Kalray pour prendre en compte l'aspect dynamique d'une application [Hascoet et al., 2017]. Ceci montre l'intérêt du flot-de-données pour la programmation de puces massivement parallèles, en complément des HAL plus bas-niveau. Mais revenons au début des années 2010.

Dans Σ-C plusieurs choix ont été faits afin de favoriser le portage du code métier existant (*legacy code* en anglais). Le langage est un sur-ensemble du langage ANSI C. Il est donc possible d'écrire le code utilisateur en C, de le copier à partir d'une application C existante et d'utiliser les bibliothèques tierces. Le langage Σ-C consiste en un ensemble de mots clés spécifiques utilisés pour décorer et structurer un code C. Le but du compilateur est alors de transformer le programme Σ-C en un programme ANSI C qu'il est possible de compiler avec un outil classique tel que `gcc`. C'est donc un compilateur source-à-source. Un autre avantage de cette approche est de s'abstraire de la plateforme matérielle ciblée, en laissant la possibilité d'utiliser un compilateur spécifique pour produire le binaire et ainsi profiter des optimisations associées. Il est aussi possible d'utiliser une chaîne de compilation croisée pour générer un binaire ciblant une architecture processeur différente du processeur de la machine hôte.

Listing 2 –  Un sous-graphe Σ-C racine permettant de construire (instancier) le graphe de communication, issue d'une application de détection de contours d'une image basée sur un opérateur Laplacien.

```
1  subgraph root(int width, int height) {
2          interface { spec {}; }
3          map {
4                  agent output = new StreamWriter<int>(ADDROUT, width * height);
5                  agent sy1 = new Split<int>(width , 1);
6                  agent sy2 = new Split<int>(width , 1);
7                  agent jf = new Join<int>(width , 1);
8                  connect (jf.output, output.input);
9                  for (i=0; i < width ; i ++) {
10                         agent cf = new ColumnFilter (height);
11                         connect (sy1.output[i], cf.in1);
```

```
12                      connect (sy2.output[i], cf.in2);
13                      connect (cf.out1, jf.input[i]);
14              }
15      }
16  }
```

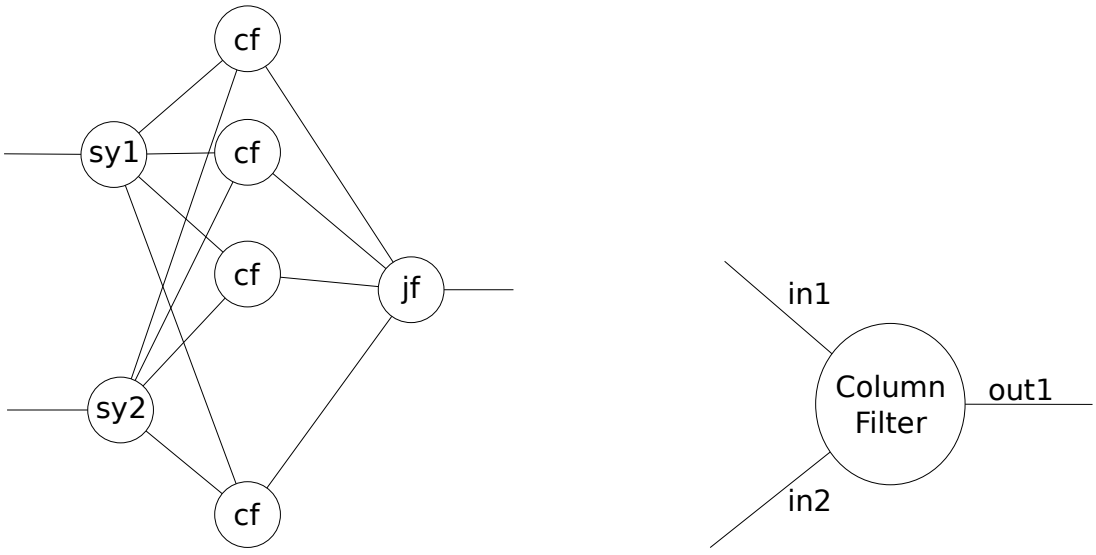
Listing 3 – Un agent Σ-C encapsulant un code utilisateur convolutif.

```
1   agent ColumnFilter(int height) {
2           interface {
3                   in<int> in1, in2;
4                   out<int> out1;
5                   spec { in1[height]; in2[height]; out1[height]}; }
6           void start () exchange (in1 a[height], in2 b[height], out1 c[height]) {
7                   static const int
8                           g1 [11] = {-1, -6, -17, -17, 18, 46, 18, -17, -17, -6, -1} ,
9                           g2 [11] = {0, 1, 5, 17, 36, 46, 36, 17, 5, 1, 0};
10                  int i, j;
11                  for (i=0; i < height; i++) {
12                          c[i] = 0;
13                          if (i < height - 11)
14                                  for (j=0; j < 11; j++) {
15                                          c[i] += g2[j] * a[i + j];
16                                          c[i] += g1[j] * b[i + j]; }
17                  }
18          }
19  }
```

Une application Σ-C est composée d'agents et de sous-graphes. Un exemple est donné dans les listings 2 et 3 et les graphes de communication correspondant sont respectivement donnés dans les figures 2.8a et 2.8b. Les mots clés ajoutés au langage C sont indiqués en **orange**. Les agents sont des entités autonomes pourvues d'un fil d'exécution et d'un espace mémoire propre. Une interface permet de décrire les ports de communication, leur direction et le type de données acceptées. La section **interface** contient aussi une spécification (section **spec**) du comportement de l'agent définie comme une séquence circulaire de transitions, chaque transition étant caractérisée par des quantités de données reçues et/ou émises sur les ports de communication. Il existe deux types d'agents : les agents utilisateur dont le code est fourni par le développeur, et les agents système spécialisés dans la réorganisation des données (par exemple les agents `Split` et `Join` permettent de dispatcher un flux de données sur un ensemble de tâches de traitement puis de regrouper les résultats). Les agents système sont fournis par la chaîne de compilation et leur compilation est décrite en section 2.1.3.

Un sous-graphe **subgraph** est un ensemble d'agents et/ou de sous-graphes interconnectés. Tout comme les agents, un sous-graphe est caractérisé par une interface avec des ports de communication. En revanche, la spécification du comportement du sous-graphe est inférée à partir de la composition des comportements des entités qu'il contient. Une autre particularité est que les ports du sous-graphe sont connectés à l'extérieur et à l'intérieur, ce qui permet aux données d'entrer et sortir du sous-graphe. Les agents et les sous-graphes sont instanciés dans une section du code appelée section **map**, en utilisant le mot clé **new**. Dans cette section il est possible d'écrire du code C classique et d'utiliser notamment des boucles pour instancier et connecter de nombreux agents et sous-graphes via la primitive **connect**. Les définitions récursives sont autorisées et encouragées. La section **map** est compilée et exécutée hors-ligne. Une conséquence directe est que le graphe de communication est statique : aucune nouvelle instance ou modification du graphe n'est possible pendant l'exécution. Ce choix est dirigé par le besoin de maîtriser le comportement de l'application sur la puce et permet d'apporter des preuves notamment sur l'empreinte mémoire ou la bande passante. Une autre conséquence est que l'environnement de compilation est susceptible de différer entre la compilation de la section **map** sur la machine hôte et la compilation du code utilisateur pour la cible : si le recours au langage ANSI C permet d'atténuer largement cette contrainte, certaines bibliothèques tierces peuvent ne pas être disponibles dans les deux cas et il faut donc programmer ces deux sections en conséquence.

La chaîne de compilation Σ-C est organisée en plusieurs étapes, chacune intervenant sur des opérations de transformation du code, de vérification de propriétés, d'optimisation du graphe de communication et de génération d'objets pour l'édition des liens. La figure 2.9 présente ces différentes étapes, sous forme d'une chaîne de compilation, du code source à la construction du binaire. Les contributions suivantes concernent les phases d'instanciation, de réduction du parallélisme et de dimensionnement des tampons de communication.



(a) (b)

FIGURE 2.8 – Graphe de communication obtenu après exécution de la section **map** dans le code Σ-C du listing 2 (Figure 2.8a). Agent ColumnFilter (cf) obtenu après la compilation du code Σ-C du listing 3 (Figure 2.8b).

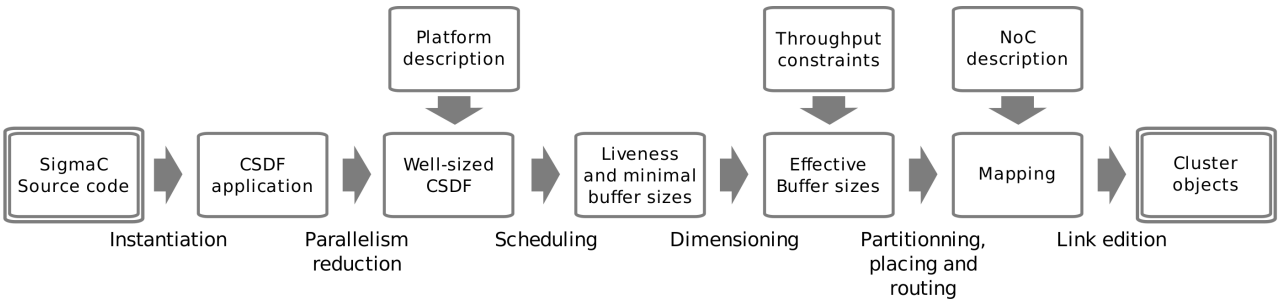


FIGURE 2.9 – Les différentes étapes de la chaîne de compilation Σ-C.

2.1.2 Réduire le parallélisme à la compilation

> Carpov, S., Cudennec, L., and Sirdey, R. (2013). Throughput constrained parallelism reduction in cyclo-static dataflow applications. In Alexandrov, V. N., Lees, M., Krzhizhanovskaya, V. V., Dongarra, J. J., and Sloot, P. M. A., editors, Proceedings of the International Conference on Computational Science, ICCS 2013, Barcelona, Spain, 5-7 June, 2013, volume 18 of Procedia Computer Science, pages 30–39. Elsevier.
>
> [Carpov et al., 2013] CEA LIST.
>
> Contributions | ☐ initiative ■ idées ☐ implémentation ☐ expérimentations ☐ rédaction ☐ présentation

> Cudennec, L. and Sirdey, R. (2012). Parallelism reduction based on pattern substitution in dataflow oriented programming languages. In Ali, H. H., Shi, Y., Khazanchi, D., Lees, M., van Albada, G. D., Dongarra, J. J., and Sloot, P. M. A., editors, Proceedings of the International Conference on Computational Science, ICCS 2012, Omaha, Nebraska, USA, 4-6 June, 2012, volume 9 of Procedia Computer Science, pages 146–155. Elsevier.
>
> [Cudennec and Sirdey, 2012] CEA LIST.
>
> Contributions | ■ initiative ■ idées ■ implémentation ■ expérimentations ■ rédaction ■ présentation

Σ-C permet d'écrire des programmes parallèles et répartis. Le parallélisme peut s'exprimer au niveau de l'agent, en écrivant du code parallèle, par exemple avec les *Posix Threads* ou *OpenMP* si l'architecture cible le permet. Le parallélisme est aussi naturellement exprimé par le flot de données, dans lequel les instances d'agents sont autant de processus exécutés de manière concurrente. Dans cette section nous considérons uniquement le parallélisme induit par le flot-de-données.

La compilation du parallélisme consiste à identifier et instancier les agents et les liens de communication déclarés dans la section **map** de l'application. Cette étape construit un graphe de communication dans lequel les noeuds sont des instances d'agent et les liens sont des connexions entre ports de communication. De l'information sémantique est attachée sur le graphe afin de faire le lien avec le code source et les autres résidus de compilation. Différentes

analyses sont effectuées sur ce graphe afin de vérifier qu'il est correctement formé. Dans le code Σ-C, l'instanciation des agents s'effectue hors-ligne, en exécutant un code utilisateur écrit dans la section **map**. Le modèle de programmation encourage le développeur à exprimer un haut degré de parallélisme, c'est-à-dire une granularité de tâches très fine. Par exemple, un traitement appliqué à une matrice peut s'exprimer en déclarant une tâche pour chaque ligne, ce qui peut entraîner un nombre important d'agents à instancier dans le graphe, et donc à déployer sur l'architecture cible.

Le parallélisme pour many-cœurs est caractérisé par une courbe d'accélération (le *speed-up*) proche des multi-cœurs dans lequel une utilisation raisonnable du CPU consiste à exécuter autant de processus que de cœurs physiques disponibles. Si l'application présente moins d'instances que de PE alors la puce est sous-exploitée (ce qui ne veut pas dire pour autant que l'application n'est pas correctement dimensionnée). Si l'application présente plus d'instances que de PE alors il y existe un seuil à partir duquel la puce devient surexploitée et les performances s'écroulent. Ce seuil est atteint lorsqu'un élément matériel placé sur le chemin critique est saturé. Par exemple lorsque le processeur passe trop de temps à effectuer des changements de contexte entre les processus, lorsque la hiérarchie de cache est polluée par les changements de contexte ou encore lorsqu'il existe trop de contention sur le NoC. Ces phénomènes de saturation et de contention apportent de plus beaucoup d'incertitude sur les temps d'exécution et ne permettent que peu de garanties dans un environnement contraint ou pseudo temps-réel. Utiliser efficacement un processeur multi-cœur consiste donc à trouver le bon compromis entre d'une part 1) une application suffisamment parallèle, possiblement composée d'un nombre de tâches supérieur au nombre de cœurs, ce qui permet de masquer les temps d'attente lorsque des tâches effectuent des entrées-sorties, et d'autre part 2) réduire le parallélisme afin de ne pas tomber dans une situation de saturation ou de contention d'un élément matériel.

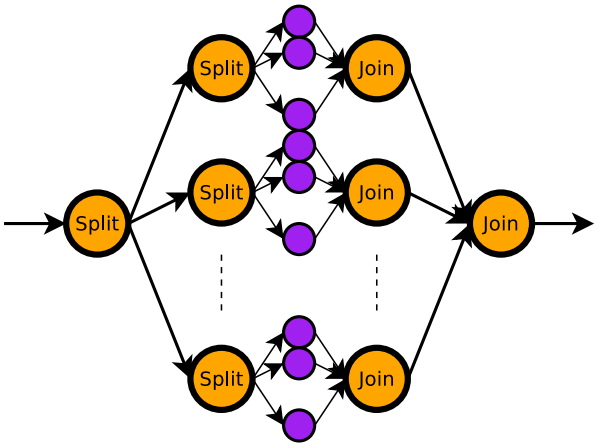
Dans le contexte d'un programme Σ-C compilé pour le MPPA, les contraintes liées à la maîtrise de l'exécution imposent de déterminer le nombre d'instances d'agents à la compilation, et ce afin de conserver un déterminisme d'exécution. Un travail de recherche a donc été effectué afin de réduire, si nécessaire, le parallélisme de l'application en fonction de divers objectifs tels que le nombre de tâches par coeur ou encore le débit de traitement de l'information désiré pour le flot de données, ce qui s'avère important pour les applications temps-réel.

L'approche par substitution de motifs [Cudennec and Sirdey, 2012] consiste à remplacer des parties du graphe de communication du flot de données par des sous-graphes équivalents. Le graphe de communication obtenu est structurellement différent mais il conserve la même spécification fonctionnelle. La réduction du parallélisme est possible dès lors que les sous-graphes substitués présentent un nombre d'instances utilisateur différent. Ceci concerne uniquement les agents utilisateurs, qui seront effectivement compilés sous forme de processus et nécessitent d'être déployés sur un PE (contrairement aux agents systèmes dont la compilation ne donne pas lieu à un processus et aux sous-graphes **sub-graph** qui n'ont pas d'existence au-delà de la construction du graphe de communication). Des exemples de motifs sont présentés dans les figures 2.10a, 2.10b et 2.10c. Ces motifs sont paramétrés, c'est-à-dire que la description structurelle du sous-graphe s'accompagne de règles de transformation vers un sous-graphe équivalent, ces règles étant paramétrées par un *facteur de réduction* f. Le rôle de f est d'appliquer la transformation du motif avec plus ou moins de force : c'est un réel compris entre 0 et 1, la réduction étant plus forte lorsque celui-ci tend vers 1 et son application est linéaire.

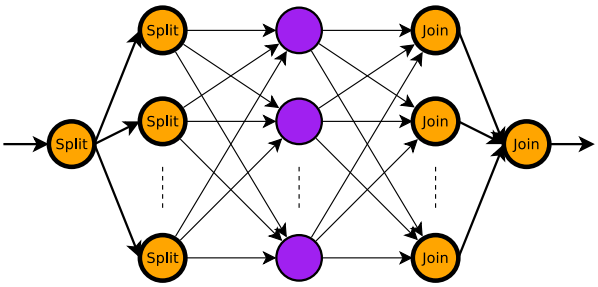
Considérons un motif Split-*-Join composé d'un agent système Split ouvrant sur n sous-graphes identiques et sans état (les *) et se fermant sur les n entrées d'un Join (c'est un cas particulier du motif cascade présenté en figure 2.10a avec une profondeur de 1). Le principe d'un Split est de distribuer les données entrantes sur les ports de sortie en utilisant un tourniquet (*round-robin* en anglais). Ce motif est détecté dans le graphe de communication à l'aide d'algorithmes d'isomorphisme de graphes. Pour chaque détection, la capacité de réduction (c) est calculée, c'est-à-dire le nombre maximal d'instances utilisateur pouvant être supprimées lorsque la fonction de réduction est appliquée avec $f = 1$. Dans le cas d'un Split-*-Join cette capacité est $c = n - 1$ car il faut au minimum une branche active pour conserver la fonctionnalité d'origine. Il est alors possible de remplacer le Split-*-Join par une structure équivalente mais dont le degré d'ouverture est n' avec $(n-c) \leq n' < n$. Les branches restantes traitent alors plus de données puisque le degré de parallélisme est plus faible. Le nouveau degré d'ouverture n' est obtenu en choisissant une valeur de f correspondante, ce qui s'avère être une tâche plus complexe qu'il n'y paraît et ce pour plusieurs raisons :

- un graphe de communication peut contenir plusieurs instances de différents motifs (par exemple plusieurs Split-*-Join). Il faut donc attribuer une valeur de f à chacune de ces instances. Une approche naïve consiste à déterminer le nombre d'instances utilisateur à supprimer (s) dans le graphe de communication entier, à calculer la capacité de réduction du graphe complet comme la somme des capacités des instances de motifs détectés ($c_g = \sum_{i=0}^{I} c_i$), de calculer un f_g global à partir de s, c_g et d'un objectif en terme de nombre de tâches à conserver dans l'application, et enfin d'appliquer ce f_g à toutes les instances de motifs en espérant que l'objectif soit atteint. Cette méthode est simple à mettre en œuvre mais n'est pas généralisable à cause du point suivant :
- des instances de motifs peuvent se trouver imbriquées dans d'autres instances de motifs. La conséquence directe est que lors de la réduction du parallélisme d'une instance, les instances imbriquées peuvent disparaître et l'objectif de réduction ne sera pas tenu (non atteint ou dépassé). Une autre conséquence est que l'opérateur de réduction n'est pas commutatif : l'ordre dans lequel les réductions sont appliquées sur les instances de motifs a une influence sur le résultat. Une solution pour réduire le parallélisme n'est donc pas seulement constituée des facteurs de réduction f_i, mais aussi de l'ordre dans lequel ils sont appliqués.

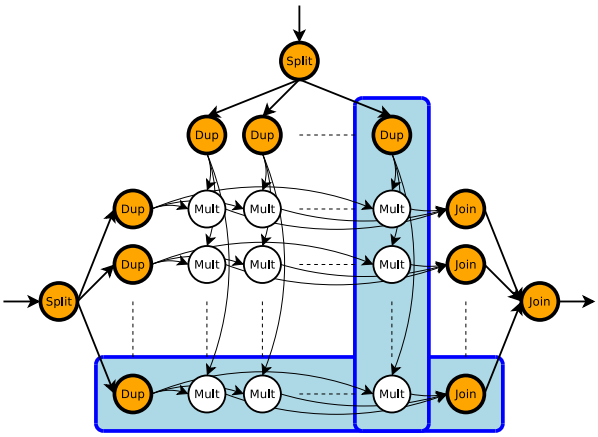
Un moteur de réduction du parallélisme a été implémenté dans la chaîne de compilation Σ-C, tel que décrit en figure 2.10d. Celui-ci détecte les motifs et construit différentes solutions pour appliquer une réduction dans un ordre particulier. La détection s'effectue par isomorphisme de graphes avec l'aide d'une table de Floyd-Warshall pour déterminer les distances des plus courts chemins entre deux instances dans le graphe. Cette table permet de vérifier rapidement des propriétés de symétrie et accélère la détection des motifs. L'exploration des solutions est un problème complexe car sujet à une forte combinatoire. En pratique, les applications rencontrées exhibent un nombre restreint de motifs, tout au plus une dizaine, et il est envisageable de mener une recherche exhaustive pour trouver une solution optimale. Dans le cas contraire, des stratégies métaheuristiques détaillées dans le papier permettent de classer les instances de motifs selon une relation d'ordre arbitraire (par capacité de réduction décroissante, par degré de recouvrement croissant avec d'autres motifs, par tri aléatoire…) et de ne considérer que les n premières instances afin de limiter l'espace de recherche. L'évaluation d'une solution nécessite la modification du graphe de communication pour calculer et évaluer le graphe transformé. Cette exploration est massivement parallélisable car la construction des solutions n'est pas dépendante des résultats précédents.



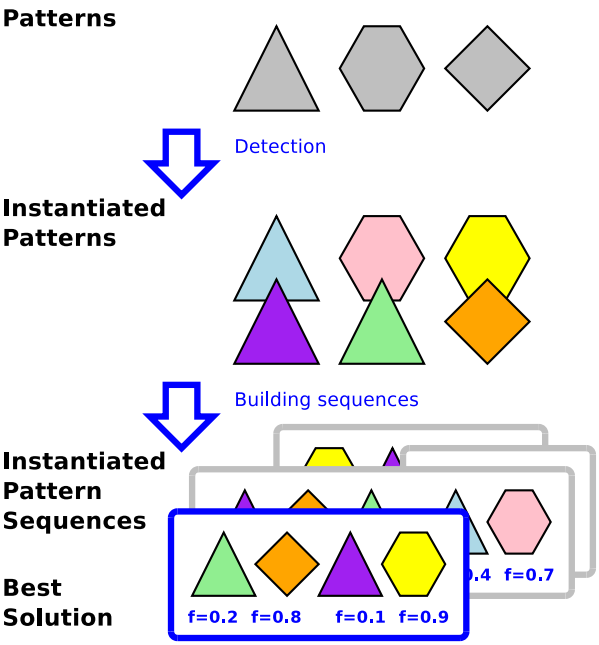
(a) Motif Split-Join récursif (cascade).



(c) Motif Split-Join en papillon.



(b) Motif de multiplication de matrice systolique.

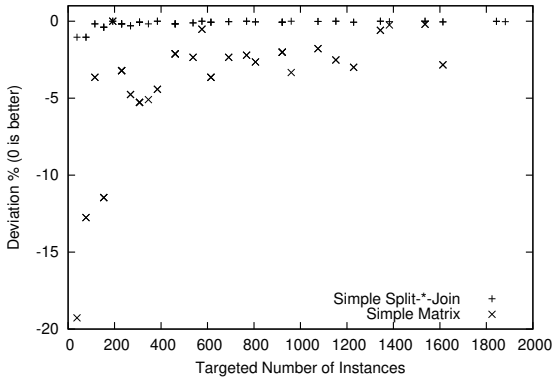


(d) Moteur de réduction du parallélisme par reconnaissance de motifs dans le graphe de communication et recherche d'une solution pour atteindre un objectif de nombre de tâches.

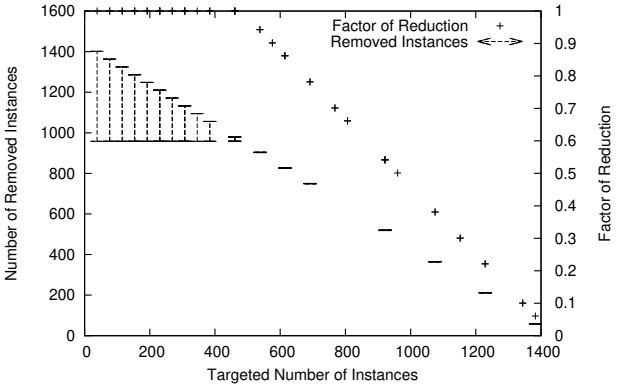
FIGURE 2.10 – Exemples de motifs structurels reconnus dans le graphe de communication de Σ-C et moteur de réduction du parallélisme, de la reconnaissance de motif à la recherche de solution.

La qualité des solutions trouvées par le moteur de réduction du parallélisme a été évaluée dans le cadre de deux applications synthétiques et d'une application de traitement d'image. Les résultats sont présentés sur la figure 2.11. Les principales conclusions sont les suivantes : réduire le parallélisme en présence d'un motif Split-*-Join est relativement précis car la granularité pour supprimer des tâches est fine. C'est ce qui est constaté sur la figure 2.11a. En revanche, le motif de multiplication de matrices implique de supprimer ligne par ligne et colonne par colonne, ce qui implique des ajustements à plus gros grain et explique un écart plus important avec les objectifs de réduction imposés. Enfin, la réduction du parallélisme sur le filtre Deriche illustrée en figure 2.11b montre qu'il est possible de redimensionner le graphe avec beaucoup de précision jusqu'à un certain seuil en dessous duquel il n'est plus possible de supprimer

des tâches. Le moteur de réduction du parallélisme retourne alors un graphe de communication minimal pour cette application.



(a) Pourcentage d'écart entre l'objectif fixé en terme de nombre de tâches et le nombre obtenu après réduction du parallélisme pour deux applications simples formées respectivement d'un motif Split-*-Join et d'un motif de multiplication de matrices. Les valeurs négatives indiquent que le moteur de réduction a supprimé plus de tâches que nécessaire.



(b) Nombre d'instances utilisateur supprimées et facteur de réduction appliqué pour un objectif fixé en terme de nombre de tâches après réduction du parallélisme pour une application de traitement d'image (un filtre Deriche composé de 2 motifs Split-*-Join, 1461 instances utilisateur et une capacité de réduction de 958 instances). Les barres d'erreur verticales représentent le nombre d'instances n'ayant pu être supprimées.

FIGURE 2.11 – Évaluation des solutions trouvées par le moteur de réduction du parallélisme.

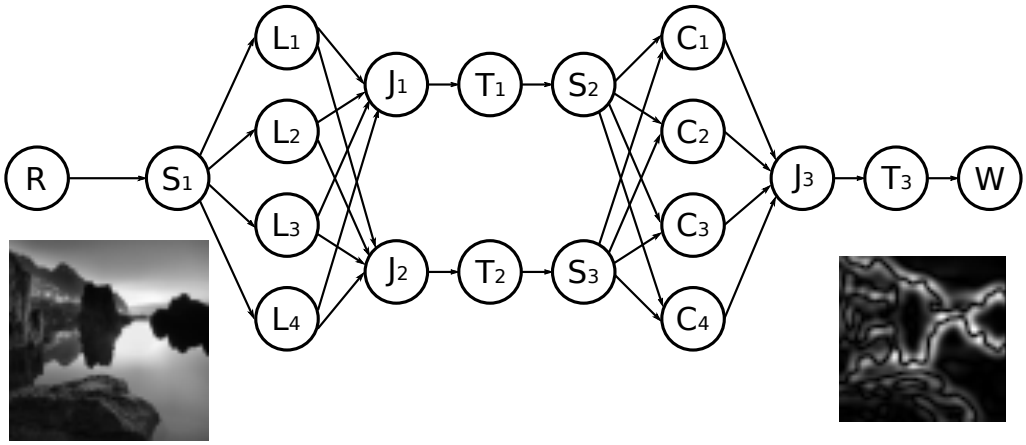
L'approche de réduction du parallélisme par substitution de motifs redimensionne le graphe de communication avec une grande précision afin de cibler une architecture matérielle spécifique. Si cette approche structurelle permet de conserver les fonctionnalités de l'application, elle modifie cependant le graphe sans prendre en compte la performance calculatoire de l'application. Ainsi les temps de calcul, les débits et latences bout en bout peuvent être dégradés suite à une décision malheureuse du moteur de réduction. La contribution suivante, complémentaire à cette approche, permet de réduire le parallélisme en prenant en compte le débit du flot-de-données.

L'approche par factorisation des instances équivalentes [Carpov et al., 2013] consiste à factoriser des instances d'agents et de multiplexer les communications entrantes et sortantes. Cette approche s'inspire des travaux menés dans la chaîne de compilation Streamit dans laquelle les tâches peuvent être fusionnées afin d'adapter la granularité de l'application à la plateforme matérielle [Gordon et al., 2006]. La méthode implémentée dans Streamit se limite cependant à 1) la fusion de tâches organisées sous forme de *pipeline*, ce qui revient à colocaliser les tâches sur un même PE et de faire en sorte que les communications entre ces tâches soient remplacées par un partage d'information efficace et 2) la fusion de tâches organisées sous forme d'un rideau de tâches dans un motif Split-*-Join, ce qui est similaire aux mécanismes de réduction du parallélisme par substitution de motifs présentés précédemment.

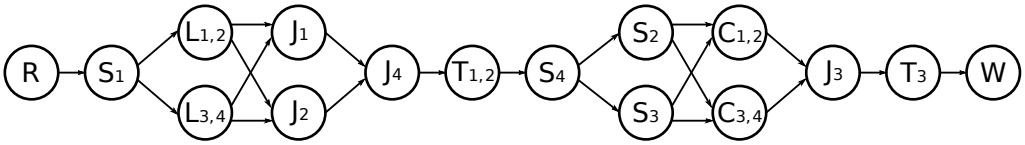
Dans ce travail, nous considérons toutes les instances d'agents équivalentes, indépendamment de leur organisation dans le graphe de communication, ce qui rend l'approche générique contrairement à celle de Streamit qui repose plus sur la notion de motifs. La réduction du parallélisme est appliquée de la manière suivante :

1. Pour chaque instance d'agent la fréquence d'exécution représente le nombre d'occurrences (*firings*) par unité de temps. Une fréquence d'exécution globale est fixée pour l'application complète en spécifiant la fréquence des occurrences des entrées et des sorties du graphe de communication. Les fréquences d'occurrence sont ensuite inférées pour chaque instance connectée à l'intérieur du graphe en se basant sur la spécification des productions et des consommations.
2. Les instances d'agent sont regroupées par classe d'équivalence, c'est-à-dire qu'elles sont regroupées dès lors qu'elles ont été instanciées à partir du même code source.
3. Chaque classe d'équivalence est partitionnée et les instances d'argent appartenant à une même partition sont remplacées par une seule instance fusionnée de la même classe. Un nouveau graphe de communication est créé en multiplexant les entrées des instances fusionnées par un Join et en démultiplexant les sorties des instances fusionnées par un Split. De plus, le partitionnement est établi de telle sorte que la composition des fréquences d'occurrences des instances formant le nouveau graphe de communication préserve à minima la fréquence d'occurrence de l'application complète.

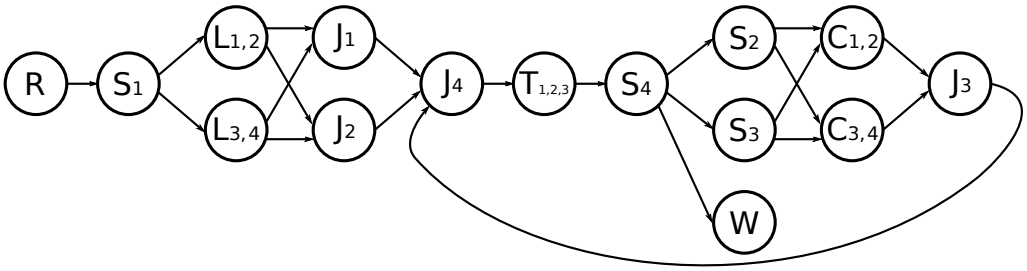
La réduction de parallélisme générique a été implémentée dans la chaîne de compilation Σ-C et son efficacité a été évaluée en utilisant une application de détection de contours dans une image. Cette application basée sur un filtre Laplacien à impulsion Gaussienne (LoG) décompose les images pour appliquer des traitements ligne par ligne puis



(a) Graphe de communication complet instancié avec un degré de parallélisme égal à 4.



(b) Graphe de communication après réduction partielle du parallélisme.



(c) Graphe de communication après réduction complète du parallélisme.

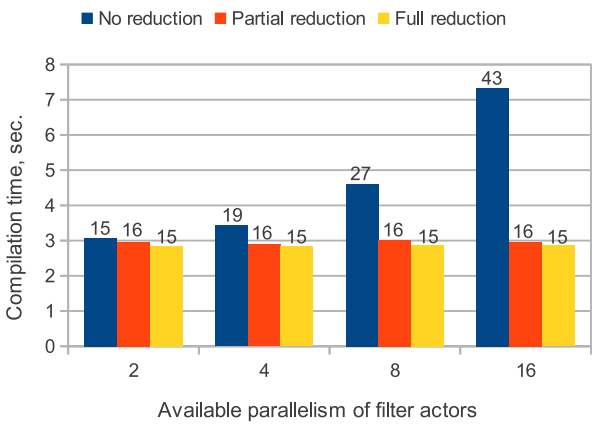
FIGURE 2.12 – Réduction du parallélisme par fusion de tâches et multiplexage des communications appliquée à une application Σ-C implémentant un filtre LoG (*Laplacian of Gaussian*).

colonne par colonne. Le graphe de communication est présenté en figure 2.12a. Des agents systèmes Transpose [2] (T_i) sont utilisés pour effectuer des transpositions entre le traitement des lignes et le traitement des colonnes, puis entre le traitement des colonnes et l'écriture sur la sortie. Deux exemples de réduction sont présentés :

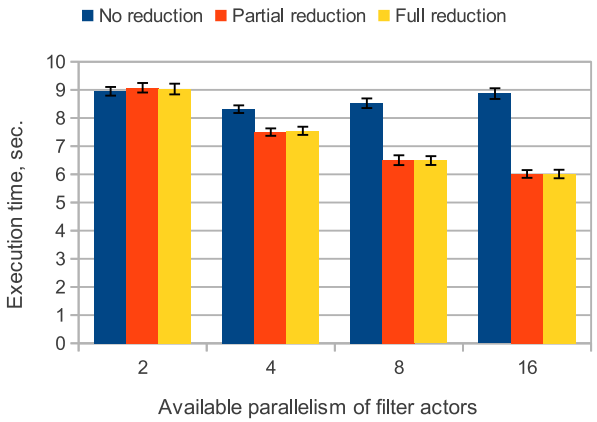
- Une réduction partielle dans laquelle l'opérateur T_3 n'est pas considéré équivalent aux deux autres opérateurs T_1 et T_2. Ces trois instances ne peuvent donc pas être factorisées. Le graphe calculé par le moteur de réduction est présenté dans la figure 2.12b. Les instances utilisateur pour le traitement des lignes (L_i) et des colonnes (C_j) ont été factorisées deux à deux.
- Une réduction complète dans laquelle le *pipeline* central multiplexe aussi la transposition finale avant d'écrire le résultat sur la sortie W. Cette solution peu intuitive montre les possibilités de restructuration du graphe de communication que ne peuvent atteindre les approches implémentées dans Streamit ou par substitution de motifs.

Les résultats en terme de temps de compilation et d'exécution sont présentés dans la figure 2.13. Les principales conclusions sont les suivantes : L'augmentation du degré de parallélisme s'accompagne de la multiplication du nombre d'instances dans le graphe, ce qui implique des temps de compilation plus longs, notamment dans les phases de placement-routage, de dimensionnement des tampons de communication et de génération des objets binaires. C'est ce qui apparaît dans le cas où l'application n'est pas réduite (Figure 2.13a). La compilation du parallélisme permet donc de garder un temps de compilation constant en factorisant suffisamment d'instances. Les temps d'exécution de l'application (ici compilée pour une exécution Posix sur la machine hôte) montre qu'au delà d'un degré de parallélisme de 4 les performances ne progressent plus, minées par une granularité trop fine et des communications trop présentes. À contrario, la factorisation des instances permet d'alimenter les tâches de traitement avec plus de données et de mieux

2. Dans Σ-C la transposition d'une matrice s'écrit comme un Split-Join dont le degré d'ouverture est égal au nombre de colonnes, avec la production élément par élément en sortie du Split et la lecture colonne par colonne sur chaque entrée du Join.



(a) Temps de compilation de la chaîne Σ-C en fonction du degré de parallélisme décidé par le développeur, avant la factorisation des instances.



(b) Temps d'exécution de l'application en fonction du degré de parallélisme décidé par le développeur, avant la factorisation des instances. Les barres d'erreur indiquent la variance liée à la répétition de l'expérimentation (100 itérations)

FIGURE 2.13 – Évaluation du moteur de réduction du parallélisme par factorisation pour l'application LoG écrite avec différents degrés de parallélisme, comparaison entre l'application non réduite, partiellement réduite et complètement réduite.

utiliser la bande passante entre les instances grâce au multiplexage/démultiplexage des flux. Le temps d'exécution diminue sans pour autant augmenter le parallélisme de manière effective.

En conclusion, la réduction du parallélisme est une étape nécessaire dans la chaîne de compilation Σ-C dont l'objectif est de produire des applications correctement dimensionnées. La réduction par substitution de motifs permet de maîtriser le nombre de tâches déployées sur la puce, de limiter l'empreinte mémoire et de s'assurer que l'exécution puisse se réaliser avec une mémoire embarquée contrainte. La réduction par factorisation est une méthode générique qui permet d'adapter le nombre tâches tout en garantissant que les performances calculatoires sont préservées. Ces deux méthodes sont complémentaires et peuvent être appliquées séquentiellement à la compilation en appliquant la substitution puis la factorisation afin de ne pas altérer le graphe une fois que la factorisation ait ajusté les performances en terme de débit. Une approche plus intégrée est cependant souhaitable, afin de ne pas dissocier ces deux phases et de considérer le problème multiobjectifs. Une perspective pour cette étape de compilation est de reposer sur un DSL tel que proposé dans [de Oliveira Castro et al., 2010]. Ce langage haut-niveau permet d'exprimer des réorganisations de données en utilisant des concepts simples comme les itérateurs, plutôt que de construire un graphe complexe avec des agents système, dont le degré de parallélisme est directement exposé au développeur. Ce langage est ensuite compilé en une composition d'agents système, ce qui pourrait être fait de manière à favoriser l'étape de réduction du parallélisme, notamment en privilégiant la construction de motifs et en adaptant en amont les productions et consommations pour satisfaire des contraintes de débit.

2.1.3 Calculer les accès aux tampons partagés

> Cudennec, L., Dubrulle, P., Galea, F., Goubier, T., and Sirdey, R. (2014). Generating code and memory buffers to reorganize data on many-core architectures. In Abramson, D., Lees, M., Krzhizhanovskaya, V. V., Dongarra, J. J., and Sloot, P. M. A., editors, Proceedings of the International Conference on Computational Science, ICCS 2014, Cairns, Queensland, Australia, 10-12 June, 2014, volume 29 of Procedia Computer Science, pages 1123–1133. Elsevier.
>
> [Cudennec et al., 2014] CEA LIST.
>
> Contributions | ■ initiative ■ idées ■ implémentation ■ expérimentations ■ rédaction ■ présentation

Dans la plupart des langages flot-de-données certains acteurs sont dédiés au découpage, à la distribution et à la réorganisation des données. C'est le cas par exemple dans les langages CAL [Eker and Janneck, 2003], GreenStreams [Bartenstein and Liu, 2013], Streamit ou encore τ-C [Goubier et al., 2014]. Dans Σ-C ces acteurs, nommés *agents système* incluent principalement les opérations de distribution (Split), regroupement (Join), duplication (Dup) et de défausse (Sink). Leur interface et leur spécification est fixée par le langage mais il est possible de les paramétrer à l'instanciation dans les sections **map** : par exemple en indiquant le degré d'ouverture d'un Split (le nombre de branches de sortie), la taille de la donnée produite sur chaque branche de sortie ainsi que le type de la donnée, comme dans un *template* C++. La manière dont sont effectivement réorganisées les données est décidée par la chaîne de compilation. Ceci procure

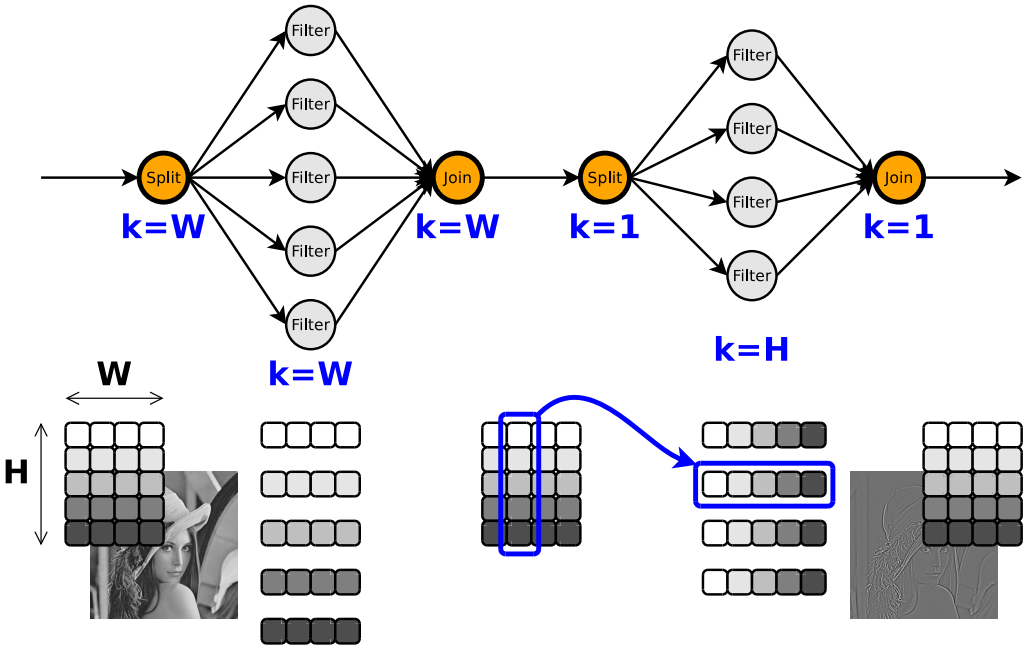


FIGURE 2.14 – (haut) Graphe de communication d'une application de détection de contours composée de deux rideaux de tâches pour traiter les lignes puis les colonnes. La taille de la donnée k consommée lors d'une occurrence est indiquée pour chaque instance d'agent. (bas) Découpage et réorganisation du flot de données par les agents système Split et Join pour alimenter les tâches de traitement avec des lignes et des colonnes.

deux avantages : 1) la mise en œuvre est transparente pour le développeur et 2) la chaîne de compilation peut effectuer des optimisations relativement triviales car ces agents sont clairement identifiés et leur instanciation se conforme à un cadre prédéfini (ce qui n'est pas le cas avec des agents utilisateur, dont l'interface et la spécification sont libres).

Un exemple d'application utilisant des agents système est présenté dans la figure 2.14. Ce graphe de communication lit une image en entrée sous forme d'une suite de pixels, ligne par ligne (cette représentation est courante pour stocker une image ou une matrice en mémoire). Les consommations k indiquées sur les agents Split et Join permettent de distribuer les lignes sur le premier rideau de tâches puis de réorganiser les données pour distribuer les colonnes sur le deuxième rideau de tâches (ce qui est équivalent à une transposition). Une implémentation naïve des agents système consiste à créer un espace mémoire (un tampon partagé) pour chaque lien de communication puis de générer une boucle qui recopie les données entre les tampons partagés en entrée et en sortie de chaque agent en suivant les spécifications. Cette approche génère un grand nombre de tampons partagés, notamment lorsque des agents système sont instanciés avec un grand degré d'ouverture, ce qui est préjudiciable pour l'empreinte mémoire de l'application. La recopie entre les tampons est aussi un processus défavorable car il nécessite du temps processeur et est susceptible de polluer des caches. Une approche plus efficace repose sur la stratégie du zéro-copie, dans laquelle un unique tampon partagé est utilisé par les producteurs et les consommateurs. Pour cela, nous avons introduit le concept de *surface d'agents système.*

Une surface d'agents système est définie comme un sous-graphe sans îlots du graphe de communication tel que tous les nœuds sont des instances d'agents système. La compilation de ces surfaces repose sur les principes suivants : 1) un tampon partagé est créé pour chaque surface, 2) pour chaque lien de communication entrant dans la surface, c'est-à-dire dont l'origine n'est pas un agent système et la destination est un agent système de la surface, un motif d'accès mémoire en lecture sur le tampon partagé de la surface est calculé, et 3) de la même façon un motif d'accès mémoire en écriture sur le tampon partagé est calculé pour tous liens de communication sortant de la surface. A partir des motifs d'accès mémoire ainsi calculés il est possible de déduire la taille minimale du tampon partagé, ce qui répond à la problématique du dimensionnement.

L'équivalence de pointeur (EoP (*Equivalence of Pointer*)) est une propriété garantissant que lors d'une occurrence de tâche, les espaces mémoire mis à disposition de la tâche pour accéder à ses ports de communication sont individuellement contigus. Cette propriété est fondamentale dans Σ-C car le modèle de programmation autorise l'usage de l'arithmétique de pointeurs [3] sur les canaux de communication. Cette propriété permet au développeur d'accéder directement aux données sans passer par une phase de recopie dans des structures de données propres à l'algorithme, comme cela peut être le cas avec certains systèmes de sérialisation des objets.

3. l'arithmétique de pointeur consiste à accéder à des éléments d'un espace mémoire en additionnant des adresses mémoire avec des entiers. Lors d'une allocation, le système garantit que l'espace demandé est réservé dans une zone contiguë en mémoire, et non fragmenté à divers endroits.

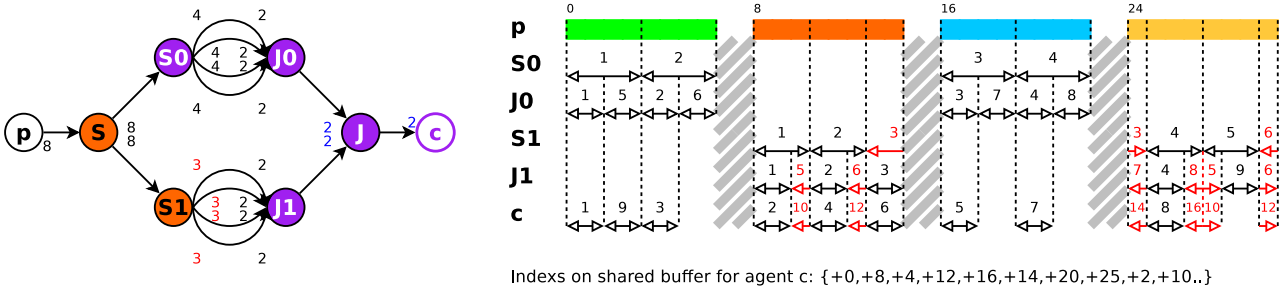


FIGURE 2.15 – Illustration de la non-conservation de l'EoP dans une cascade de Split-*-Join. (gauche) Le graphe de communication avec les consommations et productions. (droite) Motifs d'accès en lecture pour chaque instance d'agent sur le tampon partagé écrit séquentiellement par le producteur p. Pour chaque occurrence de tâche un segment délimité par des flèches $\longleftrightarrow$ est indiqué sur le tampon partagé avec le numéro d'occurrence au dessus. Les segments rouges correspondent à des accès non-contigus qui ne respectent pas l'EoP.

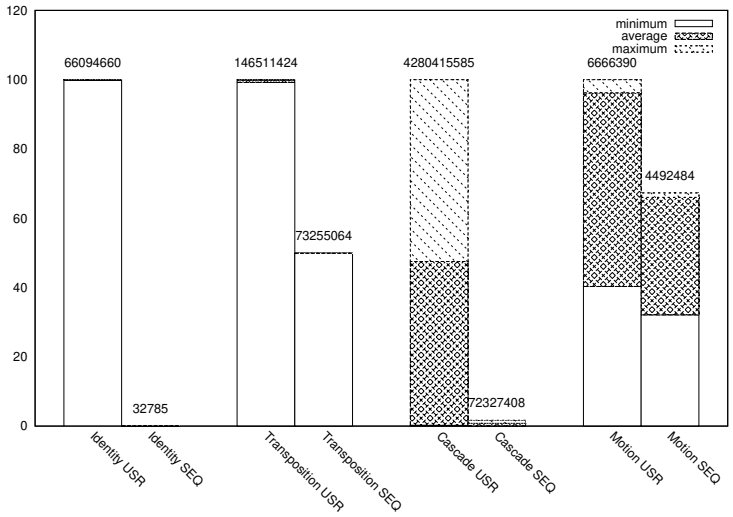


FIGURE 2.16 – Comparaison en nombre de cycles processeur requis pour exécuter différentes applications Σ-C avec la compilation des agents système en mode tampon partagés (SEQ) et en mode utilisateur (USR).

Une illustration de la non-conservation de l'EoP est donnée dans la figure 2.15. Le graphe de communication est constitué d'une cascade Split-*-Join de profondeur 2. La branche S0 conserve l'EoP contrairement à la branche S1. Dans cet exemple, l'EoP n'est pas conservée car les consommations de S1 ne divisent pas les productions de S, tout comme les consommations de J1 ne divisent pas les productions de S1. Il en résulte des accès fragmentés sur le tampon partagé représentés en rouge : l'espace mémoire mis à disposition du code utilisateur ne permet alors pas d'effectuer de l'arithmétique de pointeur, ce qui ne respecte pas le modèle de calcul Σ-C. Ces instances d'agents système ne peuvent donc pas être compilées en un unique tampon partagé : la chaîne de compilation va donc compiler les instances violettes comme un tampon partagé et décider d'un autre mécanisme pour les instances oranges.

Lorsqu'il n'est pas possible de compiler les accès mémoire dans un tampon partagé, un microcode DMA peut est généré pour programmer des motifs d'accès mémoire et reconstituer un espace contigu. Cette solution est par exemple souhaitable lorsque des tâches s'exécutent dans des *clusters* différents et ne partagent donc pas de mémoire physique. Cette approche génère cependant des interactions plus coûteuses qu'un accès mémoire. Elle est de plus limitée par la dimension du DMA, c'est-à-dire la profondeur maximale d'imbrication des boucles itérant sur les adresses mémoire et qui permettent de décrire des motifs plus ou moins complexes (des *strides* pour décomposer un espace mémoire en segments, en blocs, en volumes…). Une autre solution lorsque l'usage du DMA n'est pas possible est de générer un code Σ-C utilisateur pour chaque agent système décrivant les recopies et les réorganisations des données entre les ports de communication en entrée et en sortie. Cette dernière méthode est inefficace car elle ajoute des instances dans l'application, ce qui augmente la complexité des phases de compilation suivantes (partitionnement, placement-routage, édition des liens), augmente l'empreinte mémoire et alimente l'ordonnancement en ligne. Cependant, la compilation des agents système sous forme de code utilisateur permet de traiter les cas les plus complexes et est utilisée comme solution de dernier recours.

La compilation des agents système a été implémentée dans la chaîne de compilation Σ-C. Contrairement à la réduction du parallélisme, c'est une phase non optionnelle du processus. La figure 2.16 illustre l'importance de traduire les réorganisations de données dans des mécanismes bas-niveau efficaces. Différentes applications sont évaluées sur

un ISS fourni par Kalray et simulant fidèlement la puce MPPA-256 (un cycle processeur dans l'ISS correspond à 2,5 ns avec une fréquence d'horloge de 400 MHz). Le nombre de cycles processeur est comparé pour chaque application lors d'une compilation des agents système avec tampons partagés et lors d'une compilation avec des agents utilisateur. Un exemple extrême est donné par le programme synthétique *Identity* composé de deux agents système chaînés produisant sur leur port de sortie les données reçues sur leur port d'entrée : l'exécution est 2000 fois plus lente en mode utilisateur (USR). L'application de détection de mouvement (Motion) est composée de 50 instances d'agents utilisateur embarquant du code de traitement d'image et de 13 instances d'agents système, dont 11 sont potentiellement compilées par des accès à un tampon partagé. Sur cette application de classe industrielle le gain se traduit par une exécution 1,5 fois plus rapide.

En conclusion, La composition d'agents système permet d'exprimer des réorganisations de données complexes, notamment grâce à la grande expressivité offerte par le langage Σ-C lors de la compilation et l'exécution hors-ligne des sections **map**. Cette approche est de plus très intuitive avec un retour visuel pour le développeur sous forme de graphe de communication. Comme bien souvent, offrir un haut niveau d'abstraction au développeur s'accompagne d'une gestion plus complexe du logiciel système pour obtenir de bonnes performances. La compilation des agents système est un problème simple au premier abord et dont la réalisation relève de l'ingénierie. En fait sa résolution a nécessité de proposer de nouveaux concepts et algorithmes pour partitionner les surfaces d'agents systèmes et décider quelle type de compilation doit être appliquée (motifs d'accès à un tampon partagé, génération de microcode DMA ou génération de code Σ-C), ce qui est loin d'être trivial. La compilation des agents système a été transférée à Kalray pour être intégrée dans l'environnement de développement du MPPA-256 [de Dinechin et al., 2013a]. Tout comme la phase de réduction du parallélisme, l'efficacité de cette phase de compilation repose sur la présence de sous-graphes de communication dont la spécification des productions et des consommations permet la compilation en mémoire partagée. Ceci requiert une compréhension fine des techniques de compilation de la part du développeur, ce qui ne peut être exigé et ne favorise pas l'adoption d'un nouveau langage. Une perspective est donc d'apporter des règles de transformation du graphe de communication qui permettent à la compilation de s'assurer que toutes les surfaces d'agents système préservent l'EoP : s'il est difficile de modifier les quantités pour les productions et les consommations sans remettre en cause le bon fonctionnement du code utilisateur, il est toutefois possible de contraindre un agent utilisateur connecté à une surface système d'écrire dans un tampon partagé selon un schéma d'accès précalculé, et non plus séquentiellement, ce schéma d'accès favorisant la préservation de l'EoP pour les agents système connectés en aval. Pour autant, décider s'il existe une transformation du graphe préservant la réorganisation des données spécifiée par le développeur tout en apportant l'EoP relève de la théorie et n'a pas été étudiée ici.

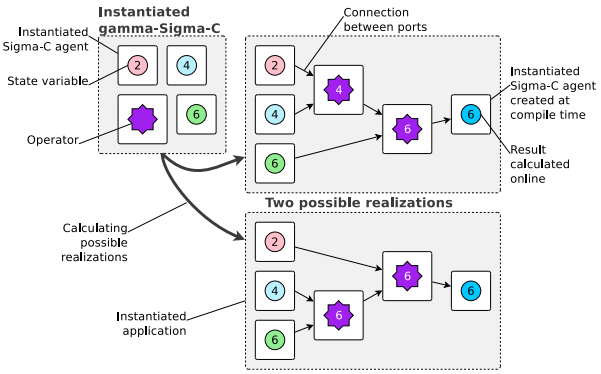
2.1.4 Programmation chimique sur Σ-C

> Cudennec, L. and Goubier, T. (2015). A short overview of executing Γ chemical reactions over the Σ-C and τ-C dataflow programming models. In Koziel, S., Leifsson, L. T., Lees, M., Krzhizhanovskaya, V. V., Dongarra, J. J., and Sloot, P. M. A., editors, Proceedings of the International Conference on Computational Science, ICCS 2015, Computational Science at the Gates of Nature, Reykjavík, Iceland, 1-3 June, 2015, 2014, volume 51 of Procedia Computer Science, pages 1413–1422. Elsevier.
>
> [Cudennec and Goubier, 2015] CEA LIST.
>
> Contributions | ■ initiative ■ idées ■ implémentation ■ expérimentations ■ rédaction ■ présentation

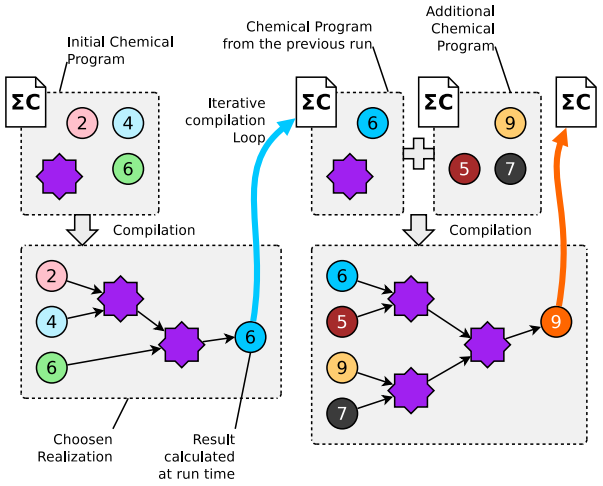
Le principe d'un DSL est de fournir une interface de programmation adaptée à un modèle de programmation particulier. En ce sens Σ-C est un DSL flot-de-données dont le compilateur traduit les concepts d'acteurs et de liens de communication vers un modèle de programmation hybride en mémoire partagée et par passage de messages. Avec Σ-C le développeur doit écrire le code des acteurs et décrire les liens pour construire le graphe de communication. Mais peut-on imaginer un DSL dans lequel cette phase de description du graphe de communication n'est plus requise, et ainsi permettre au développeur de se concentrer uniquement sur le code métier ? Dans un tel système auto-organisant, un ensemble d'agents Σ-C est instancié et la construction du graphe de communication est laissée à la charge du compilateur. Cette approche soulève deux problématiques : 1) un nombre potentiellement important de possibilités à explorer pour construire ce graphe avec la perte du déterminisme à l'exécution et 2) la difficulté d'apporter un cadre de programmation garantissant que quelque soit le graphe de communication retenu à la compilation, le résultat de l'exécution préserve la sémantique exprimée par le développeur (l'application est fonctionnellement correcte). Dans ce travail, j'ai proposé de recourir au modèle de programmation chimique pour apporter ce cadre de programmation haut-niveau.

Le modèle de programmation chimique GAMMA [Banâtre and Métayer, 1990] a été introduit à la fin des années 1980 comme une manière élégante de définir mathématiquement des programmes répartis. Le principe repose sur l'analogie des réactions chimiques, dans lequel un ensemble de molécules réagissent pour en former de nouvelles selon la formule `replace` P `by` M `if` C (avec P et M des molécules et C une condition). La programmation chimique consiste à déclarer des données initiales typées ainsi que des opérateurs. L'ordre des réactions - la manière dont

sont appliqués les opérateurs sur les données - est résolu à l'exécution, sans indications de la part du développeur. Les programmes chimiques sont par nature parallèles et non déterministes. Ce paradigme a été utilisé pour les grappes et grilles de calculateurs et il reste pertinent pour les processeurs many-cœurs. Des exemples d'applications sont proposés dans [Radenac, 2007] et incluent des algorithmes de parcours de graphes, un serveur de messagerie électronique, un système de versionnage de fichiers, du traitement d'image et même un noyau de système d'exploitation [Banâtre et al., 2000]. Une grande part de la complexité des programmes chimiques réside dans le logiciel système qui a la charge d'orchestrer les réactions. L'implémentation de ce logiciel fait resurgir les problèmes classiques d'exécution en environnement réparti pour des applications irrégulières, ce qui explique qu'il est difficile de mettre au point une implémentation efficace du langage chimique. Plusieurs pistes pour une implémentation ont été proposées, de la version séquentielle [Creveuil, 1991] jusqu'à une exécution sous forme d'un flux de travaux (*workflow* en anglais) sur grille de calculateurs [Németh et al., 2006]. L'implémentation que nous avons proposée est basée sur une approche de compilation itérative dans laquelle les prochaines réactions sont décidées à la compilation, le résultat est calculé à l'exécution puis réutilisé lors d'une nouvelle compilation pour déterminer quelles sont les réactions suivantes. Cette approche est en rupture avec une implémentation où les réactions se décident à l'exécution, de manière décentralisée.



(a) Deux graphes de communication possibles à partir d'un même programme chimique Σ-C. Le choix est effectué dans la chaîne de compilation selon des critères imposés par la plateforme cible.



(b) Compilation itérative dans laquelle un résultat intermédiaire est calculé puis réutilisé avec un autre programme chimique Σ-C.

FIGURE 2.17 – Compilation et exécution d'un programme chimique Σ-C appliqué à un opérateur max à deux entrées.

Le calcul chimique peut être comparé à un flot de données, dans lequel les réactions sont représentées par les agents et les molécules par les liens de communications. Réciproquement, une application flot de données peut décrire un ensemble de réactions chimiques appliquées sur des molécules, dans un ordre partiel. Dans cette analogie, écrire un programme chimique en Σ-C consiste à déclarer un état initial composé d'un ensemble d'agents représentant les molécules (un agent avec un unique port de communication typé sur lequel est émis la valeur de la molécule) et un ensemble d'agents représentant les réactions (un agent avec au moins un port en entrée et au moins un port en sortie). Les connexions entre agents ne sont cependant pas décrites dans le programme (absence de section **map**). L'une des particularité de l'approche est de choisir un ordre partiel d'exécution des réactions chimiques lors de la phase de compilation. C'est lors de cette phase que les connexions sont établies afin de former un graphe d'instanciation de l'application, comme présenté dans la figure 2.17a. Lors de la compilation les agents représentant les molécules initiales ne peuvent pas être instanciés plus d'une fois alors que les agents représentant les réactions peuvent être instanciés autant de fois que nécessaire. Pour chaque terminaison du graphe, un agent représentant la molécule produite est instancié (un port de communication en entrée) afin de collecter le résultat du calcul. Ce graphe est ensuite exécuté sur la cible, produisant un ensemble de résultats assimilés aux molécules obtenues. Ces résultats peuvent alors être utilisés comme données initiales d'un nouveau programme chimique Σ-C (Figure 2.17b). Cette compilation rebouclée peut s'alimenter jusqu'à l'obtention d'un état stable, c'est à dire ne permettant plus de construire un nouveau graphe de communication. Les instances d'agent restantes constituent le résidu de la réaction et la solution trouvée à l'issue de l'exécution du programme.

Une implémentation de l'approche itérative pour la programmation chimique a été mise en œuvre dans Σ-C ainsi que dans τ-C [Goubier et al., 2014], une chaîne de compilation similaire à Σ-C, mais avec un objectif expérimental là où Σ-C vise un transfert industriel. La figure 2.18 illustre la modification opérée sur la chaîne de compilation. Elle consiste en 1) l'ajout d'une étape de construction du graphe de communication entre la phase d'instanciation et la phase d'optimisation du déploiement et 2) la collecte des résultats à l'issue de l'exécution suivie de l'écriture d'un nouveau code Σ-C représentant les molécules ainsi obtenues. Cette deuxième modification implémente le rebouclage entre les

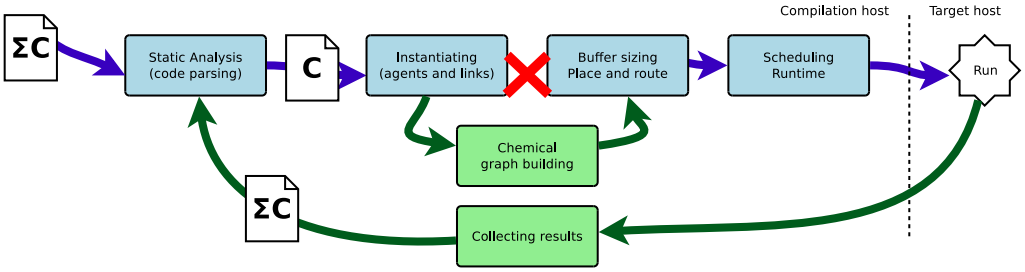


FIGURE 2.18 – Modification de la chaîne de compilation Σ-C pour compiler des programmes chimiques.

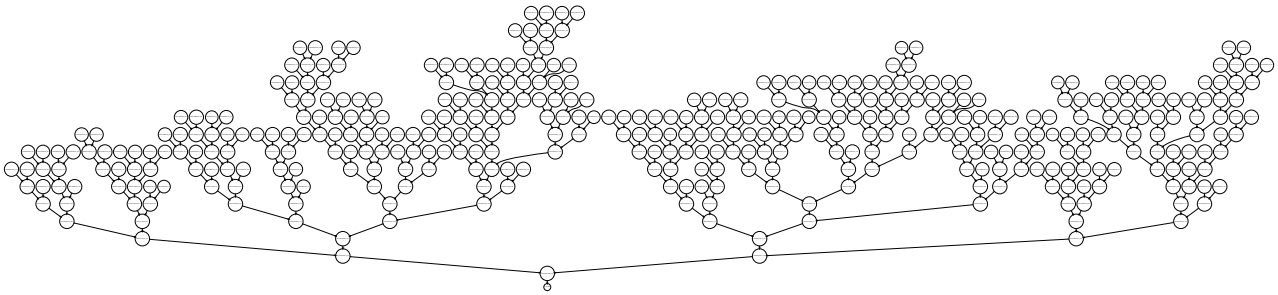
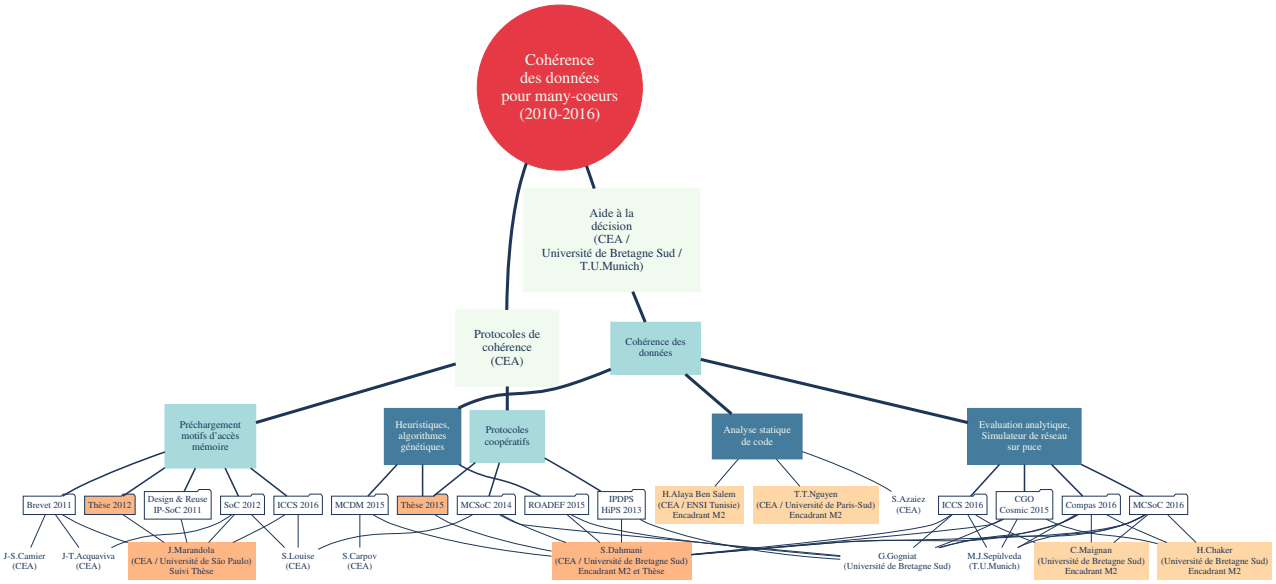


FIGURE 2.19 – Graphe de communication construit à partir d'un opérateur max et de 200 molécules initiales.

cycles de compilation et d'exécution. Un exemple de graphe de communication construit par la chaîne de compilation est donné en figure 2.19 à partir d'un opérateur max et de 200 molécules initiales. Cette application donne lieu à la construction d'un seul graphe du fait de la simplicité de l'opérateur max et de l'unique type de donnée déclaré (un entier). Pour des applications plus complexes, plusieurs graphes de communication indépendants (des îlots) sont créés et exécutés en parallèle sur le processeur many-cœur.

En conclusion, le modèle de programmation flot-de-données apporte un formalisme et une structuration des applications suffisamment haut niveau pour pouvoir y projeter des modèles de programmation abstraits tels que la programmation chimique. Cette approche n'implémente cependant pas toute l'expressivité rendue possible par le modèle GAMMA, notamment la possibilité pour une réaction de créer de nouvelles réactions en sus de nouvelles molécules, comme dans le modèle HOCL [Banâtre et al., 2004]. Si la compilation itérative sur Σ-C est une contribution modeste pour l'implémentation efficace du langage de programmation chimique, celle-ci démontre tout de même les avantages apportés par une structuration claire du code et un modèle de calcul exposés en amont au développeur, associés à une chaîne de compilation modulaire.

Une seconde observation est que les accélérateurs visés par l'approche itérative sont classiquement connectés à un processeur hôte par un bus PCIe (*Peripheral Component Interconnect Express*). Ce type de bus est performant lorsque le programme charge des données en un seul bloc vers l'accélérateur, déporte le calcul puis récupère les résultats en un seul bloc sur le processeur hôte. Dans ces conditions, les temps de transfert bien supérieurs à un accès mémoire sont idéalement masqués par un temps de calcul long sur l'accélérateur. Dans le cas de la compilation itérative d'un programme chimique, le bus PCIe est un facteur limitant car cette utilisation nécessite des échanges fréquents avec une granularité de données fine : le graphe complet ne peut en effet être déterminé en amont et sa construction itérative s'effectue au rythme des résultats partiels calculés sur l'accélérateur. Cependant, les accélérateurs bénéficient aujourd'hui d'une intégration matérielle plus efficace, à travers des bus d'interconnexion pour composants hétérogènes et permettant l'accès à de la mémoire physique partagée ou directement à la mémoire des composants distants, sans recopie. Ces architectures sont donc plus adaptées à un programme dont le comportement tient plus de la collaboration entre un processeur et un accélérateur, plutôt qu'une organisation asymétrique maître-esclave. La compilation itérative de programmes chimiques pourraient donc bénéficier des avancées technologiques présentes sur de tels systèmes.



2.2 Gestion de la cohérence des données sur many-cœur

Les processeurs many-cœurs sont des architectures singulières, à cheval entre les composants électroniques intégrés et les systèmes massivement parallèles et distribués, ou comment concevoir une grille de calculateurs sur quelques centaines de mm^2. Dès lors, les problématiques rencontrées pour gérer les données partagées relèvent de deux principaux domaines déjà bien étudiés : les hiérarchies de caches pour les processeurs et les systèmes de caches répartis pour les architectures distribuées. Ce qui en fait une thématique de recherche à part entière est l'addition des contraintes et des opportunités issues de ces deux domaines bien souvent étudiés par des communautés de chercheurs distinctes. Des architectures processeurs, les many-cœurs héritent des technologies de caches et des réseaux sur puce performants mais qui ne passent cependant pas à l'échelle facilement du fait de la complexité matérielle sous-jacente. Des architectures distribuées, les many-cœurs héritent de l'accumulation des unités de traitement et de stockage, ainsi que des organisations décentralisées pour le passage à l'échelle. Dans ce contexte, l'enjeu est de conserver la vision simple d'un processeur classique tout en exploitant cette accumulation de ressources de manière transparente.

En 2012, lorsque j'initie un sujet de recherche sur la cohérence des données pour les architectures MPPA, les processeurs many-cœurs sont scindés en deux catégories (cf. table 2.1) : la première, représentée par Intel et Tilera propose des mécanismes de cohérence des caches matériels. Avec les architectures TILE, deux NoC sur six sont exclusivement réservés à la gestion des caches pour les messages de contrôle et de transfert de données. Les architectures MIC/Xeon Phi se basent sur un anneau bidirectionnel interconnectant les PE [Rahman, 2013]. Ces approches permettent d'offrir un système programmable avec des outils conventionnels, tels que le modèle *Posix Threads* en mémoire partagée. La taille de ces architectures reste cependant modeste, de l'ordre de quelques dizaines de cœurs. La deuxième catégorie, représentée par Adapteva, Kalray et PEZY n'offre pas de mécanismes de gestion de la cohérence de cache à l'échelle de la puce. La programmation de ces architectures se base sur un modèle distribué ou hybride, bien souvent par l'intermédiaire d'une API propriétaire. Cette approche permet de concevoir des puces avec un grand nombre de cœurs mais au prix d'imposer un modèle de programmation et de forcer le portage ou la réécriture des applications. Les puces de la première catégorie sont adaptées au contexte HPC dans lequel les applications s'exécutent sur des processeurs génériques, alors que les puces de la seconde catégorie visent un marché de niche dans lequel les applications sont développées pour une architecture spécifique. La gestion de la cohérence des caches dans les many-cœurs est donc un enjeu en terme de programmabilité et détermine d'une certaine manière les contextes d'utilisation.

La cohérence des données est un sujet traité dans les architectures parallèles et réparties dès lors que les données sont accédées par des processus concurrents. La question est alors de savoir pour chaque lecture quelle version de la données est retournée, ce qui implique de déterminer quelles modifications sont visibles et dans quel ordre. Pour répondre à cette question, un modèle de cohérence est utilisé, qui peut être vu comme un contrat passé entre le développeur et le système en charge de la cohérence. Ce contrat spécifie formellement l'ordre partiel des modifications appliquées à une donnée partagée pour chaque accès en lecture. Un protocole de cohérence consiste en une implémentation d'un modèle de cohérence, c'est-à-dire une spécification de la séquence des messages de contrôle (des métadonnées) et des données échangées entre les participants lors d'un évènement tel qu'une demande d'accès en lecture ou en écriture. Un protocole de cohérence peut être modélisé à l'aide d'un automate fini réparti.

Il existe des modèles de cohérence plus ou moins relâchés, c'est-à-dire qui établissent un ordre plus ou moins strict sur les évènements. Nous pouvons citer quelques exemples : le modèle de cohérence le plus contraignant est le modèle strict *atomic consistency*, un modèle idéal où chaque lecture rend la dernière valeur écrite dans la donnée. Dans les systèmes distribués, un tel modèle nécessite l'utilisation d'une horloge globale, ce qui rend son implémentation impossible. Le modèle séquentiel *sequential consistency* [Lamport, 1979] assure que les évènements sont vus dans le même ordre sur chaque participant. Le modèle causal (*causal consistency*) [Lamport, 1978] se base sur la notion de causalité pour déterminer l'ordre entre deux évènements. Ce modèle permet de lier certains évènements entre eux par un ordre bien fondé tout en relâchant les contraintes sur les évènements indépendants. Les protocoles de cohérence implémentant ces modèles de cohérence génèrent un nombre important de messages, parfois inutiles lorsqu'il s'agit de mettre à jour des données qui ne seront plus accédées par la suite. Ces implémentations sont peu adaptées pour passer à l'échelle dans des grands systèmes distribués. D'autres modèles dits *relâchés* permettent de diminuer la pression sur le protocole de cohérence et de favoriser le passage à l'échelle. Certains modèles peuvent autoriser des écritures et des lectures concurrentes, et le développeur de l'application doit accepter que la donnée lue n'est pas nécessairement la plus récente à un *instant t*. Un exemple de modèle de cohérence utilisé dans les travaux présentés dans ce manuscrit est la cohérence à l'entrée (*entry consistency*) [Bershad et al., 1993]. Dans ce modèle un verrou est associé à chaque donnée partagée, il permet de protéger les accès au sein d'une portée délimitée par l'acquisition du verrou et sa libération. En dehors de cette portée la donnée est considérée comme indéterminée. Ce modèle limite le faux-partage en discriminant les accès sur une donnée partagée et non pas un espace mémoire complet. Il permet de plus de faire varier la granularité des données partagées pour optimiser le comportement du protocole de cohérence.

> Choisir un modèle de cohérence consiste donc à trouver le bon compromis entre le maintient d'une cohérence suffisamment stricte requise par l'application tout en déployant un protocole de cohérence adapté à l'architecture cible en terme de performance calculatoire ou énergétique.

Dans ce manuscrit nous ne détaillons pas les différents modèles et protocoles de cohérence des données, ces informations pouvant par exemple être trouvées dans le cadre de ma thèse [Cudennec, 2009]. J'ai étudié ce sujet dès 2004 [Cudennec, 2005], notamment en proposant un protocole de cohérence spécialisé dans la visualisation de données partagées pour du couplage de code [Antoniu et al., 2006a]. L'évaluation de ce protocole fait partie des toutes premières expérimentations multi-sites menées sur la grille expérimentale GRID'5000 [Antoniu et al., 2006b]. Dans ce document, les travaux effectués sur la thématique de la cohérence des données pour les architectures many-cœurs sont classés en trois axes. Un premier axe concerne l'étude de nouveaux protocoles de cohérence basés sur le préchargement de motifs d'accès mémoire et sur la coopération de caches voisins. Un deuxième axe concerne l'évaluation de ces protocoles de cohérence à l'aide de méthodes analytiques et de simulateurs de NoC. Enfin, un troisième axe traite de l'aide à la décision pour le choix d'un protocole de cohérence adapté à une application et un contexte d'exécution.

2.2.1 Protocoles de cohérence de caches coopératifs

> Marandola, J., Louise, S., Cudennec, L., Acquaviva, J., and Bader, D. A. (2012). Enhancing cache coherent architectures with access patterns for embedded manycore systems. In 2012 International Symposium on System on Chip, ISSoC 2012, Tampere, Finland, October 10-12, 2012, pages 1–7. IEEE.
>
> [Marandola et al., 2012] CEA LIST, Georgia Institute of Technology.
>
> Contributions | ★ initiative ■ idées ☐ implémentation ☐ expérimentations ■ rédaction ☐ présentation

> Dahmani, S., Cudennec, L., and Gogniat, G. (2013). Introducing a data sliding mechanism for cooperative caching in manycore architectures. In 2013 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum, Cambridge, MA, USA, May 20-24, 2013, pages 335–344. IEEE.
>
> [Dahmani et al., 2013] CEA LIST, Lab-STICC.
>
> Contributions | ★ initiative ■ idées ☐ implémentation ☐ expérimentations ■ rédaction ☐ présentation

> Dahmani, S., Cudennec, L., Louise, S., and Gogniat, G. (2014). Using the spring physical model to extend a cooperative caching protocol for many-core processors. In IEEE 8th International Symposium on Embedded Multicore/Manycore SoCs, MCSoC 2014, Aizu-Wakamatsu, Japan, September 23-25, 2014, pages 303–310. IEEE Computer Society.
>
> [Dahmani et al., 2014] CEA LIST, Lab-STICC.
>
> Contributions | ★ initiative ■ idées ☐ implémentation ☐ expérimentations ■ rédaction ☐ présentation

Proposer un protocole de cohérence pour un processeur many-cœur nécessite de prendre en compte les spécificités de ces architectures pour assurer un passage à l'échelle dans de bonnes conditions tout en offrant une implémentation

Protocole	Constructeur	Références	États	Premières cibles	Année
Hammer HyperTransport	AMD	[Keltcher et al., 2002]	MOESI	Opteron, Athlon64	2002
QuickPath Interconnect	Intel	[Intel, 2009]	MESIF	Core Nehalem	2008
AMBA 5 CHI ACE	ARM	[ARM, 2017]	(MO)ESI	Cortex A	2013
Infinity Data Fabric	AMD	[Lepak et al., 2017]	MDOEFSI	EPYC	2017
Ultra Path Interconnect	Intel	[Mulnix et al., 2017]	MESIF	Xeon Skylake	2017
TAG Directory	Fujitsu	[Fujitsu Limited, 2020]	MESI	A64FX	2019

TABLE 2.2 – Protocoles de cohérence des caches et bus d'interconnexion pour quelques architectures processeur.

réaliste sur le plan matériel et logiciel. Les approches classiques sont héritées des protocoles déployés sur les processeurs multi-cœurs. Elles reposent sur le principe du répertoire [Culler et al., 1999] avec une description de l'état des données partagées. Dans la table 2.2 plusieurs protocoles de cohérence de cache sont présentés pour des architectures multi-cœurs très largement répandues ciblant des domaines d'application grand public, HPC et embarqué. Ces protocoles reposent sur des briques matérielles spécifiques, comme des bus d'interconnexion ou des NoC propriétaires qui rendent les solutions non portables et non applicables pour d'autres architectures (à l'exception de l'approche ARM qui permet d'accéder aux spécifications et d'en effectuer une implémentation).

Le répertoire (*directory-based protocol*) est un espace regroupant des métadonnées sur l'état global ou partiel du système de caches. Ce répertoire peut être centralisé ou décentralisé, c'est-à-dire distribué sur plusieurs unités de traitement de la puce. Les informations ainsi stockées concernent notamment l'état courant des données partagées ainsi que leur présence ou non sur les différents espaces mémoire. Ces informations sont utilisées par le protocole de cohérence pour localiser les données, autoriser ou non un accès ou encore commander un transfert. Lorsque le répertoire est distribué, une méthode courante pour connaître quelle unité contacter lors d'un accès à une donnée est d'utiliser la notion de nœud référent (HN (*Home-Node*)). Pour chaque donnée partagée, un HN est associé suivant une politique connue par tous les nœuds. Pour connaître le HN, une fonction de hachage prend un identifiant de donnée partagée et retourne l'identifiant du nœud référent. Une fonction classique (mais naïve) consiste à effectuer un modulo sur les identifiants des nœuds, ce qui revient à effectuer une assignation par la méthode du tourniquet (*round-robin*).

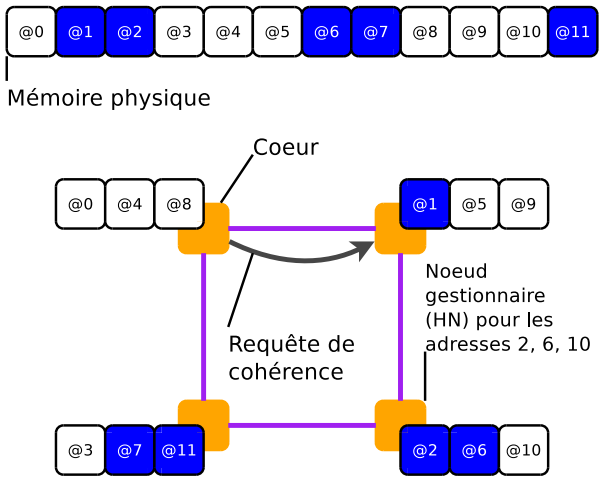
L'état d'une donnée partagée dépend des accès en cours et/ou des derniers accès effectués. Le nombre et le type des états dépend directement du protocole de cohérence implémenté. Les 4 états les plus courants sont MESI, correspondant à Modifié (un processus a modifié la donnée et les copies ne sont plus à jour), Exclusif (un processus est en cours d'écriture et les accès suivants doivent attendre la fin de cet accès), Shared (un processus est en cours de lecture et des accès concurrents en lecture seule sont possibles) et Invalidée (la donnée est une copie et nécessite une mise à jour). Dans cet exemple, MESI est associé à un protocole permettant des lectures concurrentes et des écritures exclusives (MRSW (*Multiple Readers, Single Writer*)). D'autres états ont été proposés pour optimiser la gestion des données, par exemple l'état Forward du protocole MESIF permet de désigner le cache ayant effectué le dernier accès en lecture comme détenteur de la donnée plutôt que de la chercher dans la mémoire physique. L'état Owner du protocole MOESI est équivalent à l'état Forward de MESIF, à l'exception du fait que chaque modification donne lieu à une mise à jour de la mémoire physique, ce qui peut pénaliser l'application dans certains cas.

La coopération des caches consiste à mutualiser les ressources matérielles, par exemple en stockant des données sur des caches distants lorsque le cache local est saturé, ou encore en profitant d'une bande passante agrégée lors d'un accès à une donnée présente sur plusieurs caches. L'intérêt pour la mise en place d'une coopération des caches grandit à mesure que la taille de la puce augmente : les problématiques classiques rencontrées avec le passage à l'échelle nécessite la mise en place de stratégies de groupe et l'équilibrage de charge sur les ressources. Cet aspect n'est pas développé dans les processeurs multi-cœurs traditionnels du fait du nombre modeste de cœurs. Dans les architectures many-cœurs dotées de mécanismes de cohérence, l'organisation des caches est calquée sur les grandes machines NUMA, avec une structuration hiérarchique qui ne favorise pas des collaborations directes, notamment au voisinage. C'est la raison pour laquelle nous avons exploré différentes stratégies de collaboration : un mécanisme logiciel-matériel permettant la gestion répartie de motifs d'accès mémoire et des mécanismes logiciels ou matériels permettant l'utilisation de caches voisins par glissement de données.

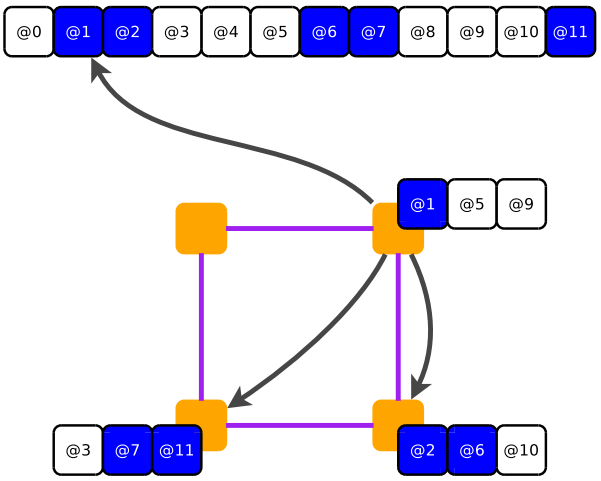
Le projet CoCCA (*Co-designed Coherent Cache Architecture*) [Marandola et al., 2012] propose d'étendre un protocole de cohérence de caches classique avec une unité de traitement matérielle de motifs d'accès mémoire pour effectuer du préchargement de données. Cette unité de traitement située sur chaque cœur, permet d'intercepter des accès à des adresses identifiées préalablement puis de déclencher une série d'accès correspondant à un motif mémoire afin de remplir le cache par anticipation. L'unité comprend une table associative, chaque entrée étant composée d'un déclencheur (*trigger* en anglais) et de la description du motif sous forme compacte. Une version simple du déclencheur consiste à

indiquer la première adresse mémoire du motif. La forme compacte d'un motif consiste en un quadruplet (adresse de base, taille, décalage, longueur) permettant de décrire un motif 2D.

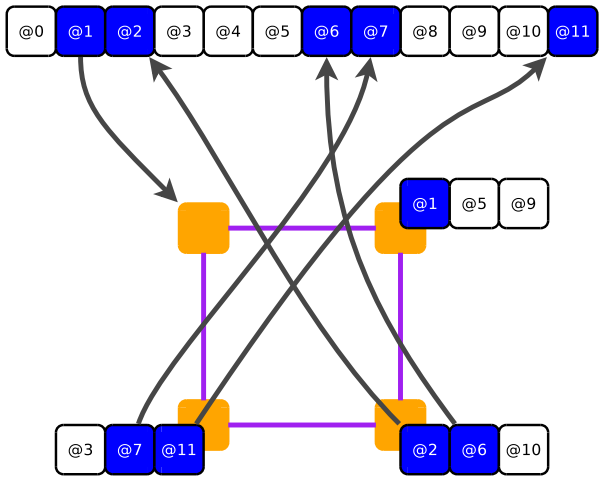
Les motifs d'accès mémoire sont déterminés hors-ligne et construits de différentes manières : 1) en se basant sur une analyse de code, et plus précisément les boucles imbriquées, 2) en effectuant une exécution du binaire sur le processeur hôte, instrumenté par l'outil de compilation JIT (*Just in Time*) Pin/Pintools [Bach et al., 2010] et en analysant les accès mémoire dans les traces d'exécution ou 3) en acceptant directement des motifs décrits par l'application. La figure 2.20 illustre le comportement du protocole CoCCA lors d'un accès à un motif connu. Une simple requête à une adresse déclenche le transfert de plusieurs données de la mémoire principale vers le cache du cœur émetteur. Cette version du protocole peut encore être optimisée en fusionnant les deux premières étapes présentées en figures 2.20a et 2.20b.



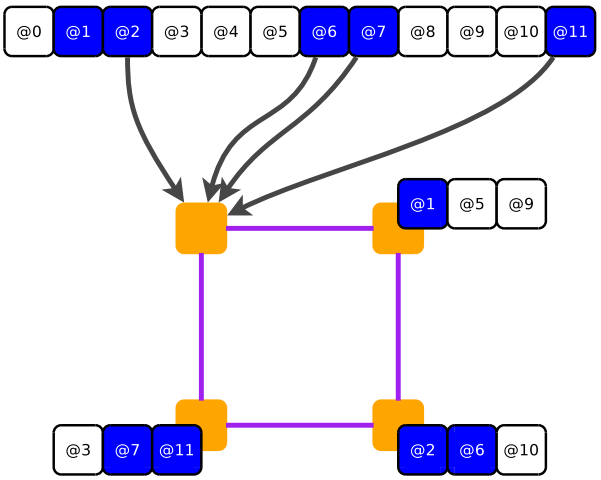
(a) Représentation de la mémoire physique (en haut) contenant les adresses numérotées de 0 à 11. Le motif 2D sélectionné apparaît en bleu (adresse de base @1, taille 2, décalage 3, longueur 3). Représentation de la répartition des HN sur les 4 cœurs (en bas). Les données ne sont pas nécessairement stockées dans les caches des coeurs HN.



(b) Le HN responsable de l'adresse @1 reçoit une requête d'accès. Cette adresse est associée dans sa table CoCCA à un motif 2D. Un ordre de transfert est émis en mémoire physique pour la première adresse. Les HN correspondants aux adresses du motif déclenché sont contactés en parallèle pour anticiper les accès suivants.



(c) Le nœud à l'origine de la requête sur l'adresse @1 reçoit la donnée demandée (comportement classique du protocole de cohérence MESI). Les HN contactés de manière spéculative émettent un ordre de transfert en mémoire physique pour les adresses du motif.



(d) Le nœud à l'origine de la requête initiale reçoit dans son cache les données correspondant aux motif reconnu, limitant ainsi les défauts de cache pour les accès suivants.

FIGURE 2.20 – Fonctionnement du protocole CoCCA sur un processeur à 4 cœurs pour un accès à un motif 2D.

Des évaluations analytiques du protocole ont été menées dans [Marandola et al., 2012] et [Marandola et al., 2016]. La méthodologie consiste à instrumenter une application avec l'outil *pinatrace* de Pin/Pintools qui permet de journaliser les accès aux données partagées pour chaque cœur. Une machine à état permet ensuite, en fonction d'un protocole de cohérence et d'une architecture donnés, de reconstituer l'état du cache et les messages de cohérence échangés. La table 2.3 donne le nombre de messages de cohérence générés par un filtre convolutif en cascade sur une architecture composée d'une grille de 7×7 cœurs. Le protocole CoCCA réduit le nombre de messages de cohérence de l'ordre de 37%. Par ailleurs, le préchargement des données dans les caches permet de réduire le temps de latence à la mémoire. Pour

Messages de CC ↓ Protocole →	MESI	CoCCA
Invalidation de ligne de cache partagée	34560	17283
Accès exclusif à une ligne de cache partagée entre 2 cœurs	12768	12768
Accès exclusif à une ligne de cache partagée entre 4 cœurs	1344	772
Total	48672	30723

TABLE 2.3 – Nombre de messages de CC émis lors de l'exécution d'un filtre convolutif en cascade sur une image de taille 640×480, sur une architecture composée d'une grille de 7×7 cœurs et pour les protocoles MESI et CoCCA. L'image est découpée en 49 rectangles de tailles identiques, chaque cœur travaille sur un rectangle. Une zone de recouvrement (*shadowing* en anglais) est présente sur les bords : certaines données sont donc partagées entre 2 ou 4 cœurs.

obtenir un ordre de grandeur, nous avons mesuré le nombre de cycles nécessaire à l'exécution du filtre convolutif sur un unique cœur d'un processeur Intel Xeon Nehalem, ce qui s'apparente à l'utilisation du protocole CoCCA lorsque tous les motifs sont préchargés dans les caches. La valeur obtenue est comparée avec une exécution utilisant le protocole par défaut, ce qui donne un temps d'exécution 67% plus lent pour MESI par rapport à CoCCA (le détail du calcul se trouve dans [Marandola et al., 2012]).

J'ai pris la direction du projet interne CoCCA sur financement CEA Challenge Innovation de 2010 à 2012. Les résultats ont donné lieu à la soutenance de thèse de Jussara Marândola Kofuji [Marândola Kofuji, 2011], à un dépôt de brevet [Cudennec et al., 2012] et à trois publications internationales. En 2019, le processeur Fujitsu A64FX équipant la machine Fugaku (RIKEN, Japon) classée première du TOP500 depuis juin 2020, intègre un circuit de préchargement de données en cache basé sur les motifs d'accès mémoire [Okazaki et al., 2020]. Ce circuit effectue du *prefetch* matériel soit par détection automatique d'une séquence d'adresses en ligne, soit par une méthode appelée *injection* qui prend en charge des motifs d'accès par décalage (*strided access patterns* en anglais) directement déclarés par le programme. L'intégration de tels mécanismes dans le processeur A64FX démontre la pertinence de l'approche proposée dans le projet CoCCA une dizaine d'années plus tôt.

Le protocole coopératif par glissement de données (*data-sliding protocol*) [Dahmani et al., 2013] est un protocole de cohérence mettant à contribution les caches voisins pour soulager l'activité d'un cache en particulier. La coopération entre caches n'est pas une nouveauté en soi : c'est une technique éprouvée dans de nombreux domaines, par exemple pour les serveurs internet [Holmedahl et al., 1998] ou les réseaux sans-fil non-structurés [Cho et al., 2003]. Dans le domaine des architectures multiprocesseurs, une approche de cache coopératif est proposée dans [Dybdahl and Stenström, 2007]. Dans cette approche, le cache de chaque cœur est segmenté en deux parties : une partie privée et une partie partagée dans laquelle d'autres cœurs peuvent stocker des données. Une extension est proposée sous forme de cache élastique [Herrero et al., 2010] autorisant la segmentation dynamique des caches, ce qui permet de s'adapter aux besoins opérationnels. Cependant, dans le cas d'un ensemble de caches voisins fortement sollicités, ces protocoles ne permettent pas de répondre à la demande de coopération et génèrent des évictions qui se traduisent ensuite par des défauts de cache.

Sur une puce composée d'un grand nombre de caches locaux comme cela est le cas pour certains processeurs many-cœurs, une telle zone d'activité (*hot spot* en anglais) entraîne un déséquilibre entre les caches voisins fortement sollicités et les caches distants potentiellement sous-utilisés. Une approche naïve consiste à mettre à contribution des caches de plus en plus éloignés. C'est le principe du *N-Chance Forwarding*, un équivalent de la *patate chaude* qui repousse une donnée aussi loin que nécessaire. Cette stratégie s'accompagne cependant d'une augmentation de la latence pour accéder aux données à travers le NoC. Et ce NoC peut s'avérer être complexe (par exemple le tore 2D entrelacé de la puce Kalray) et induire une latence plus élevée pour atteindre certains caches distants qu'un accès à la mémoire externe ! Pour répondre à cette problématique nous avons proposé un mécanisme de glissement de données dont le principe est le suivant : A) lorsqu'un cache N_1 est saturé, celui-ci effectue une demande de coopération à un cache voisin N_2. B) Si le cache N_2 est saturé, celui-ci choisit une ligne dans son cache privé à migrer et effectue une demande de coopération à un cache voisin N_3 afin de libérer cet espace pour y stocker la donnée du cache N_1. Le mécanisme opère donc un glissement plutôt qu'un transfert, en poussant les données autour du cache saturé. De cette manière, les données restent statistiquement plus proches de leur cache propriétaire.

La figure 2.21 illustre le mécanisme de glissement de données entre 3 caches. Sur tous les caches, un espace libre flottant est maintenu afin d'honorer les requêtes de coopération en garantissant une latence correspondant à un hop (au plus loin un voisin direct). En l'absence d'une telle stratégie, et parce que cette primitive est synchrone, le cache à l'origine de la requête de coopération serait susceptible d'attendre la terminaison de toute la chaîne de coopération avant de considérer que la donnée est bien stockée. Le glissement de données peut être implémenté comme une extension d'un protocole de cohérence classique. Lors d'un accès à une donnée, le protocole classique consulte le cache local. En cas d'absence de celle-ci, le protocole coopératif consulte les caches distants. Enfin, en dernier recours, le protocole classique effectue les opérations suite à défaut de cache, qui consistent à contacter le HN et/ou déclencher un accès en mémoire externe.

Une évaluation analytique du protocole de glissement de données a été menée et des comparaisons ont été effectuées

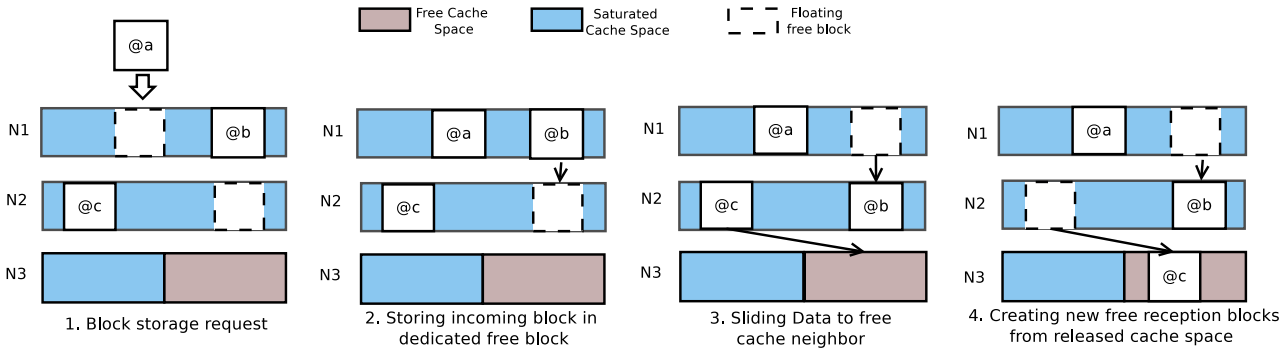
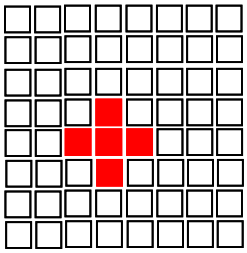
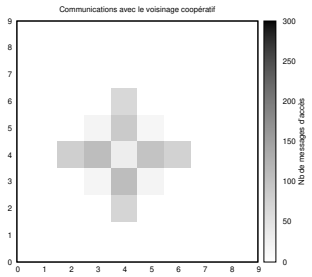


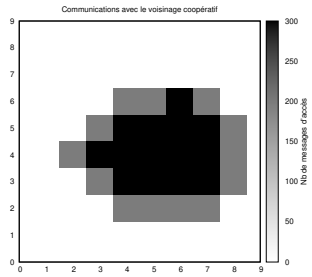
FIGURE 2.21 – Protocole de glissement de données entre 3 caches N_1, N_2 et N_3.



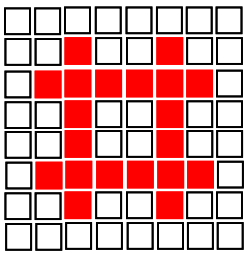
(a) Scénario à un point chaud.



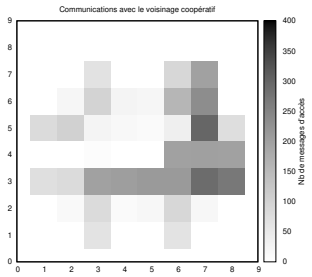
(b) MESI + Cache élastique.



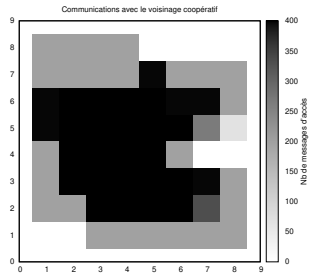
(c) MESI + Glissement de données.



(d) Scénario à 4 points chauds.



(e) MESI + Cache élastique.



(f) MESI + Glissement de données.

FIGURE 2.22 – Nombre de messages de CC émis sur chaque cœur d'une grille 8×8, pour des scénarios à 1 et 4 points chauds. Comparaison des résultats analytiques obtenus pour le protocole de cache élastique [Herrero et al., 2010] et le protocole de cache par glissement de données. Ces figures sont issues du manuscrit de thèse de Safae Dahmani [Dahmani, 2015]

avec le protocole de cache élastique. La figure 2.22 représente sous forme d'une carte de chaleur (*heatmap* en anglais) le nombre de messages de CC émis par chaque cœur. L'architecture choisie est une grille de 8×8 cœurs interconnectés par un NoC à 4 voisins. Les scénarios sont constitués de points chauds, c'est à dire qu'une application synthétique effectue un nombre important d'accès à des données partagées sur un cœur et ses voisins afin de rapidement saturer puis polluer les caches locaux. En comparaison du protocole élastique, le glissement de données permet un étalement plus important de la surface de coopération ainsi qu'une augmentation du nombre de messages de CC. Cette augmentation du nombre de messages est cependant à mettre en perspective avec le gain obtenu en terme de défaut de caches. Dans cet exemple chaque utilisation d'un nœud voisin correspond à une donnée conservée dans les caches de la puce, et non évincée en mémoire externe. Un autre paramètre à prendre en compte est la distance un nombre de hops entre un nœud effectuant une requête d'accès et le cache détenant la donnée. Bien que conçu pour limiter globalement cette distance, le protocole par glissement peut entraîner un éloignement important si le mécanisme est appliqué de manière itérative dans la même direction. Pour résoudre ce problème, une extension du protocole de glissement inspirée du modèle masse-ressort a été proposée pour limiter l'éloignement des données sur la puce.

Le protocole de glissement avec modèle masse-ressort (mass-spring protocol) [Dahmani et al., 2014] permet d'adapter dynamiquement la distance d'éloignement d'une donnée du cœur propriétaire. Cette distance d'éloignement est un élément paramétrable du protocole de glissement de données, sur le principe du *N-Chance Forwarding*, indiquant un nombre maximum de hops possibles au delà duquel la donnée n'est plus conservée sur la puce. Conserver la donnée sur la puce reste avantageux dans de nombreuses situations : la figure 2.23a montre que le taux de succès au cache augmente à mesure que la distance maximale d'éloignement autorisée augmente, jusqu'à un certain stade à

partir duquel la surface constituée par les caches coopératifs est devenue suffisante. Ce paramètre d'éloignement n'est cependant pas complet car les migrations de données ne s'effectuent pas nécessairement dans la direction opposée au cœur propriétaire : si un glissement rapproche la donnée, il sera tout de même considéré comme un éloignement au sens du nombre de hops. Un point crucial porte donc sur le choix du voisin lors d'une demande de coopération afin de ne pas systématiquement migrer dans la même direction. Dans le protocole original, le meilleur voisin est déterminé à l'aide de compteurs locaux incrémentés à chaque requête de glissement. Cette technique permet de classer les voisins par niveau de sollicitation, tout en évitant une implémentation centralisée ayant une vue globale de l'état des caches de la puce. Le choix du cœur voisin sur lequel migrer la donnée n'est donc pas dépendant de la localisation du cœur propriétaire, et aucun mécanisme ne permet de limiter la distance physique.

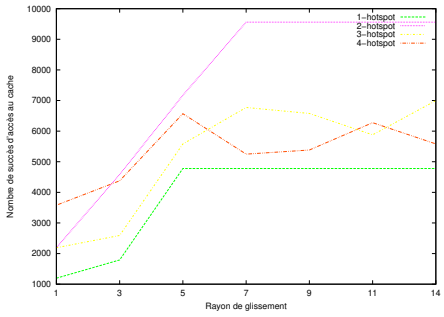
Le principe de l'extension de protocole masse-ressort est de s'inspirer du modèle physique du même nom pour décider sur quel voisin migrer les données (figure 2.23b). Dans cette analogie, le processeur est vu comme un bac-à-sable dans lequel les caches forment des creux et des bosses en fonction de leur niveau de saturation : un creux pour un cache sous-utilisé et une bosse pour un cache sur-utilisé. Les données partagées sont représentées par des billes de masse constante (en fait la masse peut être ajustée pour varier l'inertie et donc la propension à changer de direction sur la puce). Enfin, le modèle comprend un ressort dont les extrémités sont connectées au cœur propriétaire d'une part et à la donnée d'autre part. Afin d'éliminer les effets de résonance et donc éviter qu'une donnée passe son temps à osciller entre deux caches, le modèle intègre des frottements visqueux. Une modélisation mathématique donne l'équation suivante lorsque projeté sur l'axe x avec Δh la différence de potentiel entre le cœur source et le cœur destination, m la masse de la donnée, k la constante de raideur du ressort et c la constante de viscosité.

$$m\frac{d^2x}{dt^2} = \alpha mg\Delta h - c\frac{dx}{dt} - kx$$

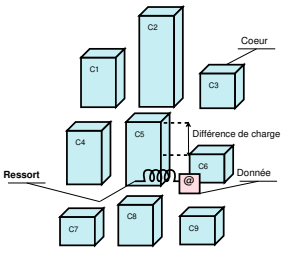
Après application d'un changement de variable, d'une simplification et d'une discrétisation temporelle (transformations décrites dans [Dahmani et al., 2014]) nous obtenons l'équation suivante.

$$D_i = 2AD_{i-1} - A^2D_{i-2} + B\Delta h$$

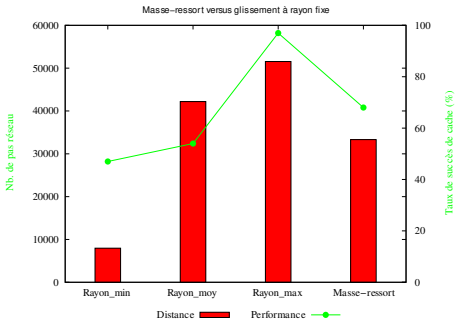
Cette équation détermine la distance D_i entre la donnée et le cœur propriétaire, avec A et B des constantes intégrant les différentes constantes de raideur du ressort, de viscosité et de masse. Elle offre par ailleurs de bonnes propriétés pour une implémentation matérielle : 1) le résultat est dépendant des deux précédentes distances calculées, ce qui n'implique donc pas de conserver un historique de grande taille pour chaque donnée, 2) les opérations mathématiques sont simples et peuvent faire l'objet d'un traitement dans une petite unité matérielle dédiée. Cette équation a été implémentée et évaluée pour être comparée au protocole de glissement de données par défaut, avec des distances de glissement fixées au minimum (1 hop), à la valeur moyenne et au maximum (la taille de la puce moins 2 hops, c'est-à-dire 14 hops). Les résultats sont présentés dans la figure 2.23c et montrent que le protocole masse-ressort trouve un compromis entre les solutions extrêmes et domine la solution consistant à prendre un rayon moyen (la moitié de la puce) en offrant un meilleur taux de succès dans les caches tout en générant moins de *hops* cumulés.



(a) Nombre de succès d'accès au cache (*cache hit*) en fonction de la distance maximale autorisée entre le cœur propriétaire et la donnée (le rayon) lors de l'utilisation du protocole de glissement de données avec différents scénarios applicatifs.



(b) Analogie entre la gestion des données dans un protocole de cache coopératif et le modèle physique masse-ressort.



(c) Nombre de *hops* cumulés et pourcentage de succès d'accès au cache (*cache hit rate*) pour le protocole de glissement de données à rayon fixe et le protocole masse-ressort.

FIGURE 2.23 – Principe et évaluation d'un protocole de cohérence basé sur le modèle physique masse-ressort. Ces figures sont issues du manuscrit de thèse de Safae Dahmani [Dahmani, 2015]

En conclusion, les processeurs many-cœurs offrent de nouvelles possibilités pour répondre efficacement à la problématique de la cohérence de cache pour les architectures massivement parallèles. Pour les grandes puces, la tendance est de favoriser une organisation plate ou faiblement hiérarchique (un niveau) dans laquelle les PE se voient attribuer

un petit cache local, notamment pour limiter la surface de la circuiterie et contenir le facteur de forme de la puce. Dès lors, pour compenser ce faible espace de stockage local, plusieurs pistes sont explorées : 1) reposer sur un NoC haute-performance permettant d'accéder à la mémoire externe sans faire chuter les performances calculatoires. Cette stratégie peut faire usage de nouvelles technologies telles que l'empilement 3D (*3D stacking* en anglais) de couches contenant des circuits pour les PE et la mémoire, comme c'est le cas dans la puce INTACT [Vivet et al., 2020]. Les communications verticales permettent alors de diminuer la distance pour accéder aux données. 2) S'inspirer des techniques de coopération et de communication de groupes pour construire un espace de stockage logique tirant partie des caches voisins. Une piste d'étude consiste à déployer des espaces logiques auto-organisants qui s'étendent et se rétractent en fonction des besoins en mémoire exprimés par les processus. Une telle organisation s'apparente à une MVP et rejoint les thématiques de recherche décrites dans le chapitre 3. Enfin 3) offrir des outils à même d'anticiper les accès et optimiser le contenu du cache. Pour cela, nous pouvons imaginer utiliser des algorithmes de recherche opérationnelle, des heuristiques et de l'apprentissage machine en ligne pour détecter des motifs d'accès au cours de l'exécution de l'application et ainsi déclencher des mécanismes de préchargement.

2.2.2 Évaluation analytique et simulateurs de réseaux sur puce

> Chaker, H., Cudennec, L., Dahmani, S., Gogniat, G., and Sepúlveda, M. J. (2015). Cycle-based model to evaluate consistency protocols within a multi-protocol compilation tool-chain. In Wang, Z., Petoumenos, P., and Leather, H., editors, Proceedings of the 2015 International Workshop on Code Optimisation for Multi and Many Cores, COSMIC@CGO 2015, San Francisco Bay Area, CA, USA, February 8, 2015, pages 8 :1–8 :10. ACM.
>
> [Chaker et al., 2015] Lab-STICC, CEA LIST, École Centrale de Lyon.
>
> Contributions | ★ initiative ■ idées □ implémentation □ expérimentations ■ rédaction □ présentation

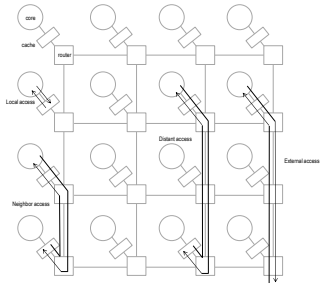
> Cudennec, L., Dahmani, S., Gogniat, G., Maignan, C., and Sepúlveda, M. J. (2016b). Network contention-aware method to evaluate data coherency protocols within a compilation toolchain. In 10th IEEE International Symposium on Embedded Multicore/Many-core Systems-on-Chip, MCSOC 2016, Lyon, France, September 21-23, 2016, pages 249–256. IEEE Computer Society.
>
> [Cudennec et al., 2016b] CEA LIST, UPMC, Lab-STICC, TU Munich.
>
> Contributions | ★ initiative ■ idées □ implémentation □ expérimentations ■ rédaction ■ présentation

Évaluer les protocoles de cohérence de cache sur des processeurs nécessite bien souvent de recourir à des modèles et des simulations du fait de la complexité ou de l'impossibilité de modifier et substituer un protocole sur une architecture donnée. Différentes méthodes existent, caractérisées par un niveau de précision pour le résultat et une complexité algorithmique pour le temps de calcul, l'optimisation de ces deux caractéristiques étant contradictoire. La méthode analytique utilisée dans les évaluations précédentes (section 2.2.1) permet de déterminer les défauts de cache, de dresser une carte des communications et de générer une *heatmap* pour identifier les liens et les caches ayant été sollicités. Cette méthode ne permet cependant pas de déterminer les temps d'accès aux données. Dans les travaux suivants, nous avons proposé deux modèles temporels de complexité incrémentale : 1) un modèle basé sur les cycles processeur permettant de calculer le nombre de cycles nécessaires pour chaque accès et 2) un modèle basé sur un simulateur de NoC permettant de calculer la contention sur les liens réseau.

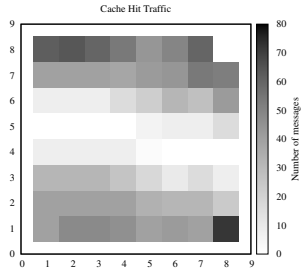
La méthode analytique basée sur les cycles processeur [Chaker et al., 2015] consiste à modéliser les différents types d'accès à une donnée selon sa localisation sur la puce, de décomposer les étapes nécessaires pour y accéder puis de calculer le nombre de cycles processeur requis pour compléter ces étapes. Quatre types d'accès ont été identifiés (représentés sur la figure 2.24a) : 1) l'accès au cache local, 2) l'accès dans le cache d'un voisin, 3) l'accès dans le cache d'un cœur distant et 4) l'accès dans la mémoire externe. Chaque type d'accès a été décomposé en une séquence d'actions sur la puce correspondant au cheminement de la requête à travers les caches, les liens et les routeurs du NoC ainsi que vers la mémoire externe. Pour chaque action il alors possible d'attribuer un coût en nombre de cycles processeur en se basant sur des valeurs fournies par l'utilisateur. Dans ce travail nous avons paramétré le modèle en suivant les recommandations pour la conception des caches fournies dans [Patterson and Hennessy, 2012] et les recommandations pour la conception de la mémoire et du NoC fournies dans [Hennessy and Patterson, 2012].

Les figures 2.24b et 2.24c présentent les résultats obtenus avec les approches analytiques par comptage de messages de CC et par comptage des cycles processeur. L'application est un noyau de convolution de taille 3 exécuté sur une puce de 8×8 cœurs avec le protocole de cache masse-ressort. Ces figures montrent bien une forte corrélation entre les messages de CC émis sur chaque cœur et le nombre de cycles processeur dédiés à la réalisation d'une requête d'accès. Une comparaison plus fine des protocoles de cohérence de cache MESI, glissement de données et masse-ressort a de plus été effectuée, notamment en faisant varier les stratégies de pistage des données migrées (comment retrouver un donnée ayant été migrée plusieurs fois et quelle stratégie consomme le plus de cycles processeurs). Les résultats peuvent être

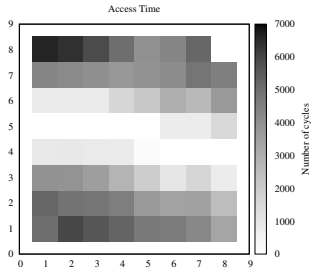
consultés dans [Chaker et al., 2015]. Si cette approche analytique permet d'affiner la modélisation du comportement d'un protocole de CC sur la puce, elle reste cependant éloignée des conditions opérationnelles, notamment car les ressources matérielles sont toujours considérées comme disponibles en l'absence de contention. Une extension de ce travail permet de prendre en compte la contention sur le NoC à l'aide d'un outil de simulation.



(a) Quatre types d'accès à une donnée à partir d'un cœur sur un NoC 2D (*mesh*). 1) accès au cache local, 2) accès dans le cache voisin, 3) accès dans un cache distant et 4) accès dans la mémoire externe.



(b) Nombre de messages de CC émis lors de l'exécution d'un noyau de convolution de taille 3. Résultats obtenus selon la méthode analytique basée sur les messages (section 2.2.1).



(c) Temps d'accès en nombre de cycles processeur cumulés lors de l'exécution du noyau de convolution. Résultats obtenus en appliquant le modèle analytique basé sur les cycles processeur.

FIGURE 2.24 – Comparaison des résultats d'exécution d'un noyau de convolution pour un protocole de cohérence masse-ressort avec les méthodes analytiques basées sur le rejeu de messages et sur les cycles processeur.

La méthode analytique basée sur un simulateur de NoC [Cudennec et al., 2016b] consiste à préparer une trace d'exécution contenant les accès aux données partagées (en utilisant la méthode précédemment présentée basée sur l'instrumentation du binaire par Pin/Pintools) puis de les rejouer dans un simulateur afin d'observer les phénomènes de contention sur les ressources du NoC. Le simulateur utilisé [Sepúlveda et al., 2007] est écrit en System-C sur le principe de la transaction (TLM (*Transaction Level Modelling*)) avec une précision au niveau du cycle. Ce simulateur a été choisi car il propose un bon compromis entre précision et rapidité d'exécution. Les fichiers de configuration permettent de décrire et paramétrer une topologie réseau arbitraire, d'injecter des communications puis de récupérer des métriques à l'aide de sondes placées sur les routeurs ou les liens de communication.

L'injection des communications nécessite un horodatage des évènements afin de les cadencer sur la puce simulée. Cette information temporelle n'est cependant pas disponible lors de la constitution des journaux d'accès mémoire à l'aide de l'outil Pin/Pintools (et plus spécifiquement le greffon *pinatrace*). Ce n'est pas une information critique car ce journal est généré en exécutant l'application sur une architecture hôte potentiellement très différente de l'accélérateur many-cœur cible. Il en résulte un comportement très différent en terme d'entrelacement des accès mémoire, et à fortiori en terme d'horodatage. En conséquence, un cadencement de la trace d'accès mémoire est reconstitué lors de la simulation. Ce cadencement est établi comme suit : 1) un niveau de parallélisme d'injection des accès mémoire est choisi, avec 0% correspondant au rejeu des accès pleinement séquentiels et avec 100% correspondant au rejeu des accès complètement concurrents (la figure 2.25 illustre différents niveaux de parallélisme). Les niveaux de parallélisme intermédiaires sont obtenus en partitionnant le journal d'accès mémoire en autant d'ensembles de traces contiguës dans le journal. Les accès appartenant à un même ensemble sont injectés en parallèle dans le simulateur. Les ensembles d'accès sont traités séquentiellement. De cette manière, il est possible de générer un comportement applicatif plus ou moins dense en terme d'accès mémoire : une configuration plutôt séquentielle permettant au NoC de traiter un évènement avant que le suivant ne se réalise, alors qu'une configuration tendant vers un haut degré de parallélisme génère de la contention sur le réseau.

L'évaluation de l'approche temporelle basée sur un simulateur de NoC a été effectuée pour différentes traces d'accès mémoire, topologies réseau et niveau de parallélisme d'injection d'évènements. La table 2.4 présente les résultats. Les accès mémoire sont générés en déployant des tâches exécutant un noyau de convolution susceptible de saturer rapidement le cache local. La simulation porte sur une puce de taille 8×8 cœurs et utilise le protocole MESI + glissement de données. Le mécanisme de coopération est enclenché aux abords des cœurs effectuant les traitements. Ces expérimentations ont permis d'observer de nouveaux phénomènes en comparaison des approches purement analytiques. Un premier résultat porte sur le choix de la topologie du NoC. Une organisation en tore semble intuitivement plus efficace qu'une grille classique car elle étend cette dernière avec de nouveaux liens sur les extrémités. Pour certains jeux de données (1 et 4) ce n'est pas le cas : une explication est que l'absence de voisins sur les quatre bords de la grille permet de préserver l'autre coté de la puce de l'influence croisée des migrations de données dues aux *hot spots*, tel un brise-lames. Un deuxième résultat porte sur le niveau de parallélisme d'injection des traces. Il montre que certains jeux de données (2 et 5) génèrent de hauts niveaux de contention sur le réseau indépendemment du niveau de parallélisme d'injection. Dans ce cas, le triptyque application-NoC-protocole de cohérence ne fonctionne pas correctement et d'autres combinaisons doivent être évaluées.

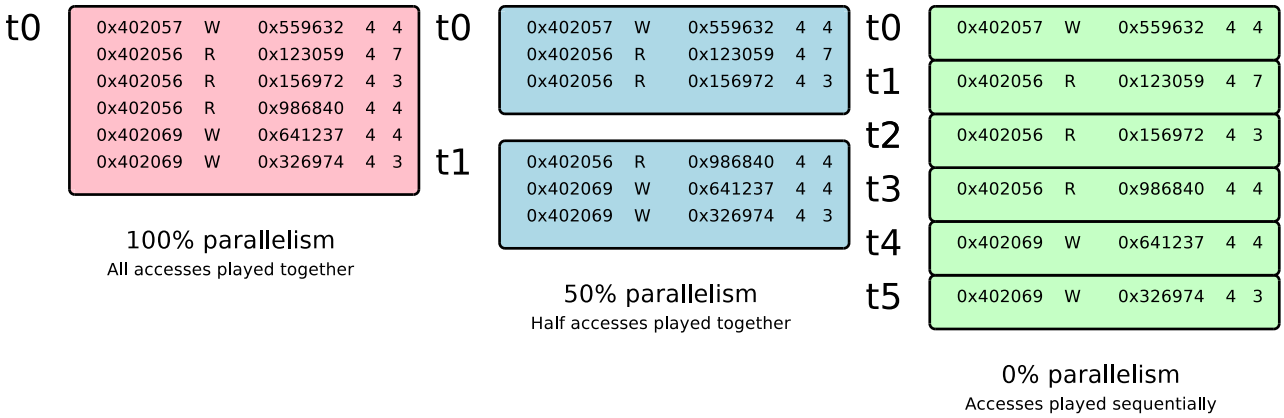


FIGURE 2.25 – Niveau de parallélisme pour l'injection des accès mémoire dans le simulateur de NoC.

Jeu de données			Temps de simulation (ms)	Topologie (nb. cycles)		% Parallélisme → % Contention				
	Données	Accès		Grille 2D	Tore 2D	0%	25%	50%	75%	100%
Jeu 1	136	329	1549	7778	9813	39%	39%	28%	45%	57%
Jeu 2	265	669	963	89625	87588	83%	88%	93%	89%	95%
Jeu 3	598	3396	1760	828885	772475	80%	76%	87%	87%	89%
Jeu 4	1071	3403	3661	1836805	1969467	66%	76%	73%	80%	66%
Jeu 5	1690	13758	10885	3719665	3584895	97%	81%	71%	78%	92%

TABLE 2.4 – Temps de simulation du NoC, nombre de cycles simulés et taux de contention maximal observé sur le réseau pour 5 jeux de données correspondant à des accès mémoire générés par des noyaux de convolution. Le protocole de cohérence simulé est MESI + glissement de données présenté en section 2.2.1, sur une puce de taille 8×8. Les temps de simulation sont obtenus sur un processeur mobile Intel Core i5-3360M (2012).

En conclusion, La méthode analytique par comptage de cycles processeur offre une complexité algorithmique très simple qui permet d'obtenir les résultats sur un temps de calcul court. La précision reste cependant à grosse maille, principalement car l'approche ne prend pas en compte les limitations matérielles de la puce. Le recours à un simulateur de niveau TLM apporte de nouvelles informations sur le taux de contention des liens et des routeurs du NoC, pour un temps de calcul significativement plus important, mais qui reste raisonnable dans le cadre d'une phase de profilage de système complexe. Cette approche n'est cependant pas complète du fait de la génération de journaux d'accès mémoire ne représentant pas l'activité réelle sur la puce : les traces sont générées en exécutant le programme sur un multi-cœur classique et ne contiennent pas les informations de placement-routage ni d'horodatage.

Les trois approches d'évaluation de protocoles de CC (analytique par comptage de messages, temporelle par comptage de cycles processeur, temporelle reposant sur un simulateur de NoC) montrent comment construire incrémentalement un modèle offrant graduellement plus de précision, au prix de complexifier l'implémentation et demander plus de temps de calcul. Comme dans de nombreux domaines, la bonne approche est souvent déterminée par le contexte opérationnel : la ligne directrice que nous avons fixée pour cette thématique de recherche consiste à rendre le choix du protocole de cohérence transparent pour l'utilisateur. Ceci implique de pouvoir explorer différentes solutions : quel protocole déployer, comment le configurer pour telle application s'exécutant sur tel processeur many-cœur ? Dans ce contexte de recherche opérationnelle il est important de pouvoir disposer d'outils d'évaluation des solutions adaptés aux algorithmes d'exploration et d'optimisation, tant sur les critères de précision que de temps de calcul. En ce sens, les contributions effectuées sur l'évaluation des protocoles de CC nous ont permis de réaliser de nombreuses études exploratoires et de proposer une chaîne de compilation pour l'aide à la décision sur la cohérence des données.

2.2.3 Aide à la décision pour le choix des protocoles de cohérence

“ Dahmani, S., Carpov, S., Cudennec, L., and Gogniat, G. (2015). A multiobjective consistency decision engine for a manycore compilation platform using an evolutionary approach. In Evolutionary Multiobjective Optimization (EMO 2015), held in conjunction with the 23rd Intl. Conf. on Multiple Criteria Decision Making (MCDM 2015), Hamburg, Germany. ”

[Dahmani et al., 2015] CEA LIST, Lab-STICC.

Contributions | ★ initiative ■ idées □ implémentation □ expérimentations ■ rédaction □ présentation

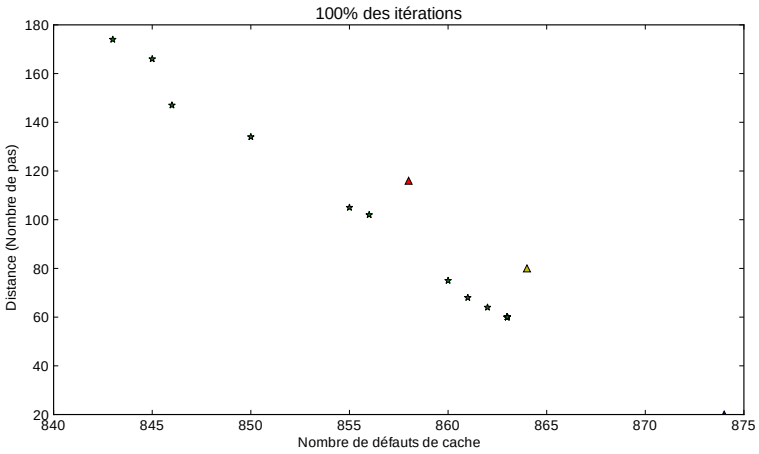


FIGURE 2.26 – Solutions trouvées pour la configuration d'un protocole de cohérence par glissement pour chaque donnée partagée. La configuration porte sur la variation du rayon maximum de glissement. Les critères d'évaluation sont respectivement représentés en abscisse et en ordonnée par le nombre de défauts de caches et le nombre cumulés de *hops* effectués sur le NoC pour atteindre une donnée. Le front de Pareto apparaît avec des étoiles. Deux solutions dominées sont représentées par des triangles. Cette figure est issue du manuscrit de thèse de Safae Dahmani [Dahmani, 2015].

L'évaluation des protocoles de CC que nous avons proposés a montré qu'il n'existe pas un protocole universel répondant à toutes les combinaisons d'application, de cible matérielle et d'objectifs de performance. Le choix d'un protocole et sa configuration reste donc dirigé par l'expertise du développeur dans les trois domaines sus-cités, ce qui éloigne de la compétence purement métier. Une approche légitime est de proposer un système d'aide à la décision permettant à partir d'une application et d'une architecture cible d'offrir un ensemble de solutions optimales répondant à divers contextes opérationnels. C'est le principe du front de Pareto pour les problèmes d'optimisation multicritères, qui permet de conserver les solutions non-dominées, c'est à dire que pour chacune d'elles, aucune solution meilleure sur tous les critères n'a été trouvée. Avec un tel outil, l'utilisateur obtient donc automatiquement des configurations adaptées au déploiement de son application sur la puce, toute l'expertise sur le comportement des protocoles de cohérence se situant désormais dans une étape d'analyse et de compilation.

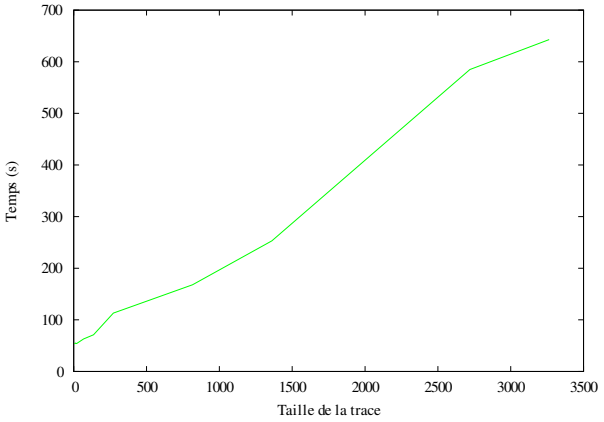
La figure 2.26 présente un exemple de front de Pareto appliqué à l'exploration de la configuration du protocole coopératif par glissement de données. La configuration porte sur la distance maximale autorisée pour repousser une donnée de voisins en voisins. Cette distance est potentiellement différente pour chaque donnée partagée. Les deux critères retenus pour évaluer les solutions sont 1) le nombre de défauts de cache et 2) la distance cumulée sur le NoC entre le cœur hébergeant une donnée et le cœur propriétaire. Ces deux critères doivent être minimisés, c'est à dire trouver des solutions qui se rapprochent de l'origine sur cette figure. Cette recherche n'est pas triviale car avec ce protocole de CC, autoriser les données à migrer de caches en caches permet de les conserver sur la puce, ce qui diminue les défauts de cache (absence de requêtes vers la mémoire externe), mais augmente dans le même temps le nombre de *hops* pour les retrouver, et donc la latence. L'exploration automatique permet alors de sélectionner des solutions jugées satisfaisantes sur ces deux critères puis de les fournir à un autre outil d'aide à la décision en charge d'orienter le choix final. Ceci peut être effectué par pondération des deux critères en se basant sur les caractéristiques techniques de l'architecture ciblée : combien coûtent des accès à la mémoire externe et aux caches distants.

Un moteur de décision multiobjectifs. Afin de construire un front de Pareto pertinent, il est nécessaire d'explorer un nombre de solutions suffisamment grand et diversifié pour ne pas tomber dans le piège du minimum local, c'est à dire se cantonner à quelques solutions proches qui semblent bonnes, alors que des solutions meilleures se situent dans des zones non couvertes par l'exploration. La méthode la plus simple est la recherche exhaustive qui évalue toutes les combinaisons possibles et retourne la ou les meilleures solutions possibles. Cette recherche n'est pas applicable lorsque le temps de calcul, qui dépend du nombre de solutions et du temps nécessaire à leur évaluation est trop important. Les méthodes approchées permettent alors d'explorer un sous-ensemble des solutions en dirigeant les recherches à l'aide d'heuristiques et de prédictions. Pour donner un ordre de grandeur, des temps de calcul sont donnés dans la table 2.27a pour comparer une recherche exhaustive et une méthode approchée basée sur un algorithme génétique. La méthode exhaustive est caractérisée par une explosion de la combinatoire qui entraîne une croissance géométrique du nombre de solutions à évaluer, là où la méthode génétique permet de maîtriser le nombre de solutions à évaluer, indépendamment de la complexité du problème (figure 2.27b). Avec des traces de seulement quelques dizaines d'accès mémoire le temps de calcul de la méthode exhaustive devient rapidement déraisonnablement long avec presque 18 heures de calcul pour une trace constituée de seulement 20 accès mémoire.

Dans ce travail nous avons défini le problème d'optimisation de la cohérence de cache de la sorte : une trace d'exé-

Taille de la trace	Exhaustif		Génétique	
	(s)	#	(s)	#
6	24	64	53	100
10	240	1024	53	100
15	10312	32768	54	100
20	63976	1048576	54	100

(a) Temps de calcul (secondes) nécessaire à l'exploration de la configuration des protocoles de CC pour les algorithmes de recherche exhaustif et génétique, en fonction du nombre d'accès mémoire contenus dans les traces d'exécution. Le rayon de glissement maximal r est fixé à 2 pour limiter les temps de calcul. Le nombre de solutions (#) est donc r^n avec n la taille de la trace. L'algorithme génétique est configuré pour 10 itérations et une population de 10 individus.



(b) Temps de calcul pour l'algorithme génétique.

FIGURE 2.27 – Comparaison du temps de calcul entre l'algorithme exhaustif et l'algorithme génétique. Expériences menées sur 4 cœurs d'un processeur AMD Opteron 6172. Ces résultats sont issus du manuscrit de thèse de Safae Dahmani [Dahmani, 2015].

cution contient N accès mémoire. Chaque accès mémoire $n \in N$ porte sur une donnée partagée d. P est un ensemble de protocoles de CC. Chaque protocole $p \in P$ peut être configuré avec un nombre variable de paramètres. L'ensemble des configurations possibles pour un protocole p est noté C_p. Un premier problème consiste à associer pour chaque donnée partagée d un protocole de cohérence $p \in P$. Un deuxième problème consiste à choisir une configuration $c \in C_p$ du protocole de cohérence p à chaque accès mémoire n auquel il est associé. L'espace de recherche est donc potentiellement très grand. Pour limiter le temps calcul, deux solutions complémentaires peuvent être mises en œuvre : 1) limiter la taille de la trace en sélectionnant des parties représentatives de l'exécution et 2) utiliser des algorithmes de RO (Recherche Opérationnelle) adaptés à la résolution de ce type de problèmes, et c'est justement la spécialité des algorithmes génétiques.

Le principe d'un algorithme génétique est de faire évoluer une population définie par un ensemble de solutions (des individus) pendant un certain nombre d'itérations. La taille de la population reste constante mais les individus qui la composent évoluent en appliquant des mécanismes inspirés de la biologie : 1) la *sélection* permet de conserver un sous-ensemble de solutions plus adaptées aux critères d'évaluation, 2) le *croisement* permet de recombiner de nouvelles solutions en empruntant des caractères à des solutions existantes et 3) les *mutations* introduisent de l'aléa en altérant certaines solutions de manière aléatoire. L'efficacité d'un algorithme génétique est en grande partie déterminée par le paramétrage des ces trois mécanismes pour obtenir de bonnes solutions dans la population en un minimum d'itérations. Dans ce travail nous utilisons l'approche FastPGA [Eskandari et al., 2006] qui inclut dans l'étape de sélection des solutions de l'algorithme génétique la notion de solutions dominées pour construire un front de Pareto.

Un facteur de diversité caractérise la distance entre les solutions contenues dans la population, ce qui est obtenu en paramétrant l'ampleur des croisements entre individus et les probabilités de mutation notées ρ. Ce facteur de diversité est utilisé à chaque itération pour paramétrer dynamiquement ρ dans l'algorithme génétique, et ainsi conserver une diversité asservie aux générations précédentes. La figure 2.28 présente le front de Pareto obtenu en affichant les solutions trouvées par les trois stratégies : un paramétrage de ρ constant minimal, maximal, et variable. Les résultats montrent que le paramétrage maximal donne de meilleurs résultats que le paramétrage minimal, permettant d'explorer des solutions plus éloignées en terme de configuration. Cependant, l'usage d'un ρ élevé mène à une diversité plus importante et à la possible perte des informations utiles qui ne sont alors plus exprimées sur les individus. Les résultats montrent aussi que le paramétrage dynamique est en mesure de trouver des solutions meilleures qu'un paramétrage maximal, pour un même nombre d'itérations et donc à complexité calculatoire identique : si l'on fusionne les deux fronts de Pareto correspondants au paramétrage maximal et au paramétrage dynamique, des solutions du premier disparaissent, ce qui n'est pas le cas pour le second.

Le moteur de décision multiobjectifs est une brique technologique déterminante pour offrir un système permettant d'automatiser le choix et la configuration des protocoles de CC. En ce sens, le travail réalisé dans le cadre de la thèse de Safae Dahmani est important car il démontre la faisabilité de l'approche en levant un verrou technologique et en s'attaquant à un problème "dur". Cette brique est un élément d'une chaîne de compilation ambitieuse dans laquelle les contributions présentées dans ce manuscrit sont assemblées pour permettre d'automatiser les traitements à partir du code source.

Une chaîne de compilation pour le choix d'un protocole de CC. Choisir et paramétrer des protocoles de cohérence est une tâche complexe : le comportement des protocoles, et donc leur performance, dépend de nombreux paramètres tels que les motifs d'accès mémoire générés par l'application, la configuration du déploiement, les caractéristiques de la plateforme matérielle et les objectifs en terme de performance calculatoire ou énergétique. Rendre ce

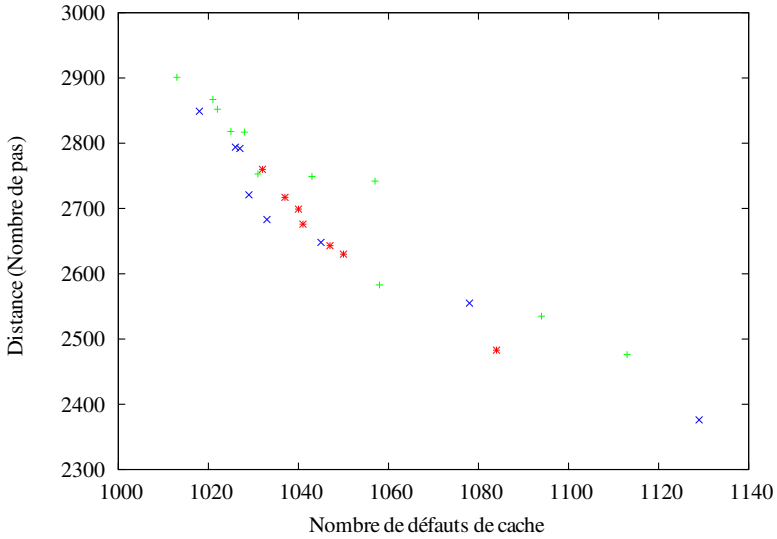


FIGURE 2.28 – Front de Pareto obtenu pour le paramétrage du protocole de cohérence de glissement de données en appliquant l'algorithme génétique FastPGA sur une trace contenant 2080 accès mémoire. Les solutions proches de l'origine sont les meilleures. Les points verts correspondent à une faible probabilité de croisement et de mutation, les points rouges à une forte probabilité et les points bleus à une probabilité variable calculée dynamiquement à chaque itération. Cette figure est issue du manuscrit de thèse de Safae Dahmani [Dahmani, 2015].

service de décision de manière transparente à l'utilisateur fait donc intervenir de nombreuses étapes spécialisées dans l'analyse et l'évaluation de ces paramètres. Le développement de la thématique sur la cohérence des données pour les architectures many-cœurs a permis de dessiner une vision plus globale du problème dans laquelle les contributions deviennent complémentaires et forment un processus de compilation. La figure 2.29 illustre l'organisation d'une telle chaîne de compilation. Cette dernière est constituée des étapes suivantes :

1. Un code source décrivant une application parallèle est fourni en entrée,
2. (a) Le code source est compilé sur la plateforme hôte et le binaire est instrumenté (par exemple par Pin/Pintools),
   (b) Le binaire est exécuté sur la plateforme hôte et différentes traces d'exécution sont collectées afin de constituer une base représentative des accès aux données partagées,
3. (a) Le code source est analysé par un outil d'analyse statique de code (par exemple Frama-C [Kirchner et al., 2015]) permettant d'extraire les accès aux données partagées et de construire un graphe de dépendances entre ces accès,
   (b) Un partitionnement des accès est produit, reflétant les différentes phases de fonctionnement de l'application, par exemple *initialisation*, *calcul*, *terminaison*...
4. Une brique de décision associe à chaque accès mémoire une configuration du protocole de cohérence choisi pour cette donnée partagée. Pour ce faire, cette brique de décision prend en entrée une bibliothèque de protocoles de CC ainsi que des objectifs et des fonctions de coût. Les solutions explorées peuvent être évaluées en faisant appel à un modèle analytique ou temporel présentés précédemment,
5. Les décisions sont inscrites dans le logiciel système généré et appliquées lors du déploiement sur la cible matérielle.

Plusieurs étapes de cette chaîne de compilation ont été conçues, implémentées, expérimentées, combinées et ont donné lieu à des publications. Certaines étapes n'ont cependant pu être suffisamment étudiées, ce qui n'a pas permis d'effectuer des expérimentations de *bout en bout*, ni d'observer si l'optimisation des mécanismes de cohérence entraîne un bénéfice réel au niveau applicatif (les démonstrations ont été effectuées à l'aide de modèles analytiques). C'est le cas de l'analyse statique de code et de la génération du logiciel système pour appliquer les décision de cohérence à l'exécution. 1) Pour l'analyse statique de code, des travaux préliminaires ont été menés dans le cadre de deux stages de master co-encadrés avec Selma Azaiez et Safae Dahmani au CEA. Ces stages ont donné lieu à la construction d'un modèle représentant les dépendances entre les accès aux données, basée sur la relation de causalité et des outils tels que les réseaux de Pétri et les arbres de décision. Ces travaux préliminaires sont présentés dans [Dahmani, 2015]. 2) L'application des configurations de protocoles de CC sur la plateforme cible renvoie au problème du déploiement ou de la substitution des protocoles de cohérence dans les processeurs. Cette opération est difficilement implémentable au niveau matériel ou au niveau système d'exploitation. L'approche la plus simple consiste à fournir un intergiciel de gestion de données partagées implémenté au niveau utilisateur. Un tel intergiciel s'apparente à une MVP dans laquelle l'utilisateur a toute la liberté d'expérimenter des stratégies de cohérence. Malheureusement, la MVP présentée en section 3.1 était encore aux toutes premières étapes de sa conception lorsque nous avons refermé cette thématique

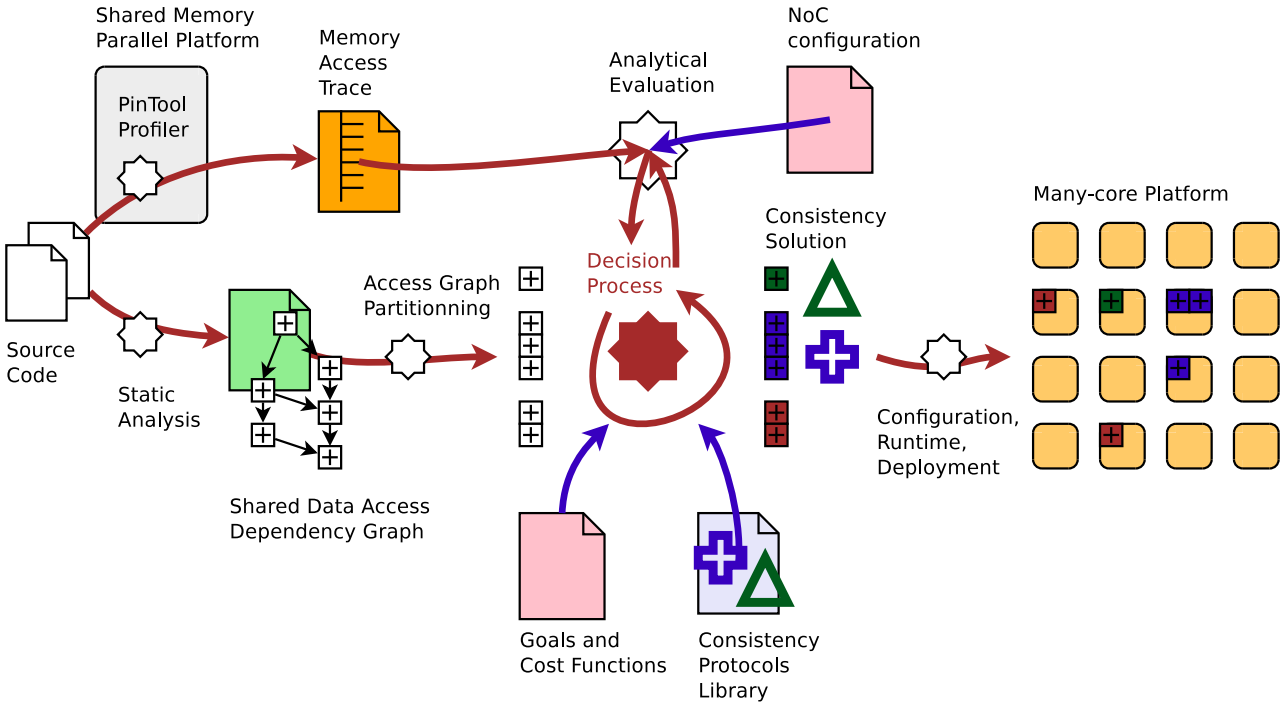


FIGURE 2.29 – Chaîne de compilation pour le choix et le paramétrage de protocoles de CC. Les étapes n'ayant pu être menées jusqu'à l'implémentation dans cette thématique de recherche sont l'analyse statique de code et la génération du logiciel système en charge de l'application des décisions de cohérence sur les accès mémoire.

de recherche. Ceci rappel l'importance d'assurer la maîtrise des outils nécessaires à l'évaluation des prototypes de recherche, de concert avec la temporalité des projets engagés.

En conclusion, concevoir un système d'optimisation de la cohérence des données, transparent pour l'utilisateur, repose sur deux principales difficultés : 1) la brique de prise de décision doit faire face à un nombre important de combinaisons possibles. L'évaluation des solutions est coûteuse, ce qui implique l'utilisation de méthodes approchées pour maîtriser la combinatoire. Sur ce point, l'utilisation de l'algorithme génétique FastPGA répond à la problématique tant sur le plan de la complexité calculatoire que sur la qualité des résultats. 2) Cette brique de prise de décision doit être correctement alimentée avec une description fidèle du comportement de l'application. Les applications parallèles considérées dans ce travail sont écrites en langage C et le parallélisme est exprimé dans le modèle *Posix Threads*, en conformité avec les recommandations de Frama-C, ce qui offre un grand degré d'expressivité sur le plan du langage. Cette expressivité rend difficile l'analyse de l'application parallèle et plus spécifiquement lorsque celle-ci est caractérisée par des accès irréguliers à la mémoire. Ces accès irréguliers sont souvent générés par des imbrications complexes de boucles (*for*, *while*) dont les bornes peuvent parfois être uniquement connues à l'exécution. Une approche possible est de recourir au modèle polyédrique [Benabderrahmane et al., 2010] qui offre un cadre algébrique pour représenter l'imbrication des boucles sous forme de polyèdre. Cette représentation permet ensuite d'effectuer des optimisations sur des parties complexes du code et fait le lien avec la compilation du parallélisme et l'ordonnancement des instructions basés sur l'analyse des boucles imbriquées [Karp et al., 1967]. L'analyse statique d'applications parallèles reste un sujet de recherche ouvert et il est bien souvent tentant de demander à l'utilisateur de décorer son code pour aider le compilateur à identifier les variables partagées et les accès concurrents. Une telle solution rend l'approche moins transparente pour l'utilisateur et demande un travail supplémentaire pour porter les applications existantes. Par ailleurs, la décoration de code est un premier pas vers l'utilisation de modèles de programmation fortement structurés, tels que le flot-de-données, pour lequel le déploiement d'un protocole de cohérence est rendu caduc par la pré-planification et la synchronisation des accès mémoire. Une autre perspective est l'utilisation des techniques d'IA pour l'optimisation des protocoles de cohérence. Cette technique est déjà appliquée à la problématique du préchargement de données dans les caches [Hashemi et al., 2018] et il est possible d'imaginer qu'elle soit étendue au choix et à la configuration du protocole de cohérence. Enfin, la conception d'une chaîne de compilation universelle incluant des étapes pour l'optimisation des mécanismes de cohérence des données, et capable d'analyser des programmes parallèles exhibant des accès irréguliers en mémoire reste peut-être aujourd'hui un objectif illusoire : une approche plus pragmatique relevant plus du principe de l'aide à la décision et s'attelant à conserver l'expertise du développeur dans la boucle.

2.3 Discussion

Les processeurs many-cœurs représentent une architecture de calcul particulière, concentrant les problématiques classiques des systèmes distribués à l'échelle d'une puce de calcul. La gestion des données partagées, le placement des tâches et d'une manière plus générale le modèle de programmation sont des thématiques bien maîtrisées pour les architectures multi-cœurs. Cependant, dans le contexte des many-cœurs, ces thématiques se heurtent au passage à l'échelle et leur résolution devient critique pour permettre une exploitation efficace des ressources sur la puce. En 2007, lorsque le français Kalray se lance dans la conception d'une telle puce, le MPPA 256 cœurs représente une architecture à l'état de l'art, si ce n'est en avance sur ses compétiteurs Intel et Tilera offrant alors des puces entre 60 et 100 cœurs. Dans ce contexte le laboratoire commun entre le CEA et Kalray a constitué une opportunité unique pour la conception finement liée d'une architecture matérielle et d'un environnement d'exploitation logiciel. Cela a aussi été l'occasion de concilier projet industriel et activités de recherche au sein du laboratoire commun, avec toutes les fertilisations croisées qui ont donné lieu au transfert technologique réussi de la chaîne de compilation Σ-C, ainsi qu'à de nombreuses publications dans des conférences internationales.

Retour d'expérience sur Σ-C. Proposer un nouveau langage pour programmer un accélérateur est un choix audacieux et difficile à justifier : les concepteurs d'applications, que ce soit dans les milieux académique ou industriel restent attachés à des langages éprouvés, possiblement moins adaptés à l'architecture ciblée, mais offrant de fortes garanties pour la réussite et la pérennité du projet. Ainsi la maturité du langage et le suivi du compilateur, le support pour l'aide, l'importance de la communauté active, la portabilité du code sur différentes architectures sont autant de critères pouvant justifier un tel choix. L'absence d'ouverture de Σ-C à la communauté internationale explique que le langage n'a jamais été utilisé en dehors de la sphère de développement. Cette stratégie du secret se traduit par plusieurs problématiques critiques : 1) le projet ne peut bénéficier d'un retour de la communauté scientifique à même d'orienter les choix de conception pour coller au mieux aux besoins des utilisateurs, 2) lorsque le langage est révélé, aucune communauté scientifique n'est constituée et certaines contributions ne sont plus à l'état de l'art face à une concurrence importante dans le domaine et 3) le choix de ne pas diffuser la chaîne de compilation sous licence libre exclut de facto la possibilité de tester, comparer et de s'impliquer dans le projet, le reléguant ainsi au statut de produit fantôme.

Pour autant, les briques technologiques constituant la chaîne de compilation Σ-C ont une grande valeur scientifique et technologique, valorisées individuellement dans des publications internationales. De plus, la chaîne de compilation a permis d'exploiter très tôt le parallélisme offert par la puce MPPA, ce qui est une véritable réussite. Dans un projet tel que Σ-C, plusieurs initiatives peuvent être envisagées pour capitaliser sur les travaux engagés. 1) Distribuer sous licence libre les premières étapes de la chaîne de compilation comprenant le *parseur*, la construction du flot-de-données et les étapes nécessaires au paramétrage d'un intergiciel d'exécution générique, pour un portage Posix ou OpenMP [Louise, 2017]. Cette approche vise à promouvoir le langage Σ-C tout en préservant la propriété intellectuelle sur les briques d'optimisation de la puce Kalray MPPA. 2) Proposer un compilateur source-à-source permettant de traduire des programmes écrits dans des langages flot-de-données ou assimilés vers du Σ-C, par exemple du Streamit ou de l'OpenCL. Cette approche permet de bénéficier de la base applicative de ces langages source et d'effectuer une comparaison sur les performances des compilateurs respectifs.

Si le constat peut sembler sévère sur la stratégie suivie pour le développement de Σ-C, il n'en reste pas moins que l'expertise acquise dans le laboratoire suite à ce projet s'est traduite par de nombreuses innovations dans le domaine des DSL pour le flot de données, notamment pour le temps-réel critique mixte [Louise et al., 2014] ou les many-cœurs en général [Goubier et al., 2014].

Le many-cœur et l'hétérogénéité. L'une des forces des architectures many-cœur, le nombre élevé de cœurs, constitue aussi leur faiblesse : comment tirer partie efficacement de toutes ces ressources, en alimentant de manière homogène les unités de calcul ? Si les applications de traitement du signal peuvent facilement bénéficier de ce type d'architecture, de nombreuses applications présentant moins de régularité dans la décomposition des tâches et des accès mémoire peinent difficilement à passer à l'échelle sur des centaines de cœurs. Avec le développement des applications liées à l'IA, notamment pour l'apprentissage machine, les nouvelles architectures de calcul incluent désormais des unités de traitement spécialisées dans la réalisation de convolutions et d'opérations vectorielles. Cette tendance est perceptible pour certains many-cœurs (MPPA Coolidge, MN-Core) dont l'architecture devient hétérogène. Ceci introduit de nouvelles problématiques sur le plan matériel (moins de régularité pour une circuiterie plus complexe) et sur le plan logiciel (comment sélectionner et déployer des noyaux de calcul issus d'applications sur certaines parties matérielles de la puce). Mais l'hétérogénéité dans les architectures de calcul n'est pas une nouveauté, comme nous allons le voir dans le chapitre suivant, et les many-cœurs sont partie intégrante de cette évolution.

□

Chapitre 3

Données partagées pour les machines hétérogènes réparties

L'essor des architectures hétérogènes est porté par plusieurs vagues technologiques et innovations d'usage, à l'instar des processeurs many-cœurs traités dans le chapitre 2. Dans les années 2000, l'hétérogénéité est souvent représentée par un système intégré comprenant un processeur généraliste hôte et un accélérateur matériel. Cette organisation est rendue populaire avec le détournement des cartes de rendu vidéo GPU pour effectuer du calcul numérique classique, permettant ainsi de démultiplier la vitesse d'exécution des codes parallèles réguliers en comparaison d'un CPU (*Central Processing Unit*). Si ce modèle perdure largement dans les systèmes HPC comme en témoignent les classements du TOP500 faisant la part belle aux GPGPU de Nvidia, la miniaturisation des composants, l'accélération du développement des systèmes embarqués grand public et l'évolution des contextes applicatifs nomades sont à l'origine de nouvelles architectures hétérogènes et distribuées. Ce constat est aujourd'hui étayé par le déferlement des applications touchant à l'IA, que ce soit pour des phases d'apprentissage machine ou d'inférence. Il est désormais courant d'utiliser dans le quotidien des systèmes embarqués composés de multiples unités de calcul dédiées à des tâches spécifiques et dont l'architecture est spécialisée en conséquence.

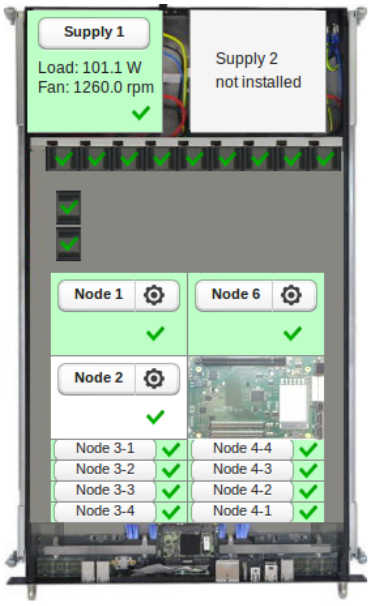
Dans une plateforme distribuée l'hétérogénéité intervient à différents niveaux : hétérogénéité des architectures de calcul (CPU, GPU, FPGA..), hétérogénéité des liens réseaux (Ethernet, Infiniband, Wifi, USB..), hétérogénéité des systèmes d'exploitation et des bibliothèques logicielles. À titre d'exemple, les processeurs basse-consommation basés sur l'architecture ARM (*Acorn/Advanced Risc Management/Machine*), développés à l'origine dans les années 1980 par Acorn (avec la participation d'Apple quelques années plus tard), présentent un jeu d'instructions et une organisation matérielle étudiés pour être plus économe en énergie que les processeurs traditionnels. Ceci justifie leur adoption massive à partir de 2010 dans les *smartphones*, les tablettes, les cartes de développement, les kits de robotique (Raspberry Pi, Odroid), plus récemment dans les ordinateurs portables Chromebook et Apple MacBook (avec la puce M1) et même dans les super-calculateurs du TOP500 comme c'est le cas pour le processeur Fujitsu A64FX [Okazaki et al., 2020]. Des infrastructures hybrides proposant des processeurs classiques (x86) et basse-consommation ARM permettent d'adapter au plus juste les performances applicatives et la consommation d'énergie, c'est le cas pour les micro-serveurs HPE Moonshot [HPE, 2014]. Ainsi, les applications peuvent être déployées sur des processeurs adaptés aux contraintes du moment. L'hétérogénéité intervient ici sur deux plans : 1) les architectures x86 et ARM sont dotées de jeux d'instruction différents et parfois d'adressage mémoire différents (32 et 64 bits) ce qui implique des versions du logiciel adaptées pouvant mener à des comportements et des résultats différents. 2) Ces deux architectures processeur visent des objectifs différents pour les performances calculatoires et la consommation énergétique, deux critères bien souvent antagonistes. Même si ces divergences tendent à s'estomper, cet exemple de plateforme hétérogène reflète la problématique de la portabilité entre nœuds de calcul.

Un autre exemple exhibant de l'hétérogénéité est représenté par les accélérateurs dédiés : Dans le contexte des tâches d'IA, de nombreuses puces GPGPU, ASIC, FPGA et many-cœurs sont proposées dans les serveurs et systèmes embarqués, comme par exemple les puces A100, T4 et Tegra de Nvidia, les TPU (*Tensor Processing Unit*) et CPU Tensor Core de Google, les *dongles* USB Movidius Myriad d'Intel, les processeurs reprogrammables Arria d'Intel, Vitis AI de Xilinx ou le many-cœur MPPA Coolidge de Kalray. Ces accélérateurs sont exploités par des suites logicielles spécifiques et bien souvent non-interopérables, ce qui rend le déploiement d'une application inter-accélérateurs compliqué techniquement.

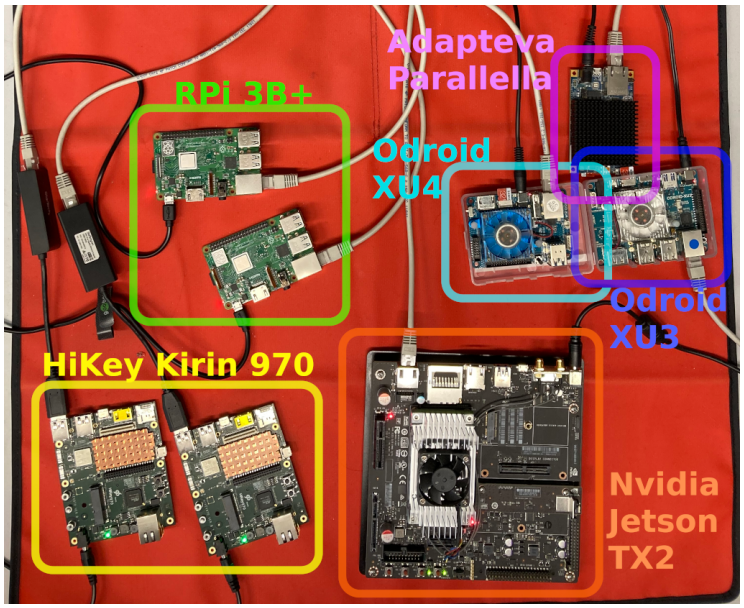
> Pourtant, ces systèmes hétérogènes répondent à un enjeu de taille : comment concilier la performance calculatoire nécessaire aux nouveaux usages tout en restant dans une enveloppe énergétique contrainte par la technologie des batteries ou la dépense raisonnée des ressources.

Les travaux présentés au sein du chapitre s'inscrivent dans ce contexte technologique, à la croisée de plusieurs

thématiques de recherche souvent étudiées de manière indépendante : les systèmes embarqués, le calcul sur ressources hétérogènes et le partage de données dans les systèmes distribués. L'ensemble apporte de nouvelles hypothèses de travail qui nécessitent des contributions innovantes pour exploiter au mieux ces nouvelles architectures. L'approche proposée ici est d'aborder la problématique à travers le prisme de la gestion des données partagées. Pour illustrer cette problématique, deux plateformes sont présentées dans la figure 3.1. La première plateforme est un micro-serveur industriel RECS|Box 3 commercialisé par Christmann (figure 3.1a) en partie développé au sein des projets H2020 FiPS [Griessl et al., 2014] puis M2DC [Oleksiak et al., 2017] dans lesquels le CEA a contribué pour l'intégration des FPGA ainsi que pour des modèles de programmation. Cette plateforme est composée d'un châssis au format 1U comprenant les réseaux pour les données, la distribution d'énergie et les capteurs. Des emplacements sont prévus pour des cartes d'extension dédiées au calcul hétérogène, permettant de mixer processeurs haute-performance, processeurs basse-consommation et accélérateurs GPU et FPGA. L'ensemble forme une plateforme très intégrée et modulaire. La deuxième plateforme est un bac-à-sable expérimental (figure 3.1b) assemblé au CEA pour évaluer des intergiciels pour les calculs distribués. Cette plateforme est constituée d'un ensemble de cartes de développement embarquées interconnectées par un réseau Ethernet. Ces cartes sont représentatives des architectures de calcul embarquées dans les *smartphones*, la robotique et les véhicules autonomes.



(a) Christmann RECS|Box 3 (Antares).



(b) Machine hétérogène bac-à-sable.

Christmann RECS|Box 3 (Antares 1U)						
Nœud	Processeur	Cœurs	Mémoire	Disque	Réseau	#
1 & 6	Intel Core i7 4700-EQ	4	16GB	mSATA SSD	Gb Ethernet	2
2	ARM Cortex A9 Xilinx XC7Z045	2	16GB	SD	USB 2.0	1
3 & 4	ARM Cortex A15 Exynos 5250	2	4GB	eMMC	USB 2.0	8
Machine hétérogène bac-à-sable						
Nœud	Processeur	Cœurs	Mémoire	Disque	Réseau	#
Gateway	Intel Core i7 6800K	6	64GB	SATA SSD	Gb Ethernet	1
Raspberry Pi 3B+	ARM Cortex A53	4	1GB	SD	USB 2.0	2
Odroid XU4	ARM Cortex A15/A7	4/4	2GB	SD	USB 3.0	1
Odroid XU3	ARM Cortex A15/A7	4/4	2GB	SD	USB 2.0	1
HiKey Kirin 970	ARM Cortex A73/A53	4/4	6GB	UFS	USB 3.0	2
Nvidia Jetson TX2	Denver/ARM Cortex A57	2/4	8GB	eMMC	Gb Ethernet	1
Adapteva Parallella	ARM Cortex A9 Xilinx 7020/Epiphany III	2/16	1GB	SD	Gb Ethernet	1

(c) Description des nœuds de calcul pour les deux machines hétérogènes. # indique le nombre de nœuds de chaque type.

FIGURE 3.1 – Les deux machines hétérogènes utilisées dans les expérimentations présentées dans ce manuscrit.

Dans ces plateformes, l'hétérogénéité se traduit aussi par des performances calculatoires et des consommations d'énergie très différentes. Par exemple la consommation instantanée passe de quelques watts pour un processeur ARM Cortex A9 à plusieurs dizaines de watts pour un Intel i7 et jusque plusieurs centaines de watts pour certains gros accélérateurs (par exemple la carte Nvidia A100 SXM a une enveloppe thermique de 400 watts). En contrepartie, le temps de calcul est largement favorable au processeur i7 ou au GPU (pour peu que l'application exploite correctement le parallélisme), ce qui donne lieu à des budgets énergétiques nécessaires pour compléter le calcul et pouvant être

comparés (par exemple en joules). Bien que conçues pour des environnements opérationnels éloignés (en production ou pour de la recherche), ces deux plateformes partagent les mêmes problématiques de programmabilité. Et dans ce cas, cette problématique ne se limite pas à la simple utilisation des ressources pour distribuer le calcul sur les nœuds : elle consiste aussi à faire des choix sur le déploiement de l'application afin de respecter des contraintes à l'exécution (performance calculatoire) et remplir des objectifs opérationnels (réduire l'empreinte énergétique).

L'approche présentée dans ce manuscrit est de proposer un modèle de programmation unifié en mémoire partagée à l'échelle de la plateforme hétérogène distribuée, et de fournir des outils pour établir de manière semi-automatique un compromis entre performance calculatoire et dépense énergétique. Ces contributions répondent aux problématiques de la programmabilité et de l'exploitation rationnelle des ressources.

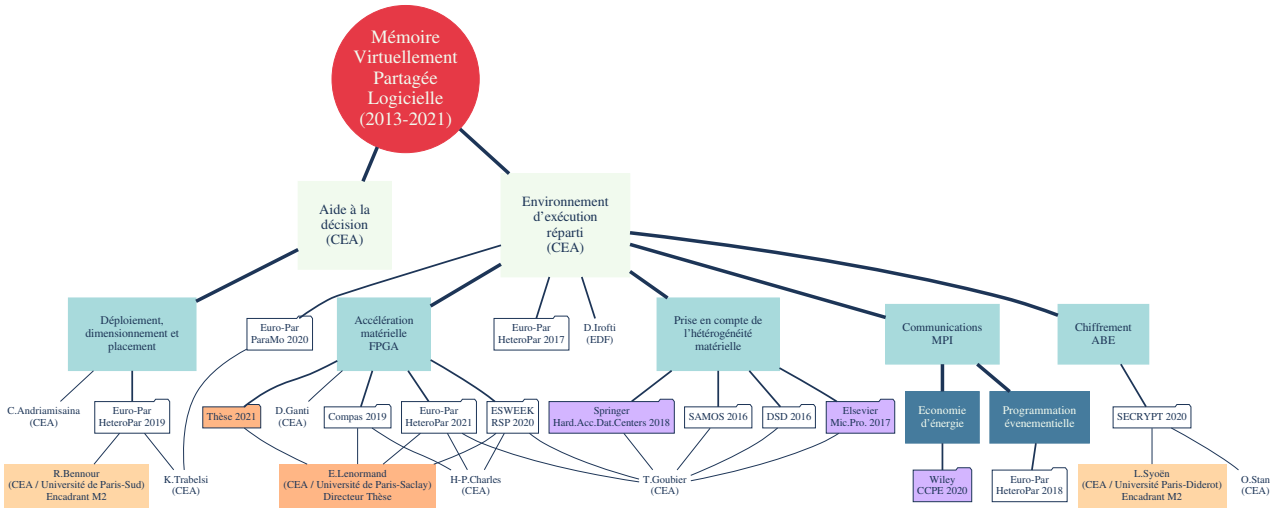
Unification des mémoires physiquement réparties. L'une des principales difficultés pour la programmation d'une plateforme telle que présentée en figure 3.1 est la gestion des données partagées entre les nœuds de calcul d'une part et entre les processeurs hôtes et les accélérateurs d'autre part. Classiquement, ces plateformes sont exploitées selon un modèle de programmation hybride constitué d'une couche de gestion haut-niveau pour la coordination des nœuds et de programmes spécifiques déployés sur chaque nœud prenant en compte le modèle de programmation et l'environnement de développement utilisé par l'accélérateur. Ce choix logiciel nécessite un effort de programmation non négligeable entraînant une mise au point du code plus difficile, avec le risque d'obtenir des performances dégradées en effectuant des choix peu avisés sur le placement des données ainsi que des copies mémoires inutiles. La programmation hybride requiert donc une expertise pointue dans différents langages ainsi qu'une connaissance fine de l'architecture matérielle ciblée. Il y a donc un intérêt à rendre la gestion des données transparente pour le développeur, d'unifier les interfaces sur chaque nœud de calcul et de fournir des outils d'optimisation pour orchestrer les transferts de données.

Les mémoires virtuellement partagées logicielles consistent à fédérer des mémoires physiquement réparties sur des nœuds distants. Elles permettent de conserver le modèle de programmation en mémoire partagée au dessus d'une architecture dépourvue de mémoire physique centrale, c'est-à-dire principalement qui ne rentre pas dans le cadre des multi-cœurs et des machines NUMA. Pour cela, un intergiciel est déployé entre d'une part le code métier et d'autre part des bibliothèques de communication ou de passage de message. Les MVP offrent un haut niveau d'abstraction, comme illustré en figure 3.2a : un espace d'adressage logique est rendu accessible pour tous les processus, et cet espace est construit indépendamment des mémoires physiques sous-jacentes. Un deuxième point est que la localisation exacte des données pointées par l'espace logique est géré par la MVP, ce qui inclut la potentielle réplication des données. Ces deux points rendent les systèmes MVP plus génériques que les systèmes PGAS (*Patitioned Global Address Space*) dans lesquels la localisation des données et les transferts entre mémoires restent à la charge du code applicatif.

Les MVP ont connu un développement rapide à la fin des années 80 avec la mise en réseau des stations de travail dans les instituts (les *clusters Beowolf*) et le déploiement de réseaux de communication entre centres de recherche. La MVP IVY [Li, 1988] est souvent citée en exemple comme ayant introduit des concepts toujours utilisés dans les implémentations modernes. Cependant, il apparaît rapidement que ces systèmes souffrent d'un problème de passage à l'échelle au delà de quelques dizaines de nœuds. Dans cette période et jusqu'à la fin des années 90, de nombreuses MVP sont proposées [Nitzberg and Lo, 1991] : les innovations ne portent pas tant sur le concept lui-même, mais bien souvent sur le protocole de cohérence des données, dont le comportement détermine les performances du système à grande échelle. C'est par exemple le cas avec les modèles de cohérence relâchés implémentés dans les MVP Munin [Carter et al., 1991], Midway [Bershad et al., 1993] et TreadMarks [Keleher et al., 1994]. Une autre problématique classiquement rencontrée à l'époque concerne la représentation des entiers (l'*endianness*) pouvant différer d'une machine à l'autre et source de contributions [Zhou et al., 1992, Rinard et al., 1992] pour prendre en compte cette forme d'hétérogénéité (les MVP concernées partagent des pages système, et non des objets). Dans les années 2000, avec l'introduction du concept de grille de calcul informatique [Foster and Kesselman, 2004] mutualisant les ressources des centres de recherche à l'échelle d'un continent, de nouveaux systèmes MVP sont proposés [Antoniu et al., 2005, Nicolae et al., 2011] faisant cette fois l'hypothèse de ressources homogènes. Enfin, quelques systèmes à mémoire partagée ciblent des architectures hétérogènes composées d'un processeur hôte et d'un accélérateur [Gelado et al., 2010, Lin et al., 2012], sans pour autant généraliser à une plateforme distribuée. Les contributions suivantes s'inscrivent dans l'évolution des architectures de calcul, de la grappe de calculateurs au micro-serveur. Les aspects hétérogènes et embarqués introduisent de nouvelles hypothèses de travail et relancent les recherches sur l'usage des MVP dans ce contexte.

Contributions pour la programmabilité des machines hétérogènes réparties. Dans la ligne directe des projets européens H2020 FiPS et M2DC portant sur la conception de micro-serveurs hétérogènes, je propose en 2016 d'évaluer la pertinence de l'emploi d'une MVP logicielle afin d'abstraire la complexité des machines. Cette initiative conduit à l'implémentation d'une MVP complète dénommée SAT pour *Share Among Things*, ainsi qu'à la réalisation de nombreux travaux satellites et collaboratifs reposant sur le système développé. Au final, sur la période 2016 à 2020, ces travaux donneront lieu à une dizaine de publications dans des conférences nationales et internationales, ainsi qu'à un article de journal. Parmi les thématiques abordées, certaines traitent de la conception de SAT [Cudennec, 2017] et de son évaluation [Cudennec and Trabelsi, 2020], du modèle de programmation hybride en mémoire partagée et événementiel [Cudennec, 2018], de l'intégration d'une méthode de chiffrement par attributs [Stan et al., 2020], de l'exploration

des configurations de déploiement sur machine hétérogène [Trabelsi et al., 2019], de l'efficacité énergétique des communications [Cudennec, 2020] et encore de l'intégration d'un FPGA dans la MVP [Lenormand et al., 2021]. SAT servira de plus de support technique pour la thèse d'Erwan Lenormand [Lenormand, 2022], soutenue en 2022, que j'ai proposée et co-dirigée avec Henri-Pierre Charles (CEA), et dont le sujet porte sur la simplification de l'utilisation des FPGA à l'aide de mémoires partagées.



3.1 Une MVP pour micro-serveur hétérogène

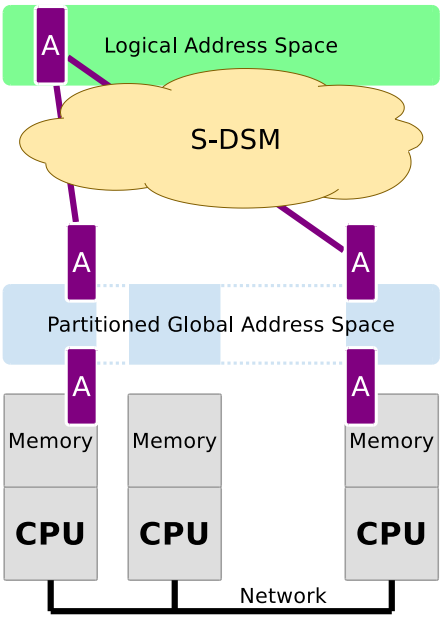
> Cudennec, L. (2017). Software-distributed shared memory over heterogeneous micro-server architecture. In Heras, D. B., Bougé, L., Mencagli, G., Jeannot, E., Sakellariou, R., Badia, R. M., Barbosa, J. G., Ricci, L., Scott, S. L., Lankes, S., and Weidendorfer, J., editors, Euro-Par 2017 : Parallel Processing Workshops - Euro-Par 2017 International Workshops, Santiago de Compostela, Spain, August 28-29, 2017, Revised Selected Papers, volume 10659 of Lecture Notes in Computer Science, pages 366–377. Springer.
>
> [Cudennec, 2017] CEA LIST.
>
> Contributions | ■ initiative ■ idées ■ implémentation ■ expérimentations ■ rédaction ■ présentation

Le développement des activités sur la cohérence des données pour les architectures many-cœurs présentées dans le chapitre 2 a démontré l'importance de disposer de moyens d'évaluation et d'expérimentation au plus proche des plateformes cibles. La MVP que j'ai conçue et implémentée, dénommée SAT (pour *Share Among Things*), est à l'origine pensée pour dérouler fonctionnellement le comportement des protocoles de cohérence sur le principe d'un simulateur dirigé par les évènements. Mais à l'arrivée c'est une mémoire virtuellement partagée logicielle complète qui a été développée, reposant sur les techniques d'algorithmique répartie. SAT répond à deux objectifs : 1) permettre la programmation d'architectures de calcul hétérogènes et distribuées, à cheval entre le monde HPC et l'embarqué et 2) offrir un socle commun pour mener des développements dans le cadre de travaux de recherche sur le partage de données.

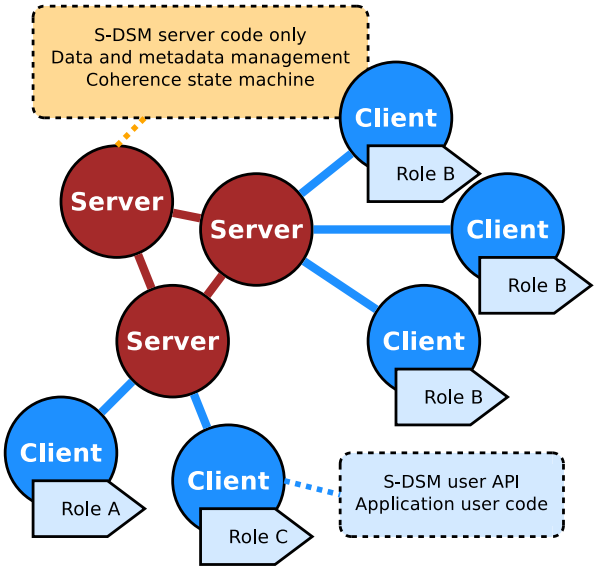
Pour le premier objectif, la MVP SAT a permis de déployer diverses applications sur la machine hétérogène distribuée Christmann RECS3, ainsi que sur des machines hétérogènes embarquées plus exploratoires, à des fins de démonstration, notamment au forum Teratec et au salon ISC en 2018. Ces démonstrations consistent à effectuer du traitement vidéo en temps réel en utilisant différentes configurations matérielles composées de nœuds de calcul haute-performance, de nœuds basse-consommation et d'accélérateurs. Ces configurations peuvent être changées dynamiquement et l'application est en mesure d'afficher les nouvelles performances de traitement (en images par seconde) et de consommation instantanée (en watts). L'application a été développée sur SAT sans aucune considération de la plateforme matérielle, en reposant sur le haut degré d'abstraction offert par le paradigme de programmation en mémoire partagée. Un deuxième objectif atteint par SAT consiste à offrir un environnement exploratoire pour différents projets de R&D. C'est le cas par exemple pour les travaux visant à intégrer des FPGA dans les systèmes distribués, effectués dans le cadre de la thèse d'Erwan Lenormand (section 3.2.3). Dans ces travaux, SAT est utilisée pour reconstituer des flux de données à partir de données partagées massives et ainsi alimenter les FPGA malgré leur mémoire physique embarquée limitée. Un autre exemple est l'étude de systèmes d'aide à la décision pour le dimensionnement et la configuration d'applications distribuées en présence de contraintes multicritères (le compromis performance-énergie). Dans ce travail SAT est utilisée pour évaluer les performances des solutions explorées par le système de décision, de par la grande malléabilité des tâches sur la MVP (à l'instar des *threads* classiques) et la facilité de reconfiguration du déploiement. De nombreuses instances de SAT ont pu être lancées afin de construire une base de connaissance permettant de caractériser le comportement énergétique des applications sur des architectures hétérogènes distribuées comme nous le verrons dans la section 3.2.1. Dans la suite de chapitre, nous présentons différentes contributions construites autour de SAT et qui auraient été difficilement réalisables sans la maîtrise complète de la MVP tant sur le plan théorique que sur le plan de l'implémentation.

3.1.1 Éléments de conception de la MVP.

> Cudennec, L. (2020). Software-distributed shared memory for heterogeneous machines : Design and use considerations. CoRR, abs/2009.01507.
>
> [Cudennec, 2020] CEA LIST, DGA MI.
>
> Contributions | ■ initiative ■ idées ■ implémentation ■ expérimentations ■ rédaction



(a) Espace d'adressage logique projeté sur les mémoires physiques.



(b) Topologie logique de la MVP sous forme de réseau super-pair semi-structuré.

FIGURE 3.2 – Principe et organisation de la MVP développée pour les machines hétérogènes.

Prendre la décision de concevoir et implémenter un système distribué aussi complexe qu'une MVP soulève des questions similaires à celles évoquées dans la section 2.1 avec la réalisation d'une chaîne de compilation pour un nouveau langage de programmation. Ce choix découle de plusieurs éléments de motivation : 1) la capacité à concevoir une MVP sur mesure, en ligne avec les objectifs thématiques de l'institut LIST au CEA (adéquation logiciel-matériel, efficacité énergétique, embarquabilité, hétérogénéité), 2) la maîtrise du code et la possibilité de disposer d'une brique technologique en dehors de toute propriété intellectuelle externe et 3) la montée en compétence pour rejoindre l'état de l'art sur le partage de données en environnement distribué. Le développement des fonctionnalités essentielles de SAT a nécessité 1,5 années-homme financées sur le projet H2020 M2DC [Oleksiak et al., 2017], dont la moitié est consacrée à la mise au point des algorithmes distribués pour la synchronisation et la gestion de la cohérence des données. Le système résultant est caractérisé par les points suivants.

- **Un modèle de tâches parallèle**, en mémoire partagée, similaire aux *threads* Posix, dans lequel les calculs sont organisés sous forme d'un ensemble de processus pouvant accéder à un espace logique commun. Les processus jouent différents rôles définis par le développeur et chaque rôle peut être instancié plusieurs fois lors du déploiement,
- **La gestion transparente de l'espace logique**, ce qui inclut la localisation et le transfert des données ainsi que la possible réplication de celles-ci sur différents nœuds (figure 3.2a).
- **Un modèle pour l'accès aux données partagées** inspiré de la cohérence à l'entrée [Bershad et al., 1993], dans lequel les accès sont conditionnés par l'acquisition d'un verrou en lecture, en écriture ou en lecture-écriture. Ce modèle garantit que les données sont disponibles et dans un état cohérent, en accord avec le modèle de cohérence choisi, dans la portée de code encadrée par la prise de verrou (*acquire*) et sa libération (*release*). En dehors de cette portée, la donnée est considérée dans un état indéterminé,
- **Des objets et opérations collectives** permettant la synchronisation des tâches tels que les rendez-vous, les barrières et les sémaphores,
- **Une conception modulaire** pour faciliter l'ajout de nouveaux protocoles de cohérence et de nouvelles fonctionnalités. SAT est une MVP multi-protocoles de cohérence, dans laquelle il est possible de déployer plusieurs protocoles en même temps pendant l'exécution d'une application et d'assigner dynamiquement un protocole spécifique à une donnée partagée lors de son allocation,
- **Une implémentation portable** permettant le déploiement sur différents types de machines opérées par différents systèmes d'exploitation. SAT représente 10 000 lignes de code ANSI C, reposant sur l'API minimale de

MPI (envoyer et recevoir des messages) et s'exécutant au niveau utilisateur. Ces choix techniques permettent de faciliter la compilation de SAT sur différents systèmes et de choisir une implémentation de MPI adaptée à l'infrastructure (SAT a été déployée sur les implémentations Open MPI [Gabriel et al., 2004], MPICH [Thakur and Gropp, 2007] et MPC [Pérache et al., 2008]),

— **Une organisation logique transparente** pour l'application, du déploiement à la terminaison : les intergiciels MPI et SAT prenant en charge les phases de démarrage (*bootstraping*), de construction du réseau logique (*overlay*) et de propagation du front de terminaison.

Plusieurs éléments sur la conception de SAT sont décrits dans le rapport [Cudennec, 2020] et seuls les points nécessaires à la compréhension des contributions présentées dans ce manuscrit sont donnés ici.

Dans les grandes lignes, SAT est une MVP logicielle qui se présente comme un intergiciel entre le programme métier et l'environnement de communication de type MPI. En pratique, c'est donc un programme MPI, formé d'une couche utilisateur faisant appel à l'API de SAT (une interface orientée mémoire partagée), et située au dessus de la couche SAT en charge de traduire ces appels en directives MPI sur le principe d'un automate réparti. L'organisation logique de SAT est un réseau super-pair semi-structuré dans lequel un ensemble de clients sont rattachés à un réseau pair-à-pair de serveurs (figure 3.2b). Les clients exécutent du code utilisateur (un rôle) et ont accès à l'API de SAT. Chaque client est rattaché à un serveur. Les serveurs sont en charge du stockage des données et de la gestion des métadonnées. Cette organisation classique se retrouve dans de nombreux systèmes distribués pour la gestion de fichiers (Ceph [Weil et al., 2006], Gfarm [Tatebe et al., 2002]) ou d'objets partagés (OceanStore [Kubiatowicz et al., 2000], JuxMem [Antoniu et al., 2005]).

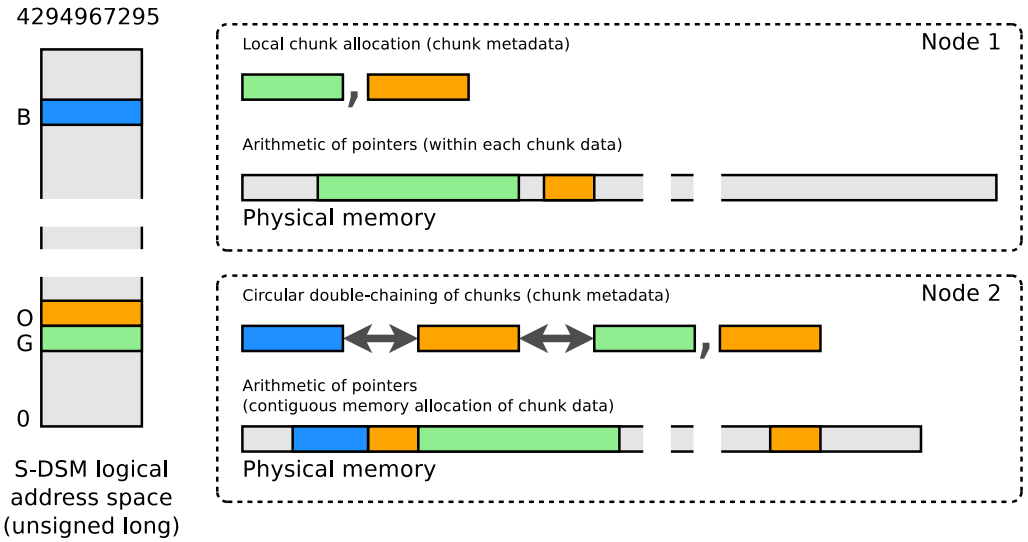


FIGURE 3.3 – Espace d'adressage logique, allocation et stockage en mémoire physique dans la MVP SAT.

Découpage des données en *chunks*. Dans SAT les données sont représentées sous forme de *chunks*, des unités atomiques comprenant les données brutes ainsi que des métadonnées. La figure 3.3 illustre les relations entre l'espace d'adressage logique, l'allocation et le stockage des données en mémoire physique. Sur la gauche, l'espace d'adressage logique est constitué d'un ensemble d'adresses, ici définies par un entier représenté sur 64 bits. Les *chunks* sont identifiés par une unique adresse de cet espace logique. Une donnée partagée peut être décomposée en plusieurs *chunks* de tailles différentes et les adresses de ces *chunks* ne sont pas nécessairement contiguës dans l'espace logique. À droite de la figure sont représentées les allocations en mémoire pour deux nœuds. Sur le nœud 1, les *chunks* G et O sont alloués de manière indépendante et se retrouvent à des adresses arbitraires en mémoire physique. Sur le nœud 2, les *chunks* B, O et G sont alloués d'un seul tenant, sous forme d'une chaîne circulaire double, ce qui implique dans le modèle SAT que les données soient allouées de manière contiguë en mémoire physique. Au niveau applicatif, il est donc possible d'effectuer de l'arithmétique de pointeurs au sein d'un *chunk*, mais aussi sur toute la longueur d'une chaîne de *chunks*. Cette organisation permet de dissocier la granularité des données exprimée par l'application de celle optimisée par le stockage dans la MVP. Elle permet de plus d'exprimer simplement des réorganisations de données telles que les transposées de matrice en allouant une chaîne de *chunks* correspondant à l'ordre dans lesquels les éléments doivent apparaître. En charge alors pour SAT d'effectuer la localisation et le transfert des *chunks* de manière transparente pour l'application.

Modèle de programmation en mémoire partagée. Les *chunks* sont des objets exposés au développeur d'application mais dont la gestion est effectuée par la MVP SAT. Ils font partie intégrante du modèle de programmation. Allouer une donnée dans SAT revient à allouer un ou plusieurs *chunks* (une chaîne de *chunks*). L'extrait de code présenté en

listing 4 montre comment allouer une donnée partagée de taille `size`, régie par le protocole de cohérence `MESI` à l'adresse `chunkid` de l'espace d'adressage logique. Dans le cas d'une chaîne de *chunks*, le *chunk* retourné par la primitive **MALLOC** est le premier de la chaîne (aussi nommé *head*). Les primitives **WRITE** et **RELEASE** délimitent une section de code dans laquelle le pointeur `chunk->data` est initialisé et les données pointées sont considérées comme cohérentes. En dehors de cette portée, ce pointeur ne doit pas être accédé car l'intergiciel est susceptible de le désallouer afin de rationaliser la mémoire physique locale. Ce type d'accès aux données partagées est le plus courant, mais il existe d'autres méthodes décrites dans la documentation [Cudennec, 2020], notamment 1) la projection d'une zone mémoire locale dans l'espace logique de SAT, en utilisant la primitive **MAP** qui prend en paramètre un pointeur fourni par l'utilisateur pour renseigner `chunk->data`. Ce champ reste alors accessible en dehors des portées de cohérence, sans pour autant offrir des garanties sur la cohérence des données qui y sont écrites. Et 2) la possibilité de construire une table des symboles effectuant une association entre une chaîne de caractères quelconque (pouvant représenter un UUID, un chemin dans un système de fichiers…) et une adresse dans l'espace logique de SAT. Dans ce cas, la MVP est en mesure de choisir les adresses des *chunks* et donc d'apporter un niveau d'abstraction supplémentaire au dessus de l'espace d'adressage logique.

Listing 4 – Allocation et accès à une donnée partagée dans la MVP SAT.

```
1 Consistency_t * consistency = newHomeBaseMESI() ;
2 Chunk_t * chunk = MALLOC(consistency, chunkid, N * sizeof(unsigned int));
3 unsigned int i = 0;
4
5 WRITE(chunk)
6  for(i = 0; i < N; i++) {
7   chunk->data[i] = i;
8  }
9 RELEASE(chunk)
```

Les primitives de synchronisation permettent d'effectuer des opérations collectives nécessaires au bon déroulement d'une application distribuée qui doit respecter certaines relations de causalité. Ces primitives incluent les barrières (tous les processus doivent atteindre cette instruction avant de poursuivre), les rendez-vous (n sur m processus, $n \leq m$, doivent atteindre cette instruction avant que les m processus puissent poursuivre) et les signaux (un processus émet un signal pour autoriser les processus endormis sur ce signal à poursuivre). Ces algorithmes ont été implémentés en suivant les recommandations formalisées dans [Raynal, 2013]. Un exemple de schéma de synchronisation très répandu est le producteur-consommateur cadencé. Dans ce schéma, un processus producteur écrit sur une variable partagée des valeurs qui sont lues par un processus consommateur. Sans cadencement, l'ordre entre les écritures et les lectures est considéré indéterminé et chaque exécution peut générer un entrelacement différent. Dans ce cas il n'est pas possible de garantir que le consommateur puisse lire toutes les valeurs écrites par le producteur. En utilisant deux objets de synchronisation de type rendez-vous, il est possible de cadencer les lectures et les écritures : un premier rendez-vous est utilisé pour s'assurer que le producteur a bien écrit une nouvelle valeur avant d'autoriser le consommateur à lire la donnée partagée. Un deuxième rendez-vous est utilisé pour s'assurer que le consommateur a bien lu la donnée partagée avant d'autoriser le producteur à écrire une nouvelle. Bien que peu optimisé, ce schéma est très connu car il permet notamment d'implémenter des canaux de communication FIFO en mémoire partagée entre des processus, par exemple pour une application flot-de-données.

Cependant, si l'usage des objets de synchronisation s'avère être efficace pour exprimer des schémas statiques et réguliers tels que le maître-esclave, ils sont en revanche peu adaptés au développement d'applications plus dynamiques, par exemple en présence d'un nombre de processus variable et au comportement volatile, comme c'est le cas pour un modèle basé sur des services. Dans ce cas, le développeur doit créer et gérer dynamiquement des objets de synchronisation au gré des requêtes, ce qui constitue une tâche complexe et l'éloigne du code métier. C'est la raison pour laquelle un modèle de programmation événementiel est proposé pour étendre le modèle en mémoire partagée de SAT.

3.1.2 Modèle de programmation événementiel

> Cudennec, L. (2018). Merging the publish-subscribe pattern with the shared memory paradigm. In Mencagli, G., Heras, D. B., Cardellini, V., Casalicchio, E., Jeannot, E., Wolf, F., Salis, A., Schifanella, C., Manumachu, R. R., Ricci, L., Beccuti, M., Antonelli, L., Sánchez, J. D. G., and Scott, S. L., editors, Euro-Par 2018 : Parallel Processing Workshops - Euro-Par 2018 International Workshops, Turin, Italy, August 27-28, 2018, Revised Selected Papers, volume 11339 of Lecture Notes in Computer Science, pages 469–480. Springer.
>
> [Cudennec, 2018] CEA LIST.
>
> Contributions | ■ initiative ■ idées ■ implémentation ■ expérimentations ■ rédaction ■ présentation

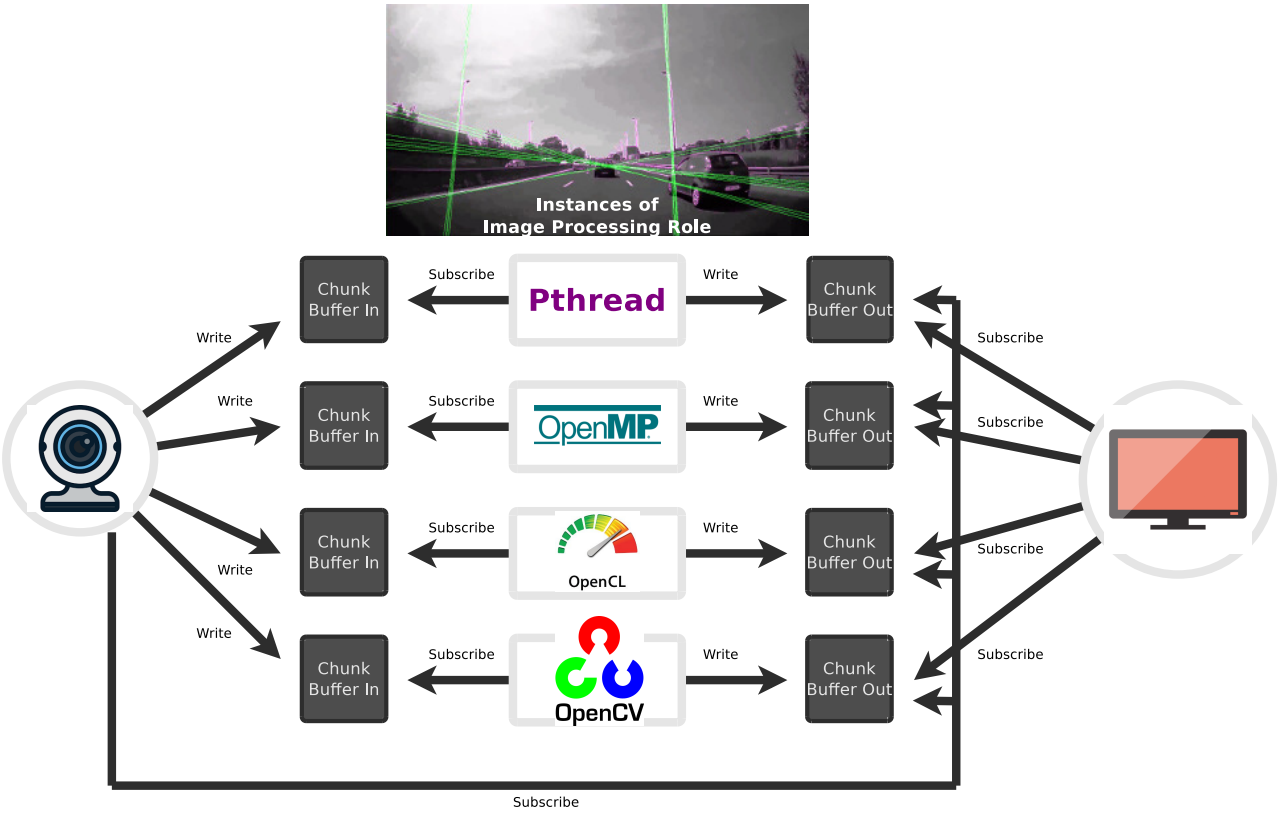


FIGURE 3.4 – Application de traitement de flux vidéo *videostream* implémentée sur la MVP SAT en tirant partie du modèle de programmation hybride mémoire partagée/programmation événementielle. Une tâche de décodage d'un flux vidéo écrit les images dans des tampons alloués dans la MVP et partagés avec les tâches de traitement. Ces tâches sont notifiées d'une nouvelle image à traiter puis écrivent le résultat dans un tampon partagé en sortie. Le cadencement du traitement et l'ordonnancement des calculs sur les tâches sont obtenus par construction (sans code utilisateur spécifique) en se reposant sur les mécanismes de synchronisation du protocole de cohérence des données et du modèle événementiel.

Dans le modèle de programmation événementiel PS (*Publish-Subscribe*), les lecteurs s'abonnent auprès d'un objet partagé et sont ensuite notifiés dès lors qu'un écrivain effectue une modification sur cet objet. Le modèle est par nature dynamique et volatile : les lecteurs peuvent s'abonner et se désabonner pendant l'exécution de l'application. Ce modèle repose sur des synchronisations relâchées et asynchrones, contrairement aux synchronisations fortes et bloquantes implémentées dans le modèle à mémoire partagée : Les processus peuvent effectuer des calculs et recevoir les notifications en parallèle, pour être traitées en même temps ou à la fin du calcul, ce qui apporte un degré de parallélisme supplémentaire pour les tâches.

Si les systèmes à MVP et les systèmes événementiels ont été largement étudiés dans leurs domaines applicatifs respectifs, une approche hybride offrant ces deux modèles de programmation au sein d'un même système n'apparaît pas de manière aussi directe dans la littérature. Certains protocoles de cohérence des données, notamment basés sur des politiques d'invalidation et de mise-à-jour, présentent des mécanismes événementiels : les mémoires détenant une copie d'une donnée reçoivent des notifications, ce qui s'apparente à un abonnement. Cependant, les actions déclenchées suite à ces notifications sont déterminées par le protocole de cohérence et ne sont donc pas modifiables pas l'application. La partie événementielle est donc un mécanisme sous-jacent, et non pas un élément constitutif du modèle de programmation. Dans [Mazzucco et al., 2009] une MVP est implémentée en utilisant le modèle de programmation événementiel de JAVA, ce qui est là encore un mécanisme sous-jacent et non une hybridation des modèles.

Les motivations pour fusionner les modèles à mémoire partagée et événementiels sont multiples : un premier exemple est le couplage de code entre d'une part des nœuds de calcul HPC sur lesquels un code de calcul numérique s'exécute, et d'autre part une infrastructure orientée service utilisée pour surveiller l'avancement des calculs ou permettre à des applicatifs tierces d'interagir avec la simulation. Un exemple concret est donné dans le projet H2020 Lexis [Goubier et al., 2020b] dans lequel un code de simulation de propagation de tsunami (écrit pour machines à mémoire partagée NUMA) est optimisé pour remonter des alertes au plus tôt dans les régions géographiques concernées, en se basant sur des compromis entre le temps de calcul et la précision de la simulation. Un couplage de code naturel s'opère donc entre la simulation numérique et le système d'alerte événementiel. Un deuxième exemple est le déploiement de nouvelles architectures pour l'automobile et les véhicules autonomes, comme étudiées dans le projet FACE [Design Spot, 2018], un projet commun entre le CEA et l'alliance Renault-Nissan-Mitsubishi. Ces architectures sont constituées d'un ensemble de calculateurs hétérogènes sur lesquels des applications critiques et non-critiques cohabitent pour apporter des fonctionnalités liées à la conduite autonome et au divertissement. Dans ce contexte applicatif, des codes numériques s'exécutent sur des accélérateurs et des modèles événementiels sont utilisés entre les nœuds, comme cela est le cas avec l'intergiciel SOME/IP [Völker, 2013] intégré dans la norme automobile AUTOSAR.

L'introduction du modèle PS dans SAT repose sur une idée simple : un *chunk* est un objet partagé mutable sur lequel les tâches peuvent s'abonner et être ainsi notifiées dès lors que l'une d'entre elles effectue une écriture -selon le modèle de programmation en mémoire partagée- sur ce *chunk*. L'interface de programmation proposée par SAT comprend une primitive d'abonnement **SUBSCRIBE** prenant en paramètres un *chunk* ainsi qu'un pointeur vers une fonction *handler* à appeler lorsque ce *chunk* est modifié. Une seconde primitive permet à l'inverse de se désabonner des modifications opérées sur un *chunk*. Si l'interface utilisateur s'avère être simple, la complexité de l'approche est en contrepartie reléguée dans l'intergiciel système. Et cette complexité détermine certains choix effectués dans le modèle de calcul.

- Les notifications sont enregistrées dans une file puis traitées séquentiellement. Ce mécanisme interdit d'appeler la fonction utilisateur *handler* lors d'une notification si le code utilisateur principal ou une autre fonction *handler* est en cours d'exécution. Ce choix a pour conséquence de limiter le parallélisme dans les processus, mais il permet une implémentation plus robuste dans l'intergiciel SAT et offre un modèle de calcul plus simple pour l'utilisateur. La raison principale est que le parallélisme dans les processus est susceptible d'introduire un entrelacement complexe des requêtes émises et des résultats reçus par le protocole de cohérence qui doit alors arbitrer localement à quel fil d'exécution transmettre les données et les privilèges d'accès, ce qui revient à concevoir un protocole de cohérence hiérarchique. Un tel protocole hiérarchique est coûteux à développer et difficile à mettre au point.
- Les notifications ne contiennent pas la donnée modifiée. Le mécanisme de notification fonctionne donc en mode *pull*, c'est à dire que le processus notifié doit effectuer une demande d'accès au *chunk* de sa propre initiative, dans la fonction *handler* par exemple (si cet accès est requis). Le mode *push* consiste à joindre la donnée modifiée avec la notification. Ce mode pose des problèmes de cohérence car il n'est pas encadré par un accès classique en mémoire partagée, et donc par une prise de verrou dans une portée de code. Dès lors, la donnée transférée échappe à toute relation de causalité en lien avec les autres évènements de cohérence, et il n'est pas possible de garantir un niveau de fraîcheur sur la version de la donnée lors de sa mise à disposition en mode *push*. Pour cette raison, ce mode de notification n'est pas proposé dans SAT. Un mode hybride peut être proposé qui consiste à associer à une notification la prise de verrou. Dans ce mode, le code utilisateur appelé dans le *handler* s'exécute avec l'assurance que le *chunk* est déjà accessible, ce qui permet de simplifier le protocole en supprimant une étape. La contrepartie est cependant d'introduire du délai dans le mécanisme, la notification ne pouvant être délivrée au code utilisateur tant que la prise de verrou sur le *chunk* n'est pas effective, ce qui peut s'avérer être inadapté dans le cas d'un système nécessitant de la réactivité..

L'application de traitement de flux vidéo *videostream* est une base de code permettant de prendre un flux vidéo en entrée (en ligne, à partir d'une caméra ou hors-ligne, à partir d'un fichier), d'appliquer un traitement d'image sur chaque *frame*, puis d'écrire la séquence obtenue dans un fichier ou de l'afficher sur un écran. Cette base de code a été portée sur la MVP SAT, en tirant partie du modèle de programmation hybride en mémoire partagée et événementiel. Le principe, illustré en figure 3.4, est le suivant : pour chaque instance de rôle de traitement d'image, un tampon partagé est déclaré pour contenir l'image d'origine et un tampon partagé est déclaré pour contenir le résultat du traitement. Ces tampons sont donc implémentés par un *chunk* ou par une chaîne de *chunks* dans SAT. Deux implémentations sont évaluées : 1) une implémentation en mémoire partagée (dénommée RR) dans laquelle des rendez-vous sont utilisés pour synchroniser les productions et consommations, ce qui donne lieu à une répartition équitable des images sur les tâches de traitement et suivant un ordonnancement de type tourniquet. 2) une implémentation hybride (dénommée PS et représentée en figure 3.4) dans laquelle les tampons sont accédés en écriture selon le modèle en mémoire partagée classique et accédés en lecture selon le modèle événementiel. Une boucle de rétropropagation est obtenue en abonnant la tâche d'acquisition aux tampons de sortie des tâches de traitement. Dès lors, lorsqu'une tâche de traitement écrit un résultat sur son tampon de sortie, une notification est envoyée par SAT à la tâche d'acquisition qui peut alors

Topologie	A	B	C	D	E	F
RR - Round-Robin (s)	220	177	286	233	221	286
PS - Publish-Subscribe (s)	218	153	401	359	209	198
Consommation (watts)	58	58	85	114	114	114

TABLE 3.1 – Temps de calcul en secondes et consommation instantanée estimée en watts pour différentes topologies (A-F) de l'application de traitement vidéo implémentée sur SAT en utilisant les objets classiques de synchronisation en mémoire partagée (un ensemble de rendez-vous permettant la distribution des images sur les tâches de traitement selon la méthode du tourniquet *Round-Robin*) et en utilisant le modèle de programmation événementiel *publish-subscribe*. Résultats obtenus sur la machine Christmann RECS|Box 3 présentée en figure 3.1a, en utilisant jusqu'à 8 processeurs ARM Cortex A15 et 2 processeurs Intel Core i7.

décider d'écrire une nouvelle image dans son tampon d'entrée, ce qui constitue un cadencement par construction. La répartition des images sur les tâches de traitement s'effectue alors en fonction de la disponibilité de ces tâches, sur le modèle de l'ordonnancement glouton. Cette approche est intuitivement adaptée aux architectures de calcul hétérogènes distribuées, dans lesquelles les unités de calcul effectuent les traitements à des vitesses différentes.

Pour vérifier cette intuition, l'application *videostream* a été évaluée dans les versions RR et PS sur le micro-serveur hétérogène Christmann RECS3, composé de 8 nœuds ARM Cortex A15 et 2 nœuds Intel Core i7. La table 3.1 présente les résultats obtenus pour différentes topologies (A-F) définies par le nombre de tâches de traitement, le nombre de serveurs SAT et la projection de ces instances sur les ressources de calcul (les détails sont donnés dans [Cudennec, 2018]). La consommation instantanée en watts induite par la topologie est indiquée à titre indicatif. Le temps de calcul est donné en secondes pour les configurations RR et PS. La figure 3.5 complète ces résultats en indiquant la répartition des images sur les tâches de traitement. Une première conclusion est qu'il existe des configurations dans lesquelles le modèle événementiel PS n'est pas la meilleure solution. Ce résultat contre-intuitif est illustré par les topologies C et D, dans lesquelles les communications induites par les messages de contrôle du modèle événementiel deviennent problématiques : les liens réseaux entre les nœuds ARM sont basés sur le protocole USB2 (un contrôleur partagé par 4 processeurs ARM) et souffrent d'une latence élevée ($\sim$1 ms), ce qui dégrade les performances en générant des temps d'attente au niveau des tâches de traitement. A l'inverse, l'implémentation RR permet de trouver un régime établi dans l'entrelacement des communications (en comparaison du comportement dynamique de PS), ce qui se traduit par des performances nettement meilleures.

Une deuxième conclusion est que la distribution de la gestion des données et des métadonnées de la MVP sur plusieurs serveurs SAT permet de contrebalancer les faibles performances des liens réseaux. Ce cas est illustré par les topologies E et F, toutes deux basées sur la topologie D mais avec respectivement 2 et 4 serveurs SAT là où la topologie D n'en contient qu'un. L'augmentation du nombre de serveurs SAT s'accompagne d'une amélioration significative du temps de calcul pour l'implémentation PS, portée par une meilleure répartition des messages de contrôle du modèle événementiel entre les nœuds. À l'inverse, les performances de l'implémentation RR diminuent, l'utilisation de plusieurs serveurs SAT s'accompagnant d'un nombre de rebonds (*hops*) plus grand pour certaines requêtes de cohérence.

En conclusion, le modèle de programmation événementiel couplé au modèle de programmation en mémoire partagée apporte une couche d'abstraction supplémentaire quant à la gestion des synchronisations et des accès aux données. Cette approche est particulièrement adaptée pour la programmation des architectures hétérogènes qui par nature demandent plus de flexibilité dans la gestion des tâches et des données contrairement à des architectures homogènes dans lesquelles les calculs peuvent être déployés de manière plus uniforme. Dans certaines configurations, cette contribution permet d'obtenir de meilleures performances calculatoires en simplifiant le motif de communication.

La programmation événementielle est aussi utilisée dans le cadre d'une application de réseau de capteurs visant à offrir un consensus-ϵ [Irofti, 2020], c'est-à-dire un consensus dans lequel les participants ne s'accordent pas sur une valeur unique, mais font en sorte que les valeurs locales convergent en un temps fini. L'implémentation du consensus sur SAT repose sur le mécanisme événementiel pour faire en sorte que les voisins d'un capteur soient notifiés d'une nouvelle valeur et ainsi affiner leur prédiction. Des expérimentations ont été menées en collaboration avec Dîna Irofti (EDF) sur des bases de données d'AirParif dans lesquelles la perte d'un ou plusieurs capteurs est simulée pour observer la résilience du système. Un deuxième exemple est la possibilité d'utiliser le système événementiel pour faciliter les interactions entre la MVP logicielle et une extension déployée sur un FPGA. Ce cas d'utilisation fait l'objet des travaux de thèse d'Erwan Lenormand (CEA) que je co-dirige avec Henri-Pierre Charles (CEA) et qui sont décrits dans la section 3.2.3.

Ces exemples applicatifs montrent qu'au delà d'offrir un intergiciel de gestion de données partagées, SAT a permis d'effectuer une réflexion sur le modèle de programmation, un élément déterminant pour l'adhésion des développeurs. Cependant, l'abstraction offerte au niveau de l'API est obtenue en contrepartie du déploiement d'un intergiciel complexe, souvent accompagné de surcoûts principalement dans les communications et pouvant entraîner une baisse des performances applicatives.

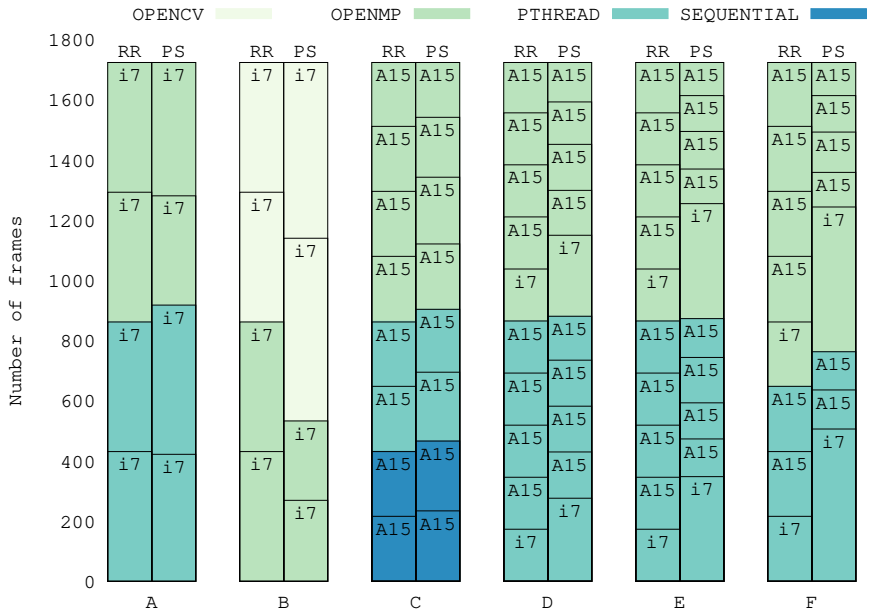


FIGURE 3.5 – Répartition du nombre d'images traitées sur chaque tâche de l'application de traitement vidéo implémentée sur SAT en utilisant la mémoire partagée (répartition équitable selon la méthode du tourniquet, RR) et son extension au modèle de programmation événementiel (répartition à la demande selon le modèle *publish-subscribe*, PS), pour différentes topologies (A-F). Résultats obtenus sur la machine Christmann RECS|Box 3 présentée en figure 3.1a, en utilisant jusqu'à 8 processeurs ARM Cortex A15 et 2 processeurs Intel Core i7. Pour chaque tâche de traitement, l'implémentation déployée est indiquée par un code couleur.

3.1.3 Comparaison des performances de la MVP avec MPI et ZeroMQ

> Cudennec, L. and Trabelsi, K. (2020). Experiments using a software-distributed shared memory, MPI and 0mq over heterogeneous computing resources. In Balis, B., Heras, D. B., Antonelli, L., Bracciali, A., Gruber, T., Hyun-Wook, J., Kuhn, M., Scott, S. L., Unat, D., and Wyrzykowski, R., editors, Euro-Par 2020 : Parallel Processing Workshops - Euro-Par 2020 International Workshops, Warsaw, Poland, August 24-25, 2020, Revised Selected Papers, volume 12480 of Lecture Notes in Computer Science, pages 237–248. Springer.
>
> [Cudennec and Trabelsi, 2020] CEA LIST, DGA MI.
>
> Contributions | ■ initiative ■ idées ■ implémentation ■ expérimentations ■ rédaction ■ présentation

Dans la littérature, les mémoires virtuellement partagées logicielles sont régulièrement décriées pour les faibles performances obtenues en comparaison de systèmes de programmation plus proches de la machine. Plusieurs études montrent en effet qu'une application écrite sur une MVP est significativement plus lente qu'une même application écrite sur un intergiciel de passage de messages. La table 3.2 regroupe un certain nombre de ces études, du début des années 90 à aujourd'hui. À la fin des années 1980, [Li, 1988] montre que la MVP Ivy, système précurseur dans les grappes de calculateurs, ne passe pas à l'échelle au delà de quelques dizaines de nœuds, ce qui limite son utilisation à des petits déploiements et ne permet pas d'obtenir des accélérations parallèles importantes. Dans la même temporalité, au début des années 1990, le standard MPI [Walker, 1992] est proposé pour programmer les environnements distribués. Le passage de messages devient alors - et reste aujourd'hui encore - le principal modèle de programmation utilisé sur les super-calculateurs, ne laissant que peu de place pour la programmation en mémoire partagée sur des déploiements inter-nœuds.

Depuis une quinzaine d'années, la tendance générale montre que le surcoût introduit par l'utilisation d'un intergiciel complexe tel qu'une MVP tend à s'estomper. Aujourd'hui, les applications déployées sur les MVP à l'état de l'art offrent des performances proches de celles déployées sur du passage par message. Dans certains cas, la MVP permet même d'obtenir de meilleures performances, ce qui est par exemple constaté avec JuxMem [Antoniu et al., 2007] dès 2007 sur la plateforme Grid'5000 [Bolze et al., 2006], ainsi que plus récemment avec Grappa [Nelson et al., 2015] et Argo [Kaxiras et al., 2015] en 2015 sur des grappes de calculateurs interconnectées par des réseaux Mellanox InfiniBand. Dans ces trois travaux, le nombre de noeuds utilisés pour évaluer les performances est raisonnablement grand, de l'ordre de la centaine, et permet d'apprécier le passage à l'échelle. Il reste cependant très en dessous des grands déploiements pouvant être menés sur des super-calculateurs, notamment pour de grandes simulations physiques impliquant des milliers, voire des dizaines de milliers de nœuds. Malgré cela, l'usage de système MVP est désormais envisageable, et est même devenu un élément de réussite déterminant pour certaines architectures, par exemple pour offrir un cache logique unifié dans les puces many-cœurs comme présenté dans la section 2.2. Si la MVP n'est pas candidate pour des

Référence	Système MP	Système MVP	#	Performance de la MVP
[Lu et al., 1995]	PVM	TreadMarks	8	15% of PVM speedup to 5% slower
[Scales and Gharachorloo, 1997]	Oracle database*	Shasta	2	2-4 times slower
[Bader and JáJá, 1999]	MPICH	CVM	8	10 times slower
[Werstein et al., 2003]	PVM, MPI	TreadMarks	32	slower, except for Mandelbrot kernel
[Antoniu et al., 2007]	GridRPC*	JuxMem	53	3 times faster
[Gelado et al., 2010]	CUDA*	GMAC ADSM	2	match
[Dimakopoulos and Hadjidoukas, 2011]	MPI	MOME, MOCHA	16	6-8 times slower
[Lin et al., 2012]	SysLink	Reflex	3	83% less energy, 2.5% slower
[Nelson et al., 2015]	MapReduce*, GraphLab*	Grappa	128	1.3 times faster
[Kaxiras et al., 2015]	MPI	Argo	128	match or exceed
[Beri et al., 2017]	StarPu*, SUMMA*	Unicorn	10	performs quite close

TABLE 3.2 – Deux décennies d'évaluation de performances entre l'utilisation d'une MVP et l'utilisation d'un intergiciel de passage de message (MP). Certains systèmes* sont déployés au dessus de systèmes MP et sont considérés équivalents à une implémentation MP. # indique le nombre de nœuds utilisés dans les expérimentations. Les résultats sont retranscris tels que indiqués dans les papiers cités et ne peuvent faire l'objet d'une comparaison directe en l'absence d'une méthodologie expérimentale commune. Il est cependant possible d'observer la tendance générale des performances obtenues entre les systèmes MVP et MP au fil des avancées technologiques.

déploiements sur de grandes infrastructures, elle devient pertinente pour des systèmes intermédiaires, notamment des architectures hétérogènes pouvant bénéficier d'une couche d'abstraction logicielle visant à masquer la complexité en terme de programmabilité.

Les raisons de cette évolution des performances sont multiples : un premier constat repose sur le développement de réseaux de communication de plus en plus performants, que ce soit à l'échelle des grappes de calcul et des puces dans lesquels les réseaux haute-performance et les NoC bénéficient de technologies innovantes telles que la fibre optique, l'empilement 3D ou la normalisation des accès directs aux mémoires distantes des accélérateurs matériels. Le deuxième constat est l'augmentation de la taille des mémoires physiques et des mémoires caches, permettant aux systèmes MVP d'offrir de grands espaces logiques implémentés dans des mémoires rapides, favorisant la réutilisation des données. Ainsi, à l'origine pénalisée par l'utilisation de protocoles de communication entraînant un grand nombre de messages et des synchronisations bloquantes, la MVP peut désormais être vue comme un grand cache unifié inter-nœuds, sur lequel un modèle de programmation à mémoire partagée est employé.

La comparaison entre une application déployée sur MVP et son équivalent sur MP n'est pas évidente à réaliser : notamment parce qu'il est difficile de déterminer si les performances de deux implémentations effectuées avec des modèles de programmation différents peuvent être rigoureusement comparées. D'inévitables biais dans la réalisation de la démonstration interviennent : les deux implémentations ont-elles reçu le même niveau d'optimisation ? Ont-elles accès aux même capacités matérielles ? Le noyau de calcul est-il adapté aux deux modèles de programmation ? Il n'y a pas à notre connaissance de méthodologie définie et normalisée pour comparer deux modèles de programmation. Ainsi, les comparaisons établies dans l'état de l'art sont le résultat, comme bien souvent, de retours d'expérience obtenus après des cycles de développements et de déploiements effectués dans divers contextes d'utilisation.

Déploiement et comparaison de ZeroMQ, MPI et SAT. La MVP SAT a été utilisée pour déployer des applications parallèles, notamment l'application de traitement vidéo *videostream* présentée en section 3.1.2, sur des architectures hétérogènes distribuées. Dans ce travail [Cudennec and Trabelsi, 2020], cette application a été réécrite en se basant sur les versions C d'OpenMPI et de ZeroMQ, deux implémentations d'intergiciels de passage de messages largement utilisées dans les systèmes HPC et HPEC. ZeroMQ [1] est une bibliothèque récente (2010) visant des applications embarquées et dont l'intergiciel est conçu de manière minimale en comparaison des implémentations classiques de MPI. L'API repose sur des motifs de communication haut-niveau (`request-reply`, `publish-subscribe`, ...) contrairement à l'approche `send-receive` orientée message de OpenMPI. En contrepartie, la construction du réseau logique de ZeroMQ permettant aux processus de communiquer n'est pas transparente et doit être exprimée au niveau applicatif, en renseignant les adresses réseau des processus avec lesquels communiquer, plutôt que de se baser sur des identifiants logiques et des groupes de communication comme le propose MPI. ZeroMQ, OpenMPI et SAT représentent donc trois niveaux croissants d'abstraction de la plateforme matérielle, ce qui usuellement se traduit par une baisse de l'efficacité calculatoire. L'objectif du portage de l'application de traitement vidéo est de comparer les approches en termes de complexité de programmation et de performances calculatoires. La figure 3.6 présente une vue synthétique des résultats obtenus par ces trois implémentations déployées sur les nœuds de la machine hétérogène bac-à-sable présentée en figure 3.1b. Le temps de calcul est mesuré pour trois résolutions d'une même vidéo (UHD, UHD-1 et UHD-2).

1. ZeroMQ `https://zeromq.org`

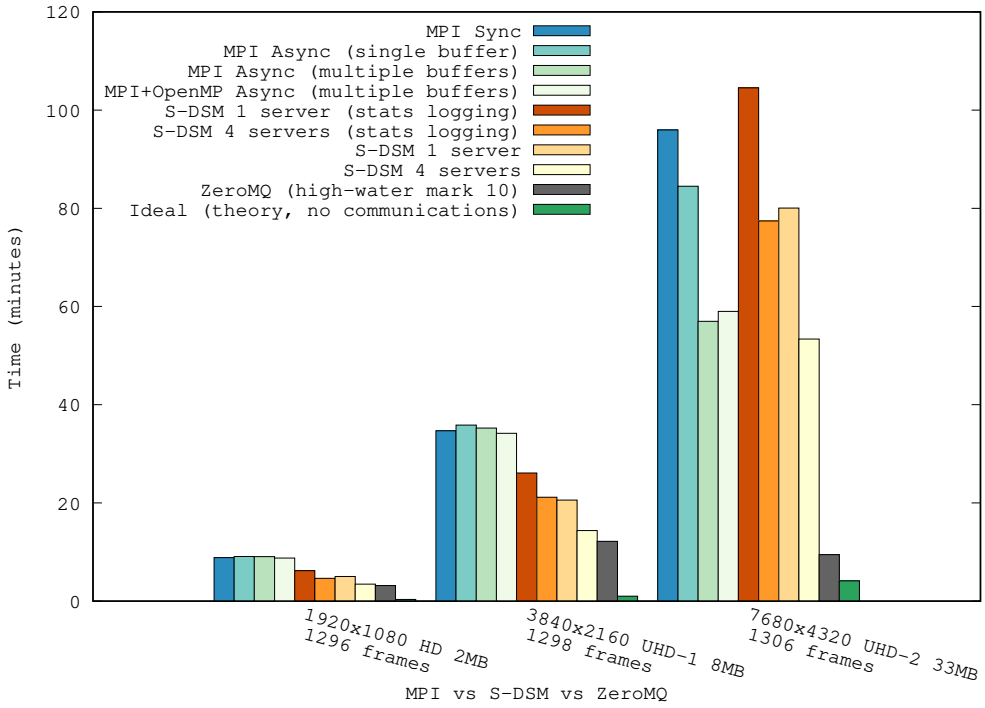


FIGURE 3.6 – Comparaison des temps de calcul mesurés pour l'application de traitement vidéo *videostream* implémentée au dessus de la MVP SAT (S-DSM), de OpenMPI et de ZeroMQ, pour différentes résolutions de flux vidéo (HD, UHD-1 et UHD-2), chaque image non compressée de la vidéo pesant respectivement 2 Mo, 8 Mo et 33 Mo. La plateforme de calcul est une grappe hétérogène présentée en figure 3.1b composée d'un serveur et de cartes de développement embarquées, interconnectées par un réseau Ethernet.

> Les résultats montrent que l'application portée sur ZeroMQ est la plus performante et que la version portée sur la MVP SAT (dont les mécanismes de communication sont implémentés sur OpenMPI) est jusqu'à deux fois plus rapide que celle écrite sur OpenMPI exclusivement.

Différentes variantes du code OpenMPI ont été testées : `MPI Sync` signifie que les primitives de communication sont bloquantes, c'est-à-dire qu'OpenMPI assure que le message a bien été délivré avant de permettre au processus de continuer son exécution. `MPI Async` indique que les primitives de communication sont asynchrones, c'est-à-dire qu'elles ne sont pas bloquantes et permettent de continuer l'exécution pendant la livraison du message. Cette approche nécessite une autre forme de synchronisation dans le code utilisateur afin de ne pas écraser le tampon d'envoi lors d'un prochain appel à la primitive. `single buffer` indique qu'un seul tampon d'envoi est alloué pour transférer les images aux processus de traitement. Enfin, `multiple buffers` indique qu'un tampon partagé est alloué par processus de traitement, ce qui augmente le degré de parallélisme. Ces quatre variantes du code représentent des versions de plus en plus complexes à écrire (gestion de tampons partagés, gestion d'objets de synchronisation entre les fils d'exécution) dans le seul but d'augmenter le degré de parallélisme et d'obtenir de meilleures performances applicatives. Ce travail d'optimisation du code porté sur MPI est aussi l'illustration de l'expertise et de l'effort de développement à fournir pour obtenir une application suffisamment optimisée.

Plusieurs déploiements du code porté sur SAT ont été effectués : ces déploiements consistent notamment à faire varier la topologie de la MVP en augmentant le nombre de serveurs de métadonnées et le placement sur les ressources de calcul. Pour un même code, ces configurations ont une influence importante sur les performances à l'exécution : du simple au double pour le traitement du flux UHD-2 entre SAT en mode journalisation des évènements et 1 serveur de métadonnées, et le mode sans journalisation avec 4 serveurs de métadonnées. Certaines configurations de la MVP permettent d'obtenir de meilleures performances que les implémentations optimisées sur MPI. Ceci est expliqué par un effet de cache logiciel inter-nœuds : les données écrites dans la MVP sont stockées sur les serveurs SAT et ne viennent pas saturer directement les mémoires physiques des nœuds comme cela est le cas pour les implémentations sur OpenMPI.

En conclusion, la MVP SAT permet de déployer efficacement des applications en mémoire partagée sur des infrastructures de calcul distribuées. Les mesures effectuées consolident les bonnes performances obtenues par les systèmes MVP modernes tels que Grappa et Argo. Dans ce travail, l'aspect hétérogène de l'architecture de calcul montre de plus que MPI n'est pas adapté pour un déploiement sur des nœuds embarqués, le principal obstacle étant la mémoire physique limitée, insuffisante pour réaliser une gestion efficace des files de messages. Un second obstacle est la stratégie de

réception des messages par attente active implémentée dans la plupart des distributions MPI et qui rend le déploiement très inefficace sur les architectures embarquées : ce point est développé dans la section 3.2.2, en lien avec la gestion de l'énergie. Bien que écrite au dessus d'OpenMPI, la MVP SAT permet d'obtenir de meilleures performances en équilibrant les données sur les mémoires physiques en fonction des capacités de chaque nœud. Cette propriété est aussi observée avec la MVP Argo, elle aussi écrite au dessus de MPI.

Pour les applications distribuées et/ou massivement parallèles, écrire un code performant n'est pas la seule problématique : la préparation et la configuration du déploiement, c'est-à-dire le dimensionnement de l'application, la sélection des ressources matérielles et l'assignation des tâches sur les ressources constituent des éléments déterminants dans la réussite du calcul. Ces éléments sont bien souvent à la charge de l'utilisateur, même dans les grands centres de calcul, ce qui se traduit par des déploiements inefficaces, par exemple pour assurer un taux d'utilisation des ressources matérielles raisonnable. C'est un constat qui est partagé dès lors qu'une application est déployée sur une infrastructure parallèle et distribuée. De plus, le volet énergétique d'une exécution est devenu un critère d'évaluation au même titre que la performance calculatoire, pour les environnements opérationnels contraints en énergie comme l'embarqué mais pas seulement : les grands centres de calcul proposent aussi des rapports d'exécution incluant le coût en énergie du calcul (par exemple en kJ) comme c'est le cas pour le TGCC (Très Grand Centre de Calcul). Dans le contexte de la MVP SAT devant la complexité des architectures hétérogènes distribuées et l'organisation logique de la mémoire partagée, un outil d'aide à la décision basé sur des techniques exploratoires et d'apprentissage machine est proposé, dont l'approche est présentée dans la suite. Cet outil vise à offrir un compromis entre la performance calculatoire et la consommation d'énergie de l'application. La section suivante présente les contributions liées à la maîtrise de la dépense énergétique dans la MVP.

3.2 Maîtrise de la consommation d'énergie dans une MVP

La dépense d'énergie d'une mémoire virtuellement partagée répartie est un sujet d'étude complexe car le système interagit avec de nombreux éléments matériels susceptibles de consommer de l'énergie de manière significative s'ils sont utilisés peu efficacement. C'est le cas par exemple pour les équipements de communication si le protocole de cohérence génère de nombreux messages, ou des unités de calcul si les tâches génèrent des accès peu optimisés à la mémoire partagée. Si la gestion de la consommation d'énergie a été étudiée dans des grands intergiciels distribués, il n'existe pas à notre connaissance de contributions visant à réduire l'empreinte énergétique d'une MVP logicielle lorsque celle-ci est déployée sur une plateforme hétérogène embarquée. Dans les travaux suivants, trois axes ont été explorés afin d'apporter des solutions au niveau 1) de la configuration de la MVP, 2) de la gestion des communications et 3) de l'intégration de ressources de calcul efficaces sur le plan énergétique. Le premier axe consiste en la recherche d'un compromis entre performance calculatoire et consommation d'énergie. Cet axe présenté en section 3.2.1 propose un outil d'aide à la décision pouvant être généralisé à d'autres systèmes distribués. Le deuxième axe consiste en la réduction de la consommation d'énergie lors des phases d'attentes actives sur la réception des messages. Cet axe présenté en section 3.2.2 vise à améliorer l'adéquation entre l'intergiciel de passage de messages et les capacités matérielles embarquées possiblement limitées. Enfin, un troisième axe consiste à étendre la MVP sur des accélérateurs reconfigurables afin de favoriser des phases calculatoires efficaces du point de vue de l'énergie. Cet axe présenté en section 3.2.3 consiste en la proposition d'un modèle de programmation pour exploiter de manière transparente une architecture hybride logicielle et matérielle.

3.2.1 Aide à la décision pour la configuration d'une MVP

> Trabelsi, K., Cudennec, L., and Bennour, R. (2019). Application topology definition and tasks mapping for efficient use of heterogeneous resources. In Schwardmann, U., Boehme, C., Heras, D. B., Cardellini, V., Jeannot, E., Salis, A., Schifanella, C., Manumachu, R. R., Schwamborn, D., Ricci, L., Sangyoon, O., Gruber, T., Antonelli, L., and Scott, S. L., editors, Euro-Par 2019 : Parallel Processing Workshops - Euro-Par 2019 International Workshops, Göttingen, Germany, August 26-30, 2019, Revised Selected Papers, volume 11997 of Lecture Notes in Computer Science, pages 258–269. Springer.
>
> [Trabelsi et al., 2019] CEA LIST.
>
> Contributions | ★ initiative ■ idées □ implémentation ■ expérimentations ■ rédaction □ présentation

Déployer une application parallèle sur une machine de calcul distribuée est un problème complexe, bien souvent sous-estimé et réalisé à la main par l'utilisateur. Cette thématique de recherche n'est pas nouvelle et a été théorisée et étudiée dès l'apparition des machines parallèles, par exemple à travers la méthode PCAM (*Partition Communicate Agglomerate Map*) proposée par Ian Foster en 1995 [Foster, 1995]. Cette méthode décompose le déploiement en quatre phases : 1) la décomposition du calcul permet d'exhiber le parallélisme de tâches ou de données, sans considérations dimensionnantes de la plateforme matérielle ciblée, 2) les communications entre tâches sont établies pour construire un graphe, 3) les tâches sont regroupées ou fusionnées par affinité et enfin 4) les groupes de tâches sont projetés

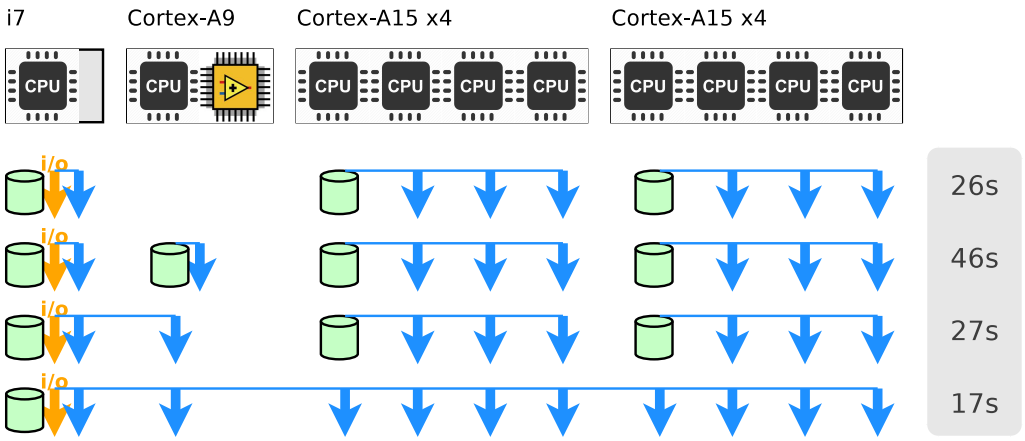


FIGURE 3.7 – Temps de calcul mesurés pour une application de traitement d'image distribuée sur le serveur Christmann RECS|Box 3 présenté en figure 3.1a pour 4 configurations du déploiement de la MVP SAT. Les serveurs de métadonnées SAT sont représentés par des cylindres verts et les processus de traitement par des flèches bleues. Les barres bleues horizontales représentent l'association entre une tâche de traitement et un serveur de métadonnées SAT. Une analyse plus complète de ces résultats est donnée dans [Cudennec, 2017].

sur les ressources matérielles. J'ai eu l'occasion d'étudier ce sujet dans le contexte opérationnel de la grille de calcul Grid'5000 pour le déploiement de la MVP JuxMem en multi-sites [Cudennec, 2009], puis dans le cadre de la chaîne de compilation Σ-C présentée en section 2.1.1 et dans laquelle une étape d'optimisation est dédiée au dimensionnement de l'application [Cudennec and Sirdey, 2012] et au placement des tâches par recuit simulé sur les PE du many-cœur [Galea and Sirdey, 2012]. Dans le contexte de la MVP SAT, différentes configurations de déploiement d'une même application sur une architecture hétérogène peut donner lieu à des performances variables comme cela est illustré par la figure 3.7. Ces résultats montrent l'importance de disposer d'outils et de l'expertise nécessaire à la configuration du déploiement pour utiliser efficacement les ressources matérielles.

De nombreux outils permettent d'automatiser le déploiement, ce qui a été facilité par les méthodes de virtualisation, de conteneurisation ainsi que des intergiciels de gestion de ressources tels que OpenStack. Si les étapes de configuration des nœuds de calcul, de recopie des données d'entrée et de lancement de l'exécution du code utilisateur sont prises en charge, la phase de pré-planification du déploiement est plus difficilement assistée et repose sur l'expertise de l'utilisateur. La pré-planification consiste à dimensionner l'application, c'est-à-dire pour simplifier, définir le nombre de processus et décider du placement individuel sur les ressources matérielles. Un objectif classique de cette phase est de minimiser le temps de calcul. Mais les contraintes opérationnelles transforment rapidement ce problème en une optimisation multicritères, dans laquelle la consommation d'énergie ou le nombre de ressources allouées doivent aussi être minimisés. Comme dans tout bon problème d'optimisation, ces critères sont bien souvent antinomiques et un compromis doit être trouvé.

Dans le contexte d'utilisation de la MVP SAT, la complexité de la pré-planification provient de deux aspects : 1) l'aspect multirôles joué par l'application dans laquelle des serveurs de métadonnées SAT et divers rôles définis par l'utilisateur cohabitent et 2) l'aspect hétérogène des plateformes ciblées dans lesquelles certaines contraintes d'association entre nœud de calcul et type de rôle dirigent la phase de placement de tâches (par exemple la présence d'un CPU basse consommation, d'un GPU, d'une quantité suffisante de mémoire physique…). Une solution de déploiement d'une application sur SAT consiste donc à instancier l'application (définir le nombre d'instances de chaque rôle à déployer), construire la topologie (comment les instances des rôles utilisateur sont connectées aux instances du rôle de serveur de métadonnées SAT) et comment projeter chaque instance sur les ressources matérielles en respectant des contraintes de placement. Cette solution de déploiement est alors évaluée selon divers critères, principalement le temps de calcul et la consommation d'énergie. L'objectif du travail mené dans [Trabelsi et al., 2019] est de fournir un outil d'aide à la décision pour semi-automatiser la configuration la plus adaptée pour un déploiement.

Dans la littérature, de nombreux travaux traitent du problème du placement des tâches sur des ressources hétérogènes en présence de contraintes de performance et d'énergie. Cette thématique de recherche s'est développée depuis une quinzaine d'années avec l'arrivée d'architectures hybrides composées d'un CPU et d'un GPU. Par exemple, un algorithme populationnel bi-objectif est utilisé dans [Friese et al., 2013] pour sélectionner les solutions dominantes présentées sous forme d'un front de Pareto et ainsi aider les administrateurs système à déployer des applications. D'autres contributions proposent des approches itératives [Legrand et al., 2004] pour construire une solution ou des techniques de partitionnement des tâches pour rechercher des solutions à plusieurs niveaux de hiérarchie [Uçar et al., 2006]. D'autres approches concernent le placement et l'ordonnancement des tâches lorsque l'application peut être représentée sous la forme d'un DAG (*Directed Acyclic Graph*), par exemple dans [Aba et al., 2020].

Cependant, ces travaux portent sur l'optimisation d'une unique phase du modèle PCAM, principalement la projection sur les ressources physiques. Les autres phases doivent être traitées avec d'autres méthodes et outils. Cette approche simplifie la complexité du problème mais peut induire des solutions globales moins optimisées en étant contrainte par les autres phases, ou en contraignant les autres phases. Dans les travaux menés sur la MVP SAT, une approche globale est proposée, prenant en compte toutes les phases de la méthodologie PCAM, du dimensionnement de l'application à la phase de projection sur les ressources matérielles.

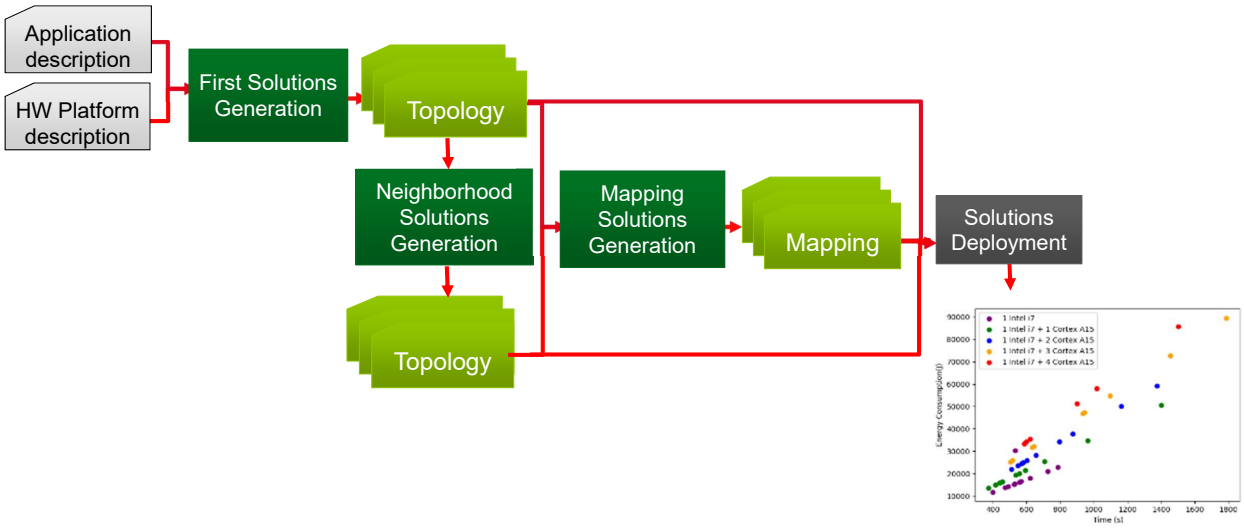
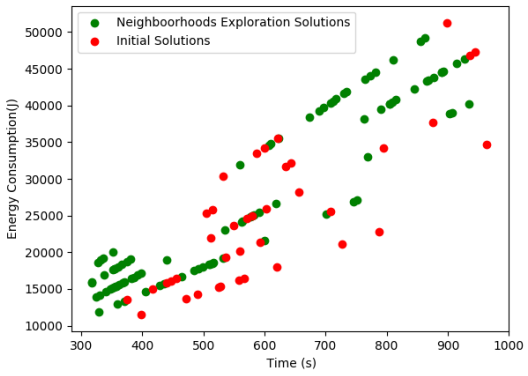


FIGURE 3.8 – Processus d'optimisation combiné des phases de dimensionnement de l'application et de sa projection sur l'architecture de calcul. Le front de Pareto obtenu est un outil d'aide à la décision afin de déployer la configuration la plus adaptée au contexte opérationnel.

La recherche de solutions par voisinage est une méthode d'exploration approchée offrant un compromis entre le temps de calcul (le coût de la recherche) et la qualité des solutions. Cette méthode est donc particulièrement bien adaptée aux problèmes complexes pour lesquels l'espace de recherche est considéré grand. Le principe est de générer un ensemble de solutions initiales, de les évaluer, puis de générer itérativement de nouvelles solutions basées sur les précédentes en appliquant des altérations. Dans ce travail, une solution est constituée 1) d'une topologie logique de la MVP telle que représentée en figure 3.2b et qui correspond aux deux premières phases de la méthode PCAM et 2) d'une projection de cette topologie sur les ressources matérielles en autorisant la colocation de certaines tâches avec leur serveur de métadonnées SAT sur le même nœud de calcul pour favoriser les communications, ce qui correspond aux deux dernières phases de la méthode PCAM. Cette approche est illustrée dans la figure 3.8 par une chaîne d'optimisation prenant en entrée une description de l'application et des ressources matérielles pour générer un ensemble de solutions à évaluer sur la plateforme cible.

Cette approche a été évaluée sur le micro-serveur Christmann RECS|Box et les résultats sont présentés en figure 3.9. La figure de gauche 3.9a représente le positionnement des solutions évaluées sur la plateforme en fonction du temps de calcul et de la consommation d'énergie, pour les solutions initiales en rouge et les solutions voisines en vert. Cette représentation montre que de meilleures solutions ont été trouvées suite à l'exploration du voisinage, principalement sur le critère du temps de calcul et sur le compromis performance/énergie. Les détails des solutions constituant le front de Pareto sont donnés en figure 3.9b. Il est à noter que le nombre de tâches de traitement, la topologie de la MVP ainsi que la projection sur les ressources physiques peuvent différer entre deux solutions. La forme du Pareto obtenu après exploration du voisinage révèle de plus une solution offrant un excellent compromis entre le temps de calcul (328 s) et la consommation d'énergie (11,8 kJ).

En conclusion, l'outil d'exploration pour le dimensionnement des applications écrites sur la MVP SAT ainsi que les projections possibles sur les ressources matérielles permet de rechercher des solutions contre-intuitives : dans les expérimentations, aucune configuration manuelle jusqu'alors utilisée pour déployer l'application de traitement vidéo sur la plateforme n'a été retenue dans le front de Pareto. Si cette approche est satisfaisante sur le plan de la qualité des solutions obtenues, elle souffre cependant d'un mode opératoire difficile à mettre en place : Dans cet exemple, l'évaluation d'une solution prend entre 5 et 15 minutes. Évaluer toutes les solutions entraîne donc un temps de calcul important, certainement impossible à réaliser tel quel sur un super-calculateur ou sur tout autre plateforme multi-utilisateurs. Plusieurs approches sont cependant envisageables pour réduire le temps de calcul ou améliorer la stratégie exploratoire.



(a) Positionnement des solutions en fonction du temps de calcul et de l'énergie dépensée. Le front de Pareto obtenu met en valeur les solutions nouvellement trouvées par exploration du voisinage. Le phénomène de traînées provient de topologies proches structurellement, par exemple suite à l'ajout d'une tâche de traitement ou d'un serveur de métadonnées de la MVP SAT.

Solutions initiales						
i7	A15	SAT	Temps (s)	FPS	kJ	FPS/kJ
1		1	398	4.3	11.5	0.38
1	1	2	375	4.6	13.5	0.34
1		1	377	4.6	13.5	0.34
Exploration du voisinage						
i7	A15	SAT	Temps (s)	FPS	kJ	FPS/kJ
1	3	4	316	5.4	15.8	0.35
1	2	3	324	5.3	13.9	0.38
1	1	2	328	5.3	11.8	0.45

(b) Solutions appartenant au front de Pareto. Pour chaque solution sont indiqués dans l'ordre des colonnes : le nombre de processeurs Intel Core i7 et Arm Cortex A15 utilisés, le nombre de serveurs de métadonnées SAT, le temps de calcul, le nombre moyen d'images traitées par seconde, l'énergie dépensée pour le calcul ainsi que le ratio entre le nombre d'images par seconde et l'énergie.

FIGURE 3.9 – Solutions initiales et solutions obtenues par exploration du voisinage appartenant au front de Pareto - solutions non dominées sur le temps de calcul et la consommation d'énergie - pour l'application de traitement vidéo *videostream* déployée sur le serveur hétérogène Christmann RECS|Box 3 présenté en figure 3.1a.

Une première méthode est d'appliquer les évaluations de solution sur un sous-ensemble de l'application, par exemple un noyau de calcul considéré comme représentatif du comportement applicatif. Ceci est particulièrement adapté pour les applications exhibant un régime établi, c'est-à-dire un motif comportemental identifiable en terme de calculs et de communications. Cette première méthode doit pouvoir faire chuter le temps de calcul de manière significative. Une deuxième méthode est d'utiliser des outils d'optimisation d'hyperparamètres tels que Optuna [Akiba et al., 2019] qui permettent d'explorer de grands espaces en termes de nombre de variables et de domaines de recherche. Ces outils utilisent des stratégies adaptées, par exemple l'inférence bayésienne pour orienter les recherches. Cette deuxième méthode doit pouvoir limiter le nombre d'évaluations nécessaires pour converger vers les meilleures solutions. Une troisième méthode consiste à construire un modèle de prédiction des performances. Cette approche doit permettre d'obtenir une réponse instantanée lors de l'évaluation d'une solution de déploiement, et ainsi permettre 1) la construction du front de Pareto hors-ligne, c'est-à-dire sans évaluation sur la machine cible et 2) une exploration efficace, en utilisant une machine spécialisée pour les problèmes de RO (par exemple une machine NUMA plutôt que la machine hétérogène distribuée ciblée). La principale complexité de cette troisième approche consiste à construire un modèle prédictif précis : plusieurs possibilités sont envisagées pour la suite de ces travaux comme par exemple 1) l'utilisation de séries temporelles appliquées à des séquences de déploiements issues d'une exploration du voisinage (et donc ayant subi de petites altérations incrémentales) et 2) un apprentissage machine sur des bases de données contenant des rapports de déploiements déjà effectués. Dans les deux cas, le paramétrage du modèle est rendu complexe par le faible nombre de données disponibles suite à des déploiements. Cette problématique rejoint une thématique actuelle de recherche sur l'IA frugale, c'est-à-dire comment entraîner des modèles avec peu de données.

Enfin, une part de la complexité de l'exploration des configurations de déploiement provient de l'aspect multicritères de l'optimisation. Jusqu'à récemment ce type de problème était uniquement axé sur la performance calculatoire : de nombreuses publications démontrent l'intérêt de leur contribution en comparant exclusivement des résultats de *benchmarks* dans lesquels l'aspect énergétique n'est pas traité. Le classement GREEN500 des super-calculateurs performants sur le plan énergétique a débuté il y a moins de 10 ans, en juin 2013, ce qui est finalement récent (cf. la figure 2.5). Si les méthodes de recherche opérationnelle permettent de trouver des compromis satisfaisants entre la performance calculatoire et la dépense énergétique, il n'en reste pas moins que ces solutions sont tributaires des choix effectués lors de l'implémentation du système MVP. Un système non pensé pour limiter la dépense énergétique reste un système énergivore quelques soient les efforts menés pour le configurer. Les contributions suivantes portent donc sur la prise en compte de la dépense d'énergie dans la MVP au niveau calculatoire et au niveau des communications.

3.2.2 Économie d'énergie dans les communications MPI, application à la MVP

> Cudennec, L. (2020). Adaptive message passing polling for energy efficiency : Application to software-distributed shared memory over heterogeneous computing resources. Concurr. Comput. Pract. Exp., 32(24).
> [Cudennec, 2020] CEA LIST, DGA MI.
> Contributions | ■ initiative ■ idées ■ implémentation ■ expérimentations ■ rédaction

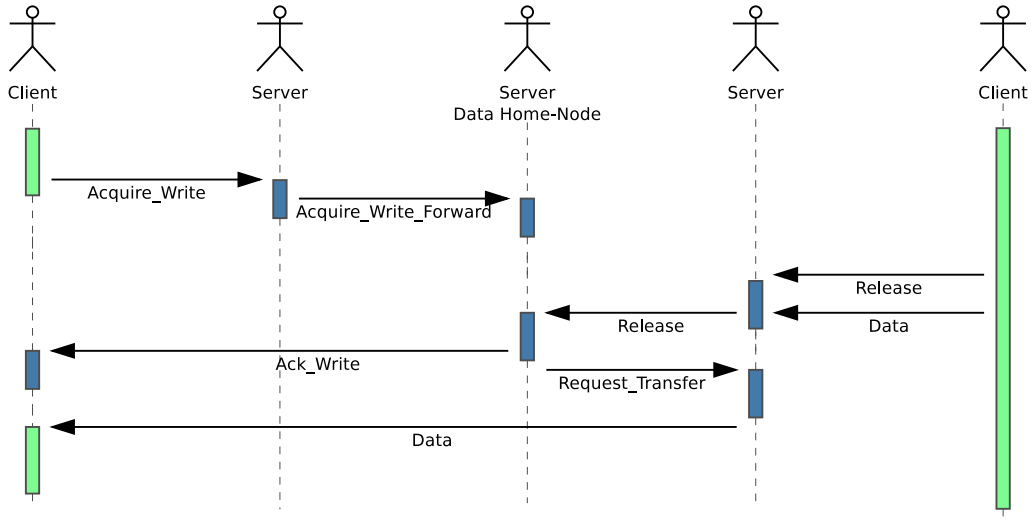
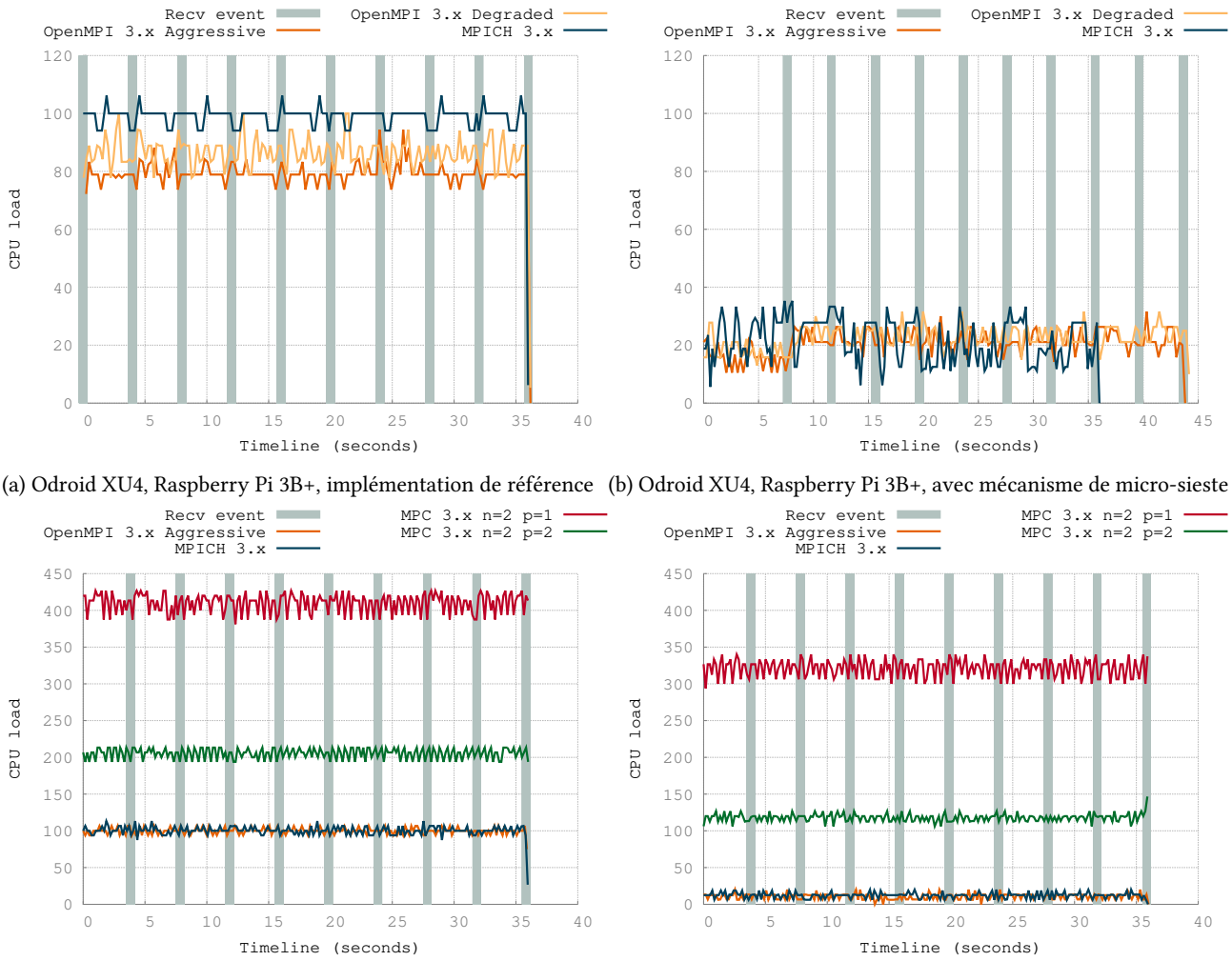


FIGURE 3.10 – Exemple de diagramme de séquence illustrant l'enchaînement des communications entre les acteurs de la MVP SAT lors d'une requête en écriture sur une donnée en cours d'accès par un processus distant. les boîtes vertes correspondent à l'exécution du code utilisateur et les boîtes bleues à l'exécution du code MVP. Les pointillés correspondent aux lignes de vie. Lorsque les pointillés sont visibles, le processus est considéré comme inactif, c'est-à-dire en attente d'un évènement. Ce motif de communication présente un pire cas dans lequel le serveur responsable de la donnée (*home-node*) est différent des serveurs sur lesquels sont connectés les clients utilisateur.

Une MVP peut être définie comme un ensemble d'automates répartis. Chaque participant, qu'il soit client ou serveur, est doté d'un état interne qui peut être modifié par des transitions. Ces transitions s'effectuent lors d'évènements tels que l'appel à une primitive de l'API utilisateur sur les serveurs ou la réception d'un message de contrôle sur les clients. Un certain nombre d'acteurs dans le système peuvent passer un temps significatif à attendre un évènement, comme cela est illustré en figure 3.10. Dans cet exemple, le client de gauche à l'initiative de la requête en écriture doit attendre le message d'acquittement et le transfert des données pour continuer le traitement. De la même manière, les serveurs de la MVP passent une grande partie du temps à attendre des messages de contrôle. La complexité calculatoire est très faible dans une MVP et le surcoût d'utilisation pour l'application est presque intégralement la conséquence du respect de dépendances causales entre évènements et de l'algorithmique répartie en général. Pendant ces phases d'attente, les cœurs de calcul exécutant les différents rôles peuvent changer de contexte et exécuter d'autres processus appartenant à d'autres applications. En pratique, lors d'un déploiement sur une infrastructure de production ou un super-calculateur, la colocalisation de processus sur un même cœur (aussi dénommé *oversubscription* dans la terminologie MPI) est rarement mis en place pour des raisons de performance, d'isolation (éviter les effets de bord, notamment la pollution des caches), et de problématiques de partage des ressources en général. Dès lors, ces phases d'inactivité peuvent mener à une sous-utilisation du cœur de calcul, ce qui doit être compensé par des économies d'énergie.

Dans le contexte de la MVP SAT, la couche de gestion des communications se conforme au standard MPI comme cela est motivé en section 3.1. La plupart des implémentations de MPI sont conçues pour le domaine du HPC dans lequel les applications sont classiquement caractérisées par un comportement régulier qui ne génère pas de temps d'attente significatifs comparé aux phases de calcul. Dès lors les primitives d'attente d'un message sont implémentées dans une boucle active afin de garantir un temps de réaction court entre la réception du message sur l'interface réseau et sa remise au niveau utilisateur. Malheureusement cette stratégie entraîne un fort taux d'utilisation du CPU et une consommation d'énergie importante. C'est ce qui est constaté avec les résultats expérimentaux présentés en figure 3.11. Dans ces expérimentations, une application composée d'un processus émetteur et d'un processus récepteur est écrite en MPI. Un message est envoyé toutes les 4 secondes, le processus récepteur se retrouve donc en attente de message la plupart du temps. Les figures de gauche présentent l'implémentation de référence sur 2 architectures matérielles et 3 implémentations de MPI (OpenMPI [Gabriel et al., 2004], MPICH [Thakur and Gropp, 2007] et MPC [Pérache et al., 2008]). Dans toutes ces configurations, la charge processeur correspond à un processus en attente active, proche de 100% pour OpenMPI et MPICH (ce qui est équivalent à un cœur en pleine utilisation), et jusqu'à 400% pour MPC dont



(a) Odroid XU4, Raspberry Pi 3B+, implémentation de référence (b) Odroid XU4, Raspberry Pi 3B+, avec mécanisme de micro-sieste

(c) Intel Core i7 6800K (6 cœurs), implémentation de référence (d) Intel Core i7 6800K (6 cœurs), avec mécanisme de micro-sieste

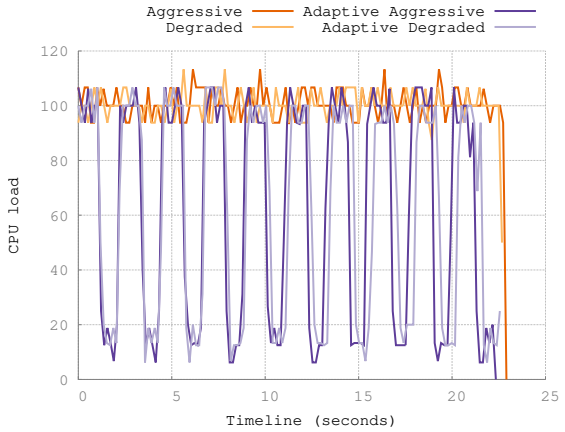
FIGURE 3.11 – Journalisation de la charge processeur dédiée au processus MPI en attente de messages lors de l'exécution d'une application d'envoi et de réception de messages. Un message est émis toutes les 4 secondes et sa réception est indiquée par les barres verticales grises. L'application est déployée sur 2 configurations matérielles et 3 implémentations de MPI. Les figures 3.11a et 3.11c sont obtenues avec les implémentations par défaut. Les figures 3.11b et 3.11d sont obtenues en activant le mécanisme de micro-sieste proposé dans [Cudennec, 2020] et dont le code source est mis à disposition dans le projet EEProbe (*Energy Efficient Probe for MPI*). https://github.com/lcudenne/eeprobe

l'implémentation est résolument parallèle. Afin de réduire ce taux d'utilisation processeur lors d'une phase de réception de message, un mécanisme de micro-sieste a été proposé avec pour objectif d'offrir un compromis entre réactivité et consommation d'énergie.

Dans la littérature, les implémentations MPI ont -de manière assez surprenante- peu fait l'objet d'études pour rationaliser la consommation d'énergie. Quelques implémentations font cependant mention de tels travaux : MPC a été développé pour tirer parti des nœuds de calcul NUMA et permet à un ordonnanceur interne de donner la main à un autre fil d'exécution MPC lors de l'attente d'un message. Le taux d'utilisation du processeur obtenu en figure 3.11c montre cependant que ce mécanisme est pensé pour établir un partage efficace des cœurs entre les fils d'exécution MPC à des fins de performance calculatoire et non d'économie d'énergie. Un mécanisme similaire (*oversubscription*) est implémenté dans OpenMPI et MPICH dans le cas de la colocalisation de processus sur un même nœud. Là encore, l'ordonnanceur interne cherche à donner la main à un autre processus MPI et ne permet pas de restituer la ressource en cas d'attente d'un message. La contribution la plus proche de cette problématique est MVAPICH2-EA [Venkatesh et al., 2015] qui se base sur un modèle de prédiction pour décider de la transmission des données et de la configuration en ligne des commutateurs réseaux. L'approche repose sur des technologies matérielles particulières fournies par Infiniband et Intel Sandy Bridge, ce qui est peu adapté au contexte du calcul embarqué. Cependant, les auteurs montrent une diminution de la consommation de plus de 40% pour seulement 5% de perte de performance, ce qui constitue une avancée significative pour le contexte HPC.

D'autres initiatives visent à apporter des outils pour maîtriser la consommation d'énergie d'une application MPI, c'est par exemple le cas avec des orchestrateurs de déploiement pouvant décider de fusionner des processus (GEOPM [Eastep et al., 2017]) ou encore d'adapter la fréquence des processeurs pendant l'exécution (Green Queue [Tiwari et al., 2012], Transparent Scaling [Lim et al., 2006]). Ces approches rejoignent les travaux engagés sur la recherche d'une configuration de déploiement répondant à des objectifs de diminution de la consommation d'énergie comme présenté en section 3.2.1. Enfin, la problématique de la dépense énergétique dans MPI est aussi traitée à l'aide de simulateurs et de modèles, comme par exemple avec HAEC-SIM [Bielert et al., 2015] et [Heinrich et al., 2017]. Dans ce dernier, les auteurs soulèvent la difficulté de caractériser la consommation énergétique de la fonction MPI `Iprobe` justement dans le cas d'une attente active de messages, ce qui rejoint la problématique traitée dans cette section.

FIGURE 3.12 – Implémentation et évaluation du mécanisme de micro-sieste implémenté entre l'API MPI et la MVP SAT.



(a) Journalisation de la charge processeur dédiée au processus MPI exécutant un rôle serveur de la MVP SAT pour deux modes de fonctionnement OpenMPI : *Aggressive* correspond au mode normal (boucle d'attente active) et *degraded* correspond à un mode spécifique (*mpi_yield_when_idle*) pour le partage de ressources entre processus MPI. Les courbes violettes (*Adaptive*) correspondent à l'utilisation du mécanisme de micro-sieste.

Listing 5 – Pseudo-code pour la fonction de réception avec mécanisme de micro-sieste.

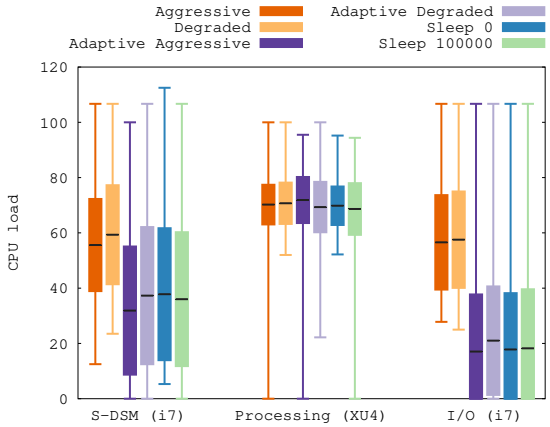
```
1  void EERecv() {
2    char * buffer = NULL;
3    size_t size = 0;
4    int flag = 0;
5    unsigned long sleep_time = 0;
6    while (flag == 0) {
7      Iprobe(&flag, &size);
8      if (flag == 0) {
9        clock_nanosleep(sleep_time);
10       sleep_time += sleep_time_step;
11     }
12   }
13   buffer = malloc(size);
14   Recv(buffer);
15 }
```

Un mécanisme de micro-sieste pour relâcher l'attente active. Économiser l'énergie dans les communications en utilisant une diminution de l'activité matérielle est un sujet qui a été étudié dans les grandes infrastructures de communication, notamment à travers le standard EEE (*Energy Efficient Ethernet*) qui permet d'éteindre sélectivement des ressources lorsque l'activité le permet [Cenedese et al., 2017]. Plus récemment, cette thématique trouve des applications dans les systèmes de calcul embarqués, l'informatique en périphérie de réseau et les réseaux d'objets communicants à très faible puissance de consommation [Djidi et al., 2021]. Appliquer certaines approches proposées dans ces thématiques de recherche doit permettre d'adapter MPI à de nouveaux contextes applicatifs, comme les machines hétérogènes embarquées.

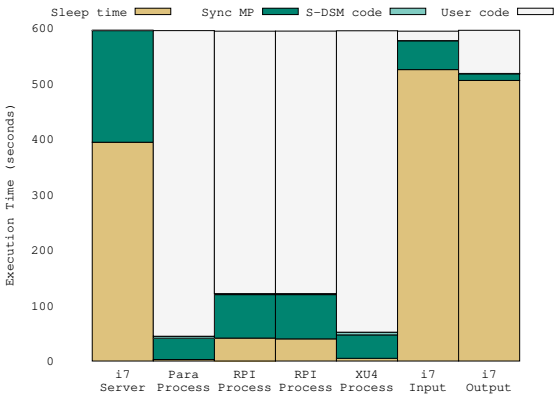
Dans le contexte de la réception de messages MPI, le mécanisme de micro-sieste consiste à remplacer la boucle d'attente active implémentée notamment dans la fonction `Recv` par une boucle comprenant deux actions : la vérification de la présence de nouveaux messages (le *polling*) et l'endormissement pour un temps arbitraire (ce qui permet à l'ordonnanceur système de donner la main à un autre processus ou de diminuer la charge processeur). En pratique, un code de réception de message MPI est souvent composé de l'enchaînement d'un appel à la fonction synchrone `Probe` qui retourne lorsqu'un message est disponible, puis de la fonction `Recv` qui permet effectivement de délivrer le message. Ce même code est proposé en incluant le mécanisme de micro-sieste dans le listing 5. Dans cette implémentation simplifiée, le temps d'endormissement est augmenté à chaque itération jusqu'à la réception d'un message. Cette stratégie permet de conserver un comportement proche de l'attente active sur les premières itérations lors de l'appel à la fonction `EERecv`, et donc de ne pas pénaliser les phases de communication denses, et dans un second temps de relâcher progressivement la pression sur le processeur. L'application de cette stratégie est évaluée sur l'application MPI d'envoi et de réception de messages et les résultats sont présentés sur les figures 3.11b et 3.11d. La charge processeur est divisée par 5 sur les cartes embarquées et par 10 sur le processeur Intel pour les implémentations OpenMPI et MPICH. Elle est aussi réduite dans une moindre mesure pour MPC, ce qui peut être expliqué par une implémentation différente en terme de gestion des fils d'exécution.

Cette réduction de la charge processeur est aussi mesurable dans le contexte de la MVP SAT. La figure 3.12a présente la charge processeur du processus MPI supportant le rôle de serveur de la MVP pendant l'exécution d'une application

itérant sur des séquences de 1000 écritures en mémoire partagée espacées d'une seconde, pour un total de 180000 messages. Entre ces séquences d'écriture la charge processeur descend à 10% lorsque le mécanisme de micro-sieste est activé, contrairement à une utilisation de MPI plus classique qui reste à 100%. Ces améliorations sont aussi perceptibles lors de l'exécution de l'application de traitement vidéo, comme illustré en figure 3.13. Les résultats montrent une diminution significative de la charge processeur (figure 3.13a) pour les processus MPI exécutant les rôles de serveur et de gestionnaire d'entrées-sorties dans la MVP. Ces résultats sont expliqués par la décomposition du temps d'exécution par processus présenté en figure 3.13b qui révèle la part importante passée dans le mécanisme de micro-sieste. Ce résultat est d'autant plus intéressant que le temps de calcul reste inchangé dans toutes les configurations et que l'utilisation du mécanisme de micro-sieste ne nécessite aucune modification du code source d'OpenMPI, de la MVP SAT ou du code applicatif.



(a) Variance de la charge processeur pour différents rôles de l'application *videostream* et pour différentes configurations du mécanisme de micro-siestes implémenté sur OpenMPI. *S-DSM* correspond à un rôle serveur de la MVP déployé sur un Intel Core i7, *Processing* à une tâche de traitement déployée sur une carte Odroid XU4 et *I/O* à un processus de gestion des entrées-sorties déployé sur le même Core i7.

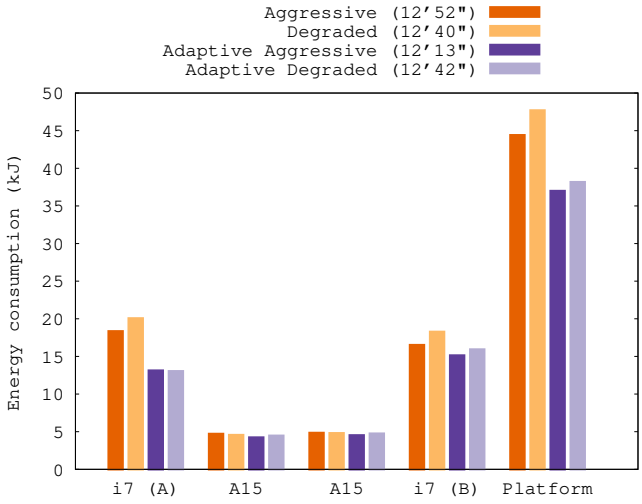


(b) Décomposition du temps d'exécution des processus de l'application *videostream*. *Sleep time* correspond au temps d'endormissement dans le mécanisme de micro-sieste, *Sync MP* au temps passé à attendre un message en attente active, *S-DSM code* au temps passé à exécuter le code de la MVP et *User code* le temps passé à exécuter le code utilisateur.

FIGURE 3.13 – Analyse de la charge processeur et décomposition du temps passé par les processus de l'application *videostream* présentée en section 3.1.2 lors d'une déploiement sur la machine hétérogène présentée en figure 3.1b. *Adaptive* correspond au mécanisme de micro-sieste avec un temps d'endormissement variable. *Sleep* 0 et 100000 correspondent au mécanisme de micro-sieste avec un temps d'endormissement fixe exprimé en µs.

Réduction de la consommation d'énergie dans une MVP. Le mécanisme de micro-sieste permet de diminuer la charge processeur dans le cas où un processus MPI effectue une attente de message. Cette diminution de la charge processeur doit permettre de faire progresser d'autres processus ou de laisser le système d'exploitation décider si un changement de mode de fonctionnement du processeur est envisageable pour effectuer des économies d'énergie (par exemple la baisse de la fréquence d'horloge ou le changement d'état d'un cœur du processeur). Afin de mesurer finement la consommation d'énergie, des expérimentations ont été menées sur le micro-serveur RECS|Box : des sondes de consommation sont embarquées dans chacun des nœuds et peuvent être consultées au travers d'une interface REST (*Representational State Transfer*). La journalisation des mesures permet ensuite de calculer l'énergie dépensée par nœud pendant le calcul. Les résultats obtenus sont présentés en figure 3.14. Une première observation est que le mécanisme de micro-sieste permet de diminuer la consommation énergétique de manière significative : jusqu'à 20% d'économies à l'échelle de la plateforme. Une deuxième observation est que le mécanisme permet de surcroît de diminuer le temps de calcul de 5%. Ce résultat est plutôt contre-intuitif : classiquement le temps de calcul et la consommation d'énergie sont des critères d'optimisation antagonistes et l'amélioration de l'un des critères se fait en défaveur du second, suivant le principe du front de Pareto, comme cela a déjà été observé pour l'aide à la décision sur le déploiement de la MVP en section 3.2.1. Il apparaît que dans le contexte applicatif de SAT, le mécanisme de micro-sieste permette de jouer sur les deux tableaux, ce qui en fait un dispositif incontournable pour le déploiement de la MVP sur des architectures embarquées.

En conclusion, l'étude de la consommation d'énergie dans MPI est un axe de recherche qui s'est imposé lors d'expérimentations menées avec la MVP sur les machines hétérogènes. Lors de ces déploiements, il est rapidement apparu que la charge CPU et la consommation d'énergie des nœuds exécutant uniquement les rôles de serveurs de données et métadonnées de SAT étaient anormalement élevées dans les phases calculatoires. Lors de ces phases, aucun accès en



(a) Consommation énergétique en kJ mesurée sur différents nœuds de calcul avec sans activation du mécanisme de micro-sieste. Seuls deux nœuds ARM Cortex A15 sont représentés sur les quatre utilisés pour des raisons de lisibilité.

	TIME	FPS	W	kJ	FPS/W
	12'52	2.23	60	44.39	0.037
	12'40	2.26	66	47.68	0.034
	12'13	2.35	52	36.98	0.045
	12'42	2.26	52	38.16	0.043

(b) Temps de calcul (min), images traitées par seconde (FPS), consommation instantanée moyenne (W), consommation totale (kJ) et performance énergétique (FPS/W). La solution *Adaptive Aggressive* correspond à l'activation du mécanisme de micro-siestes au dessus d'OpenMPI configuré en mode standard (sans prise en compte de la colocalisation de processus sur un même nœud). Cette solution se démarque en offrant le meilleur temps et la meilleure efficacité énergétique.

FIGURE 3.14 – Performance calculatoire et performance énergétique de l'application de traitement vidéo déployée sur le serveur Christmann RECS|Box avec et sans l'activation du mécanisme de micro-sieste (dénommé *Adaptive*).

mémoire partagée n'est effectué et aucune activité n'est donc déclenchée dans la MVP pouvant justifier une telle utilisation des ressources. Les différentes implémentations de MPI testées ainsi que les options de compilation et d'exécution étudiées n'ont pas permis de résoudre le problème de l'attente active. Le mécanisme de micro-sieste implémenté comme une couche logicielle interfacée entre le code applicatif et les primitives MPI est une solution répondant à plusieurs objectifs : 1) une diminution significative de la charge processeur dans les phases d'attente de messages considérées *longues*, c'est-à-dire à partir de plusieurs dizaines de millisecondes et 2) une mise-en-œuvre transparente pour l'utilisateur et qui ne nécessite pas de réécriture du code applicatif ni du code MPI. Cette approche permet d'obtenir 90% de diminution de charge CPU sur les phases d'attente de messages, 20% d'économies d'énergie et 5% d'amélioration du temps de calcul pour une application de traitement vidéo déployée sur un micro-serveur hétérogène.

Bien que d'une grande simplicité, le mécanisme de micro-sieste pose un certain nombre de problèmes reconnus pour être difficiles à résoudre, notamment celui du paramétrage des temps d'endormissement (temps minimum, temps maximum et incrément du temps d'endormissement). Le choix de ces paramètres conditionne les bonnes propriétés du mécanisme sur la capacité à s'endormir suffisamment longtemps pour observer une économie d'énergie tout en restant réactif dans les phases de communication plus denses. Dans ces travaux les paramètres ont été fixés de manière arbitraire et un axe d'amélioration consiste donc à proposer une méthode pour les déterminer à l'exécution, pour chaque processus MPI et pour des phases différentes du calcul. Une piste est de s'inspirer des techniques d'apprentissage par renforcement -le RL (*Reinforcement Learning*)- largement utilisées dans les contextes opérationnels et notamment mis en lumière avec l'apprentissage machine sur les jeux vidéos Atari [Mnih et al., 2013]. Dans un système RL, un environnement réel ou simulé est intégré dans la boucle d'apprentissage. Celui-ci prend en entrée des actions ou des commandes de la part d'un agent et retourne un score sanctionnant cette décision. Dans le contexte du mécanisme de micro-sieste, un tel système peut être déployé en considérant que les agents sont les processus MPI, les actions consistent à modifier les paramètres d'endormissement du mécanisme et que les critères d'évaluation sont dépendants du contexte opérationnel comme la charge processeur, la consommation d'énergie, la performance calculatoire ou la réactivité à la réception des messages.

Un deuxième axe d'étude consiste à évaluer le mécanisme de micro-sieste dans des contextes applicatifs issus du calcul haute-performance. Bien que principalement adapté pour des applications présentant des phases de synchronisation et des temps d'attente importants, le mécanisme doit pouvoir modifier le comportement de noyaux de calcul plus denses en permettant un meilleur partage des ressources matérielles lors d'un déploiement sur une machine à mémoire partagée multi-cœur ou NUMA. Ainsi, des travaux sont actuellement menés pour adapter automatiquement des *benchmarks* HPC tels que le *NAS Parallel Benchmark* [Bailey et al., 1991] en reposant sur la couche d'abstraction de micro-sieste `EEProbe`. Au delà des *benchmarks*, cette approche vise aussi les intergiciels d'apprentissage machine, notamment pour le RL en ligne dans un contexte embarqué et contraint par l'énergie.

3.2.3 Intégration d'accélérateurs matériels reprogrammables (FPGA) dans une MVP

> Lenormand, E., Goubier, T., Cudennec, L., and Charles, H.-P. (2021). Data management model to program irregular compute kernels on fpga : application to heterogeneous distributed system. In Euro-Par 2021 : Parallel Processing Workshops - Euro-Par 2021 International Workshops, Lisbon, Portugal, August 30 - September 3, 2021, Revised Selected Papers, Lecture Notes in Computer Science. Springer.
>
> [Lenormand et al., 2021] CEA LIST, DGA MI.
>
> Contributions | ★ initiative ■ idées ☐ implémentation ☐ expérimentations ☐ rédaction ☐ présentation

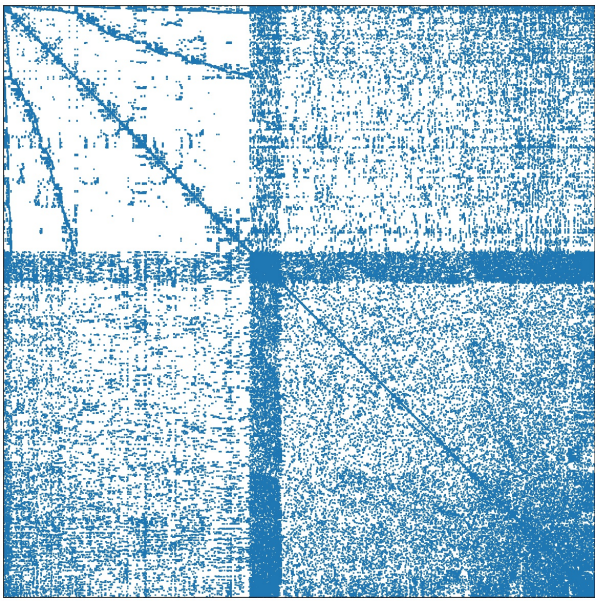
> Lenormand, E., Goubier, T., Cudennec, L., and Charles, H.-P. (2020). A combined fast/cycle accurate simulation tool for reconfigurable accelerator evaluation : application to distributed data management. In 2020 International Symposium on Rapid System Prototyping, RSP 2020, Virtual Conference, September 24-25, 2020. IEEE.
>
> [Lenormand et al., 2020] CEA LIST, DGA MI.
>
> Contributions | ★ initiative ■ idées ☐ implémentation ☐ expérimentations ■ rédaction ☐ présentation

Les FPGA sont des processeurs reprogrammables dont la logique interne peut être configurée *simplement*. Si à l'origine dans les années 1990 cette reconfiguration était basée sur une description bas-niveau de la circuiterie, elle est aujourd'hui prise en charge par des outils et des langages de conception permettant d'abstraire l'architecture de la puce tels que le HDL (*Hardware Description Language*). Sur le principe, les FPGA sont constitués d'une matrice de portes logiques programmables, d'un ensemble d'accélérateurs spécifiques de type DSP (*Digital Signal Processor*), de mémoire embarquée ainsi que de connecteurs d'entrées-sorties pour s'interfacer avec des liens de reconfiguration (JTAG (*Joint Test Action Group*)) et de transfert de données (PCIe). Ils sont souvent comparés aux circuits spécialisés, les ASIC, car ils remplissent des fonctionnalités proches : cependant, la possibilité de reprogrammation ouvre des domaines d'applications pour lesquels il est difficilement réalisable d'effectuer une mise à jour du matériel (par exemple un satellite) ou qui nécessite des mises à jour fréquentes (par exemple des tâches d'inférence en IA). L'évolution des technologies dans le domaine des semi-conducteurs a permis de rendre les FPGA plus efficaces [Kuon and Rose, 2007] en combinant les performances calculatoires et l'efficacité énergétique des architectures spécialisées face aux processeurs généralistes.

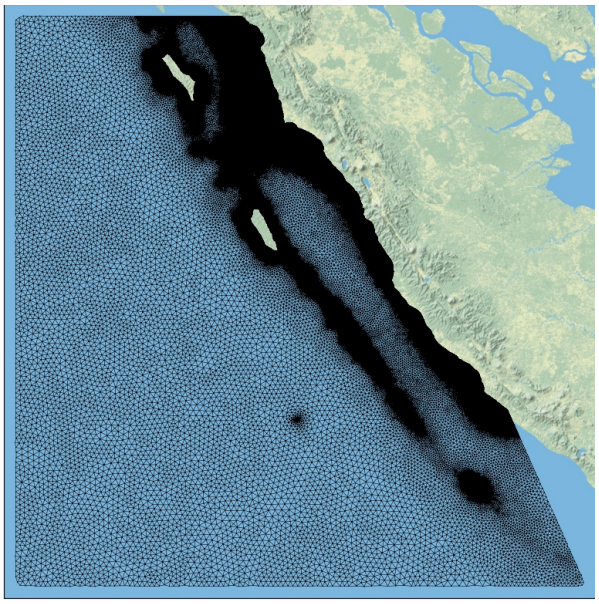
Aujourd'hui les FPGA sont utilisés dans divers domaines opérationnels se prêtant bien à l'architecture orientée traitement de flux de données en ligne : une utilisation classique est la reconnaissance d'expressions régulières dans les flux réseaux, par exemple pour de la détection d'attaques et d'intrusions qui nécessitent de l'analyse de paquets à haute fréquence [Sidhu and Prasanna, 2001]. Un autre exemple est l'utilisation des FPGA pour des tâches d'inférence IA basées sur des réseaux de neurones convolutionnels dans le contexte de l'embarqué [Abdelouahab et al., 2018], notamment avec l'essor du *New Space* et des formats de satellite *CubeSat* favorisant l'emploi de COTS (*Components off-the-shelf*) pour l'électronique embarquée [Arnold et al., 2012]. Les infrastructures *cloud* intègrent aussi des FPGA [Turan and Verbauwhede, 2021], par exemple dans le moteur de recherche Microsoft Bing [Putnam et al., 2015] ou chez les fournisseurs de service PAAS (*Platform as a Service*) comme cela est le cas avec l'offre Amazon Web Service [Amazon Web Service, 2016].

Le FPGA dans le contexte du HPC. De nombreuses contributions étudient cette possibilité depuis les années 2010, encouragées par la promesse d'architectures de calcul plus efficaces en énergie et le développement d'environnements de conception plus accessibles par les géants du secteur Altera/Intel et Xilinx/AMD. Par ailleurs le support d'OpenCL, un langage de programmation proche du flot-de-données déjà largement utilisé pour exploiter le parallélisme des GPU, permet de porter les codes numériques existants ou de les adapter plus simplement [Czajkowski et al., 2012]. De nombreux travaux visent à évaluer les performances calculatoires et énergétique de codes numériques exécutés sur des CPU, des GPU et des FPGA. Parmi les noyaux de calcul représentatifs du HPC ayant été évalués sur FPGA nous pouvons citer la mécanique des fluides avec des maillages non-structurés [Andres et al., 2009], les convolutions 3D [Zohouri et al., 2018], la mécanique du solide avec la méthode des éléments finis et les systèmes multicorps [Sozzo et al., 2017], la transformation de Fourier rapide 3D [Sanaullah and Herbordt, 2018] ou encore l'algèbre linéaire creuse [Zeni et al., 2021]. Dans ces travaux, il ressort que la vitesse de calcul des GPU et des FPGA est comprise entre 30 et 240 fois plus rapide que les CPU, les deux accélérateurs ne pouvant être départagés car les résultats sont très dépendants de la nature des calculs. Sur le plan de la consommation d'énergie cependant les FPGA sont considérés 1,8 à 2,7 fois plus efficaces que les GPU et 40 à 400 fois plus efficaces que les CPU.

Malgré leurs atouts indéniables, les FPGA n'ont véritablement pas intégré les grands systèmes de calcul, en témoigne leur absence visible des listes du TOP500 et du GREEN500. L'une des principales raisons réside dans le modèle de programmation bien éloigné des standards logiciels. Là où un développeur d'applications raisonne sur une séquence d'instructions et d'accès à des données (finalement largement basé sur l'architecture de Von Neumann), la programmation d'un FPGA nécessite une compréhension fine de l'organisation matérielle et de la logique régissant la propagation



(a) Représentation binaire de la matrice creuse F2 issue de la collection de matrices de l'Université de Floride [Davis and Hu, 2011]. Cette matrice carré contient 71505 lignes et 0.1% d'éléments non-nuls.



(b) Représentation d'un maillage issu du Projet H2020 LEXIS [Goubier et al., 2020a], suivant la zone de subduction de la fosse de Java face à l'île de Sumatra. Les longueurs d'arêtes sont comprises entre 100 mètres et 15 km.

FIGURE 3.15 – Deux exemples de structures de données utilisées dans le cadre de ces travaux sur les FPGA.

des signaux numériques. Les outils de CAO (Conception assistée par ordinateur) permettent aujourd'hui de gagner en abstraction, notamment en décrivant dans un langage haut-niveau l'interconnexion de blocs logiques sur étagère. Cependant les phases d'optimisation et de mise au point nécessitent de l'expertise métier afin de projeter efficacement la circuiterie sur la puce, de minimiser les latences bout-en-bout en évitant par exemple la génération de bascules (*latch*) introduisant plus de séquentialité dans le chemin et donc de la latence. Ce besoin d'expertise pointue dans un secteur de niche constitue un frein pour faire le choix du FPGA dans un projet opérationnel.

Dans la littérature, de nombreux travaux montrent qu'il existe un enjeu réel à proposer des outils permettant de rendre transparent l'utilisation des FPGA dans une base de code logicielle plus *classique*. Avec l'essor des pratiques liées à l'IA, de tels outils sont proposés pour des applications d'apprentissage machine et d'inférence de réseaux de neurones. C'est par exemple le cas avec la suite Xilinx Vitis AI [Kathail, 2020] ou l'environnement de conception N2D2 du CEA [CEA, 2017]. D'autres initiatives visent des applications plus larges pour du calcul numérique : un portage de certaines directives du langage de programmation parallèle OpenMP est proposé dans [Sommer et al., 2017]. Une contribution proche d'OpenMP est proposée par le BSC avec OmpSs [Filgueras et al., 2014]. Dans le domaine de l'informatique dans les nuages, Blaze [Huang et al., 2016] propose de mutualiser les FPGA au sien d'une même plateforme et de les exploiter avec des outils orientés service. Un dernier exemple est l'intergiciel d'orchestration et d'ordonnancement de tâches sur machines hétérogènes StarPu [Augonnet et al., 2011] dans lequel des travaux préliminaires ont été menés pour cibler les FPGA [Christodoulis et al., 2018]. Une validation fonctionnelle de l'approche a été effectuée dans le cadre de la multiplication de matrices. Il est aussi possible de distribuer StarPu sur plusieurs nœuds de calcul avec MPI [Augonnet et al., 2012], l'ensemble permettant alors d'abstraire pour le développeur une plateforme répartie comprenant des ressources de calcul hétérogènes. Enfin, de récents efforts portent sur la compilation automatique source-à-source pour proposer une alternative aux chaînes de compilation propriétaires avec le projet OpenFPGA [Tang et al., 2020].

À un niveau de développement plus proche du matériel, des protocoles de communication et des architectures réseau inter-puces ont été récemment proposés pour unifier les accès aux mémoires entre des processeurs et des accélérateurs localisés sur un même nœud de calcul. C'est par exemple le cas pour OpenCAPI [Stuecheli et al., 2018], Intel QuickPath Interconnect [Mutnury et al., 2010], AMD HyperTransport (avec une interface FPGA [Litz et al., 2009]) ainsi que pour les consortiums CXL/Gen-Z et CCIX. Ces technologies souffrent cependant d'un défaut de normalisation et de généralisation dans les systèmes de calcul : lorsque le consortium a dépassé le stade des spécifications, la solution est appliquée et distribuée pour une architecture processeur spécifique, à l'instar d'OpenCAPI principalement implémenté dans les processeurs IBM Power 9 et 10, peu répandus dans les infrastructures de calcul en comparaison des Intel Xeon et AMD Epyc.

D'une manière plus générale, les systèmes présentés dans cette section ne permettent pas de considérer le FPGA comme une unité de traitement à part entière, au même titre qu'un CPU : il reste cantonné au rôle d'accélérateur piloté par un processeur hôte. Les contributions suivantes visent à rendre le FPGA acteur du traitement, notamment à travers

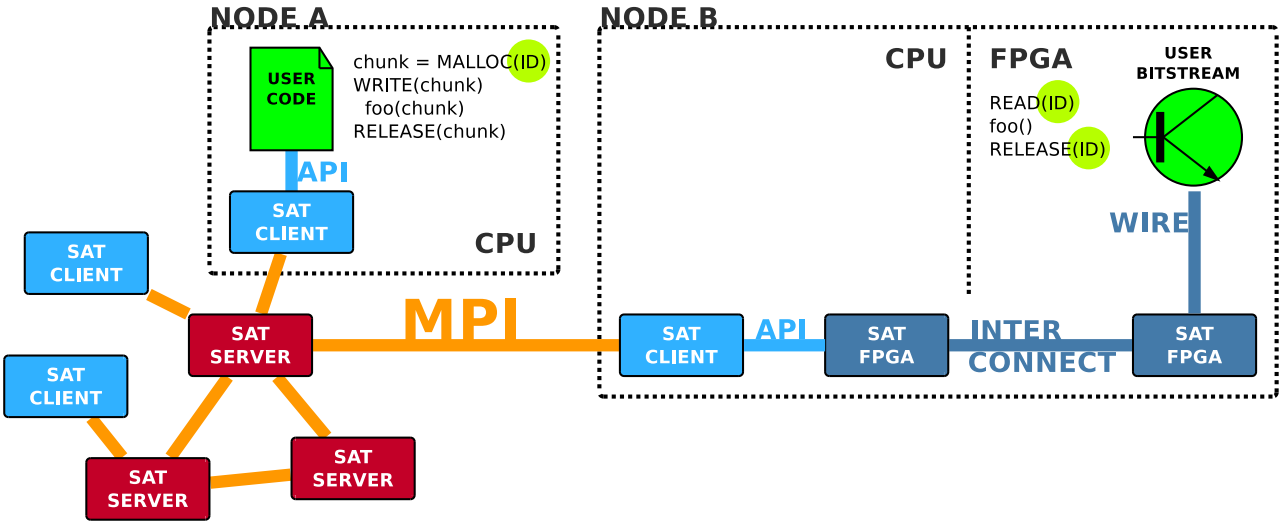


FIGURE 3.16 – Architecture logicielle et matérielle conçue pour intégrer un FPGA dans une MVP. Le code utilisateur exécuté sur le CPU du nœud A et le circuit utilisateur (*bitstream*) chargé sur le FPGA du nœud B accèdent au même espace logique de la MVP et peuvent accéder aux données partagées en utilisant les identifiants uniques ID.

la possibilité d'initier des accès aux données partagées, tout en offrant une interface unifiée avec les autres PE. En cela, l'utilisation d'une MVP permet d'abstraire la couche de gestion des données et de poser les bases d'un modèle de programmation commun.

Intégrer un FPGA dans une MVP. L'une des difficultés inhérentes à l'utilisation des FPGA réside dans le modèle de programmation hybride composé d'un coté par le constructeur de la puce (VHDL, Verilog, SystemC) et de l'autre par l'applicatif en charge de son intégration vers l'extérieur. L'un des points bloquant pour briser le modèle de programmation hybride est l'accès aux données : la plupart des architectures hétérogènes composées de processeurs hôtes et d'accélérateurs nécessitent d'exprimer séparément les accès entre processeurs et les accès entre processeurs et accélérateurs. C'est par exemple le cas pour des applications MPI+CUDA dans lesquelles les données sont explicitement déplacées entre les nœuds avec l'API MPI puis chargées dans le GPGPU à l'aide d'une seconde API.

Intégrer un FPGA dans une MVP dépasse le cadre de la simplification et de l'uniformisation des accès aux données partagées : c'est un moyen de rendre le noyau de calcul déployé sur l'accélérateur capable d'initiatives. Cette approche s'oppose au modèle maître-esclave dans lequel le FPGA subit les données à traiter en entrée, comme un simple processeur de flux. Si la logique programmable ne permet pas de concevoir des programmes avec le même degré d'expressivité qu'un logiciel exécuté sur un CPU [2], il n'en reste pas moins que donner l'initiative d'accès à des données ouvre de nouvelles possibilités d'implémentation de noyaux de calcul. C'est notamment le cas des noyaux présentant des accès irréguliers tels que la multiplication de matrices creuses (figure 3.15a) ou des opérations sur des maillages non-structurés (figure 3.15b). Dans ces deux exemples, les accès aux éléments lors d'une itération du calcul dépendent d'une itération précédente, les rendant difficilement prévisibles. Dans le cas réaliste où les données (les matrices, les maillages) ne peuvent être intégralement chargées dans la mémoire locale de l'accélérateur, sa capacité à exprimer et initier les prochains accès devient un atout.

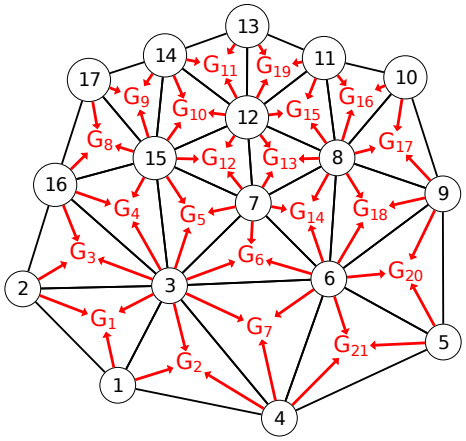
Une architecture permettant d'intégrer un FPGA dans la MVP SAT est proposée dans [Lenormand et al., 2019] et les grands principes sont exprimés dans la figure 3.16. Ce travail a été réalisé dans le cadre de la thèse d'Erwan Lenormand [Lenormand, 2022], soutenue en 2022, que j'ai proposée et co-encadrée, et qui s'est déroulée sous la direction d'Henri-Pierre Charles au CEA. Dans cette proposition, un nœud FPGA est intégré à la MVP par le truchement d'un client SAT logiciel, sur le principe du *proxy*. En lieu et place du code utilisateur, une interface mixte logicielle-matérielle SAT FPGA est en charge de traduire les requêtes d'accès à des *chunks* émises dans des files matérielles par le noyau de calcul utilisateur (le *bitstream* déployé sur le FPGA), en une séquence de primitives de l'API SAT sur le client logiciel. Cette approche présente deux bonnes propriétés : 1) pour la MVP SAT, le FPGA est vu comme un client classique et de ce fait, aucun traitement particulier, notamment sur les protocoles de cohérence des données, n'est requis. L'intégration est donc **transparente** et 2) pour le noyau de calcul déployé sur le FPGA, le moyen de désigner les données partagées repose sur le même espace logique que la MVP (les ID dans la figure 3.16). Il y a donc **continuité** de l'espace logique entre les processeurs et les accélérateurs qui ne nécessite pas de traduction d'adresses, et donc un surcoût pour gérer des tables associatives (à l'instar d'une TLB (*Translation Lookaside Buffer*)).

Si cette architecture n'a pu être implémentée intégralement pour des raisons de temps et d'ampleur de la tâche, la démarche a donné lieu à plusieurs contributions fondamentales pour sa mise en place. Une première contribution

2. Sur le plan de la programmabilité et non de la théorie.

consiste à proposer un outil de simulation pour caractériser le comportement d'un noyau de calcul sur FPGA et ainsi dimensionner les interfaces avec la MVP. Une deuxième contribution est un modèle de gestion des données visant à structurer et réorganiser les données en un flux utilisé pour masquer la latence d'accès entre le FPGA et la MVP.

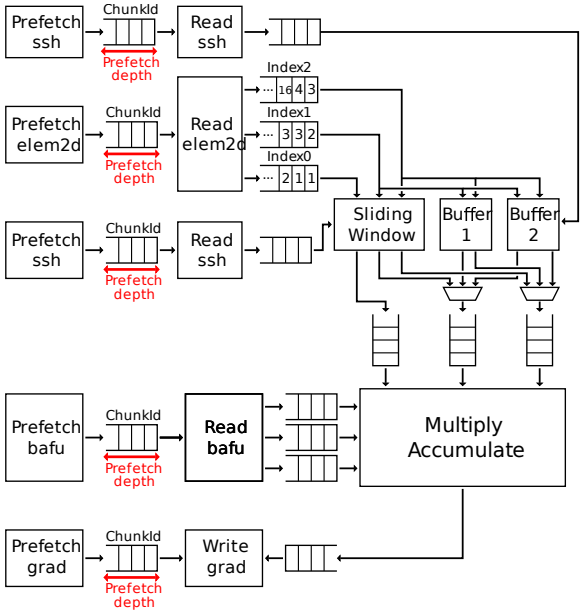
Problématiques pour l'intégration du FPGA. Prendre en compte un accélérateur dans un code numérique nécessite de l'expertise et de la mise au point pour obtenir les performances espérées. Un problème classique rencontré est la famine de données, dans lequel les unités de traitements de l'accélérateur se retrouvent sous-utilisées faute de données suffisantes transférées au bon moment pour faire avancer le calcul. Pour les GPU, une pratique courante est de charger les données en mémoire interne de l'accélérateur, de lancer le calcul, de récupérer le résultat puis de recommencer. Ainsi, le noyau de calcul dispose de toutes les données en mémoire physique. Cette approche est limitée par la taille de la mémoire, de l'ordre de quelques dizaines de gigaoctets pour les plus grosses cartes. Une autre approche est d'alimenter l'accélérateur sous forme de flux, ce qui pose notamment deux problèmes : 1) un problème de dimensionnement du matériel : par exemple les canaux de communication, la profondeur des files de transfert (FIFO) et 2) un problème de réutilisation des données pour profiter des effets de cache : par exemple la granularité adaptée pour la représentation des données.



(a) Maillage triangulaire utilisé pour le calcul du gradient.

```
for(i = 0; i < nb_triangles; i++) {
   grad [ i ] = bafu [ i ] [ 0 ] * ssh [ elem [ i ] [ 0 ] ]
              + bafu [ i ] [ 1 ] * ssh [ elem [ i ] [ 1 ] ]
              + bafu [ i ] [ 2 ] * ssh [ elem [ i ] [ 2 ] ] ;
}
```

(b) Pseudo-code correspondant au calcul du gradient.

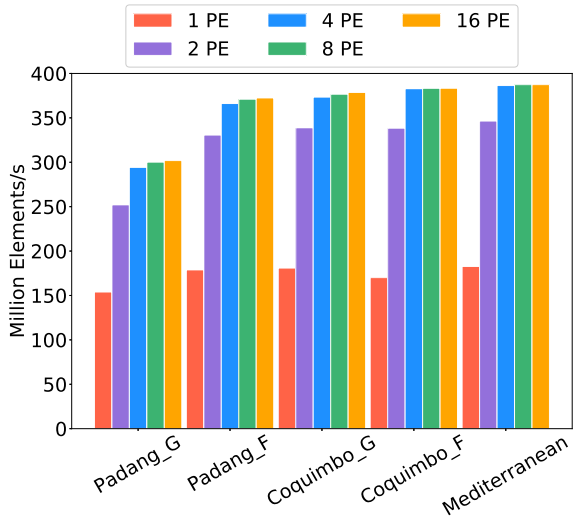


(c) Architecture proposée pour le calcul du gradient.

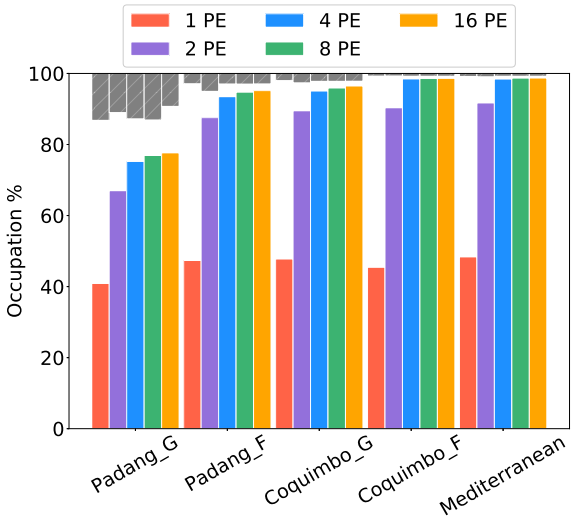
FIGURE 3.17 – Du logiciel au matériel : architecture d'un circuit déployable sur FPGA correspondant à une étape du noyau de calcul implémenté dans le code numérique TsunAWI [Goubier et al., 2020b] développé par l'institut Alfred Wegener (AWI). TsunAWI permet de simuler, par la méthode des éléments finis, la propagation des vagues dans le cas d'un séisme afin d'anticiper les vagues de submersion sur le littoral. Ces figures sont issues du manuscrit de thèse d'Erwan Lenormand [Lenormand, 2022].

Un outil de simulation précis au cycle. Un simulateur de noyau de calcul sur FPGA a été proposé pour répondre à la question du dimensionnement matériel [Lenormand et al., 2020]. L'enjeu est de pouvoir alimenter le noyau en données tout en limitant la taille du circuit, c'est-à-dire le nombre d'unités reconfigurables à allouer sur le FPGA. Le simulateur est précis au cycle et déroule les différentes étapes d'un circuit logique cadencé sur une horloge globale. Les latences de communication, largeur de bus de communication et les fréquences de fonctionnement sont paramétrées suivant les spécifications de puces FPGA documentées. L'utilisation du simulateur est hybride : la MVP SAT est déployée sur une infrastructure distribuée et un ou plusieurs clients s'interfacent avec une instance du simulateur, reproduisant le comportement du noyau de calcul sur un FPGA. Cette approche hybride permet de reproduire une dispersion réaliste des données sur les serveurs de la mémoire partagée, et donc d'obtenir des statistiques légitimes sur les transferts induits par le calcul. L'usage du simulateur permet de plus une approche exploratoire dans laquelle les paramètres du FPGA *émulé* peuvent être adaptés et évalués.

Cette approche a été évaluée sur deux applications : un noyau de multiplication de matrices creuses (plus d'informations sur ce cas d'étude sont disponibles dans [Lenormand, 2022]) et TsunAWI [Goubier et al., 2020b], un code de simulation de propagation de vagues dans le cadre d'un programme d'alerte vague-submersion développé par l'institut Alfred Wegener (AWI) en Allemagne. Ce travail a été effectué dans le cadre du projet H2020 LEXIS [Goubier et al., 2020a] dont l'un des objectifs est d'accélérer ce code numérique. La figure 3.17 illustre comment à partir d'un maillage



(a) Performance calculatoire en nombre de triangles traités par seconde pour différents degrés de parallélisme (nombre de PE).



(b) Taux d'occupation du contrôleur mémoire régissant les transferts entre le répertoire de *chunks* et les PE. Les barres grises correspondent au chargement du premier *chunk* pour initialiser le *pipeline*.

FIGURE 3.18 – Résultats de simulation hybride entre la MVP SAT et le simulateur précis au cycle de noyau de calcul sur FPGA. Le noyau simulé correspond au calcul du gradient dans le code TsunAWI. Évaluation effectuée sur 5 maillages de taille et de granularité différentes.

triangulaire (3.17a) et d'un algorithme de calcul de gradient (3.17b) un circuit logique (3.17c) est proposé pour porter cette étape du calcul de TsunAWI sur un FPGA en utilisant l'interface d'accès aux *chunks* de la MVP SAT. Le découpage du maillage en *chunks* est un point déterminant pour obtenir de bonnes performances calculatoires, et ce pour plusieurs raisons : 1) une granularité trop faible (un petit nombre de triangles par *chunk*) implique plus d'accès distants et donc de latence pour le noyau de calcul. 2) Une granularité trop importante expose au problème du faux-partage dans la MVP. Enfin, 3) le choix de colocaliser certains sommets du graphe dans un même *chunk* a une conséquence directe sur le nombre d'accès à la MVP pour reconstituer les triangles dans le noyau de calcul. Ce dernier point a motivé des travaux sur l'ordonnancement des sommets des graphes non-structurés à l'aide d'une courbe de remplissage continue de Hilbert [Hilbert, 1935] visant à améliorer la localité spatiale des triangles [Lenormand et al., 2021].

Sur cet exemple, la problématique du dimensionnement consiste à 1) déterminer la taille des FIFO de préchargement des *chunks* et 2) déterminer le nombre d'instances de ce circuit à déployer pour paralléliser le traitement tout en évitant de saturer le bus de données entre le contrôleur mémoire du FPGA et les PE. La figure 3.18 présente les résultats de simulation obtenus pour le code TsunAWI sur 5 maillages classés par taille croissante, de 230000 à 5 millions de triangles. Le FPGA choisi pour calibrer la simulation est un Xilinx Virtex-7 fonctionnant à 200 MHz. Pour chaque maillage, le nombre d'instances du circuit de traitement évalués, le degré de parallélisme, est de 1 à 16 PE. La figure 3.18a indique les performances calculatoires en nombre de triangles traités par seconde et la figure 3.18b représente le taux d'occupation du contrôleur mémoire sur le FPGA.

Cette expérience a permis de montrer que pour ce noyau de calcul, une accélération est obtenue jusqu'à un parallélisme de 4. Au delà, un phénomène de famine de données apparaît pour les PE, provoqué par la saturation du contrôleur mémoire pour les maillages volumineux et par la latence d'accès aux données pour les plus petits maillages. Des expériences additionnelles sont présentées dans [Lenormand, 2022], permettant d'encadrer la taille des FIFO et de caractériser le comportement du système lorsque les *chunks* sont présents ou non sur le nœud hôte. Enfin, ces expérimentations ont montré l'importance d'organiser et de structurer les données de telle sorte à limiter les transferts mémoire, le faux partage, et à l'inverse bénéficier de la localité spatiale des accès. Si ce problème est largement étudié pour les processeurs multi-cœurs, son application à un système hybride composé d'une MVP logicielle et d'un circuit matériel sur FPGA reste novateur.

En conclusion, l'adoption massive des processeurs programmables dans les contextes HPC et embarqués reste conditionnée par la mise à disposition d'outils de programmation et de déploiement suffisamment haut-niveau pour ne plus reposer sur l'expertise pointue en architecture matérielle, ce qui s'avère bien éloigné du domaine applicatif et du génie logiciel. Aujourd'hui, de tels efforts sont concédés par les grands acteurs du secteur (Altera/Intel et Xilinx/AMD) dans le domaine de l'apprentissage machine et de l'inférence, notamment portés par les applications IA. Un autre élément moteur est la concurrence ouverte menée par les géants du numérique sur le secteur des environnements d'apprentissage machine et des accélérateurs matériels pour l'IA. C'est par exemple le cas avec Google (TensorFlow, Tensor Processing

Unit, Tensor Chip), Facebook (PyTorch), Uber (Horovod) ou encore Amazon (AWS ML). Il faut donc des contraintes opérationnelles sérieuses pour faire le choix du FPGA et de son écosystème fermé, plutôt que de reposer sur des environnements de développement libres, suivis par une grande communauté d'utilisateurs et ciblant des architectures matérielles plus conventionnelles (CPU et GPU).

Dans ce contexte, les travaux menés pendant la thèse d'Erwan Lenormand contribuent à rendre l'utilisation des FPGA plus simple, en traitant cette problématique à travers le prisme de la gestion des données partagées. Cette approche nécessite toujours un développement des noyaux de calcul dans un langage spécialisé HDL mais les mécanismes d'accès aux données distantes, sur le processeur hôte ou sur un autre nœud, sont rendus transparents pour le développeur et ne nécessitent plus de transferts explicites. Les études préliminaires montrent que pour obtenir une implémentation efficace de SAT FPGA, l'architecture doit reposer sur de nouvelles technologies de gestion des communications et de la mémoire, notamment des mémoires à grande bande passante reposant sur la technologie HBM (*High-Bandwidth Memory*), ou sur des protocoles de communication pair-à-pair pour participants hétérogènes tels que OpenCAPI.

Une piste d'étude prometteuse reste l'intégration de l'intergiciel de passage de message MPI -sur lequel repose la MVP SAT- directement sur le FPGA, permettant ainsi de supprimer les intermédiaires entre le circuit utilisateur et les serveurs de la MVP. Ainsi, un client SAT peut être directement implémenté sur le FPGA, comprenant l'automate complet pour envoyer et recevoir des messages de contrôle et de données avec les serveurs. Un tel portage est complexe car en quelques dizaines d'années, l'API MPI s'est beaucoup étoffée avec des primitives collectives et des communications unidirectionnelles. Quelques initiatives visent cependant à porter une partie de l'API sur des FPGA : Dans [Williams et al., 2006] un portage de MPI est réalisé sur un processeur *softcore*, c'est-à-dire un processeur instancié dans la logique du FPGA et donc capable d'exécuter un système d'exploitation complet. L'approche est intéressante car elle permet de déployer du logiciel sur la puce. Cependant cette facilité a un coût important en performance et en empreinte mémoire. D'autres approches [Gao et al., 2009, Arap and Swany, 2014] ciblent des opérations collectives telles que les barrières ou les jointures de résultats. Ces contributions rejoignent les efforts récents pour l'intégration de calculs intermédiaires sur les flux de données directement dans les routeurs réseaux, comme cela est le cas pour les processeurs de données Nvidia Mellanox BlueField [Sarkauskas et al., 2021]. Il semble que la contribution la plus complète soit proposée dans TMD-MPI [Saldaña and Chow, 2006] avec un NoC spécifique déployé sur le FPGA pour échanger des messages MPI entre les nœuds, et développé pour un calculateur multi-FPGA dédié à l'étude de la dynamique moléculaire à l'Université de Toronto. Toutes ces initiatives montrent qu'il reste du chemin à parcourir pour rendre le FPGA attractif et en faire un choix naturel au même titre qu'un accélérateur GPGPU. Cela passera probablement par des standards ouverts et des environnements de développement accessibles offrant un haut-niveau d'abstraction du matériel.

3.3 Discussion

Le développement de la MVP logicielle SAT s'est déroulé dans le contexte de projets européens visant à concevoir des serveurs de calcul hétérogènes. Ceci a permis d'identifier les points techniques durs à résoudre pour offrir un environnement de programmation qui utilise efficacement les ressources. En ce sens, SAT répond à deux problématiques principales : 1) elle offre un modèle de programmation unifié à l'échelle de la plateforme et masque la gestion des données partagées et 2) les outils d'aide à la décision répondent à la problématique de l'utilisation efficace des ressources en offrant des compromis entre performance calculatoire et consommation d'énergie rendus possibles par la diversité des unités de traitement. Face à des intergiciels plus classiquement déployés sur les machines distribuées, nous avons démontré que la MVP offre une alternative sérieuse, en témoignent les démonstrations effectuées sur les machines Christmann dans les différents salons technologiques (ISC, Teratec). Pour cela, plusieurs thématiques de recherche et développement ont dû être menées pour obtenir une pile logicielle complète : 1) un modèle de programmation en mémoire partagée basé sur un partitionnement des données en *chunks*, 2) une chaîne de compilation pour le dimensionnement de SAT, la sélection des ressources et le placement des tâches, 3) un intergiciel distribué implémentant un système de stockage et des protocoles de cohérence des données. Si le cœur de la conception de SAT a pu être réalisé seul, l'ensemble des contributions menant au succès du projet est le fruit de collaborations ayant alimenté de nombreuses réflexions, notamment sur le plan de l'optimisation, de la recherche opérationnelle, de la sécurité des données et des technologies FPGA.

Travaux inachevés. [*À propos d'une API haut-niveau*] SAT propose un modèle de tâches et une API spécifique basée sur la cohérence à l'entrée (entry/scope consistency qui nécessite l'acquisition et la libération de verrou dans le code utilisateur. Les applications doivent donc être écrites ou adaptées pour son utilisation. Ce constat est partagé avec les systèmes d'entrées-sorties pour le HPC : les applications doivent par exemple être écrites spécifiquement pour utiliser les systèmes de fichiers distribués HDF5, Lustre ou FTI. Afin d'abstraire les interactions avec ces systèmes, l'intergiciel PDI [Roussel et al., 2017] développé au CEA (Maison de la Simulation) propose une interface unique et un système de greffons pour les différentes technologies. Dans ce travail mené avec Julien Bigot (CEA), un greffon PDI a été développé pour SAT en C++ permettant de transférer de manière transparente les appels aux systèmes de fichiers classiques vers du stockage en mémoire partagée. Cette approche vise notamment le couplage de codes de calcul numériques afin d'accélérer les échanges de données souvent réalisés par système de fichiers à chaque pas de simulation et qui peuvent

dans ce cas bénéficier d'une solution en mémoire physique.

[*À propos de la sécurité des données*] Les accès aux données partagées dans la MVP SAT peuvent être effectués par toute instance de client déployée dans l'application. Par défaut, seuls les identifiants des données dans l'espace d'adressage logique sont nécessaires pour effectuer un accès, ce qui constitue un secret raisonnable dans un contexte lié à l'expérimentation, mais peu envisageable dans un contexte opérationnel. Un premier travail a consisté à offrir une couche de gestion des adresses logiques en se basant sur une fonction de hachage. Ainsi, à chaque écriture d'un *chunk*, un nouvel identifiant est calculé en appliquant la fonction Bernstein hash djb2 [Yigit, 2003] sur le contenu (le champ *data*). Cette stratégie apporte deux propriétés : 1) les accès sont par construction en lecture seule car toute modification d'une donnée partagée est stockée à une adresse logique différente (des collisions peuvent cependant intervenir) et 2) la fonction de hachage offre une répartition uniforme des adresses logiques, ce qui rend plus complexes les attaques par force brute. Un deuxième travail [Stan et al., 2020] a été effectué avec Oana Stan (CEA) pour gérer les autorisations d'accès aux données partagées à l'aide du chiffrement par attributs (ABE). Ce chiffrement est asymétrique et permet, à partir d'un ensemble d'attributs et de clés privées d'appliquer une politique d'accès, par exemple basée sur des rôles. Ce travail a été réalisé lors du stage de Master de Louis Syoen (Université Paris-Diderot).

Retour d'expérience. Parfois, pour avancer, il est nécessaire de déconstruire, puis de tout recommencer. Alors que faut-il retenir des choix de conception effectués dans SAT ?

[*À propos de MPI*] Le retour d'expérience est ici mitigé : MPI apporte une grande facilité d'implémentation pour l'algorithmique répartie, notamment pour la phase de construction du réseau logique des participants. Les opérations collectives et les communications unidirectionnelles n'ont pas été utilisées, ce qui laisse des possibilités d'amélioration sur le plan des performances. L'implémentation sur MPI s'accompagne d'une topologie statique : il n'est pas possible d'ajouter ou de supprimer des processus en cours d'exécution. La conséquence pour SAT est de ne pas pouvoir s'étendre sur des nœuds au gré des besoins applicatifs, ou à l'inverse se rétracter pour laisser la place à d'autres applications. Cette propriété n'a pas été nécessaire pour le déploiement de SAT sur des micro-serveurs hétérogènes, mais d'autres usages peuvent nécessiter de la dynamicité, notamment en présence de ressources volatiles et de systèmes massivement multi-utilisateurs. Le standard MPI permet de la malléabilité sur les processus, ce qui peut être considéré pour une future implémentation. Deux autres pistes peuvent être envisagées : 1) celle explorée par JuxMem qui consiste à utiliser un intergiciel pair-à-pair dynamique, avec un coût de construction du réseau logique élevé et une grande latence lors des phases de découverte des nœuds et 2) la piste du réseau logique construit au dessus de ZeroMQ pour bénéficier des performances dans les communications mais au prix d'un important effort de développement.

[*À propos du langage C*] Dans le monde de l'embarqué et du logiciel critique le langage C, et à fortiori le ANSI C (C89), est un critère de réussite pour obtenir un code sûr, performant et portable. Ce choix a en effet permis un déploiement rapide et sans surprise de SAT sur toutes les plateformes rencontrées, quelque soit l'architecture processeur et le système d'exploitation (en base Unix). Le langage C a de plus autorisé une gestion fine de la mémoire physique, par exemple dans le cas d'une allocation contiguë de plusieurs *chunks*. Pour autant, entre 2016 et 2022, le paysage des langages et environnements de programmation a profondément changé, bousculé par la nécessité de déployer du code portable performant sur des architectures de plus en plus embarquées. Aujourd'hui, implémenter une MVP logicielle en langage C n'est pas pertinent, pour plusieurs raisons : 1) Une MVP a une complexité calculatoire très faible qui ne nécessite pas un langage proche de la machine. La performance du système repose entièrement sur l'efficacité des protocoles de cohérence et le choix d'un réseau de communication adapté. 2) Le temps de développement en C est très long et la mise au point accablante, focalisée sur des opérations bas-niveau (la manipulation de chaînes de caractères) alors que l'essentiel de la difficulté se trouve dans l'algorithmique répartie, à un niveau de conception bien plus abstrait. 3) Il existe désormais des langages haut-niveau tels que Python ayant atteint une maturité suffisante pour être performant sur le plan calculatoire tout en permettant d'écrire des algorithmes avec un grand degré de concision. De nombreuses bibliothèques Python permettent d'exploiter le parallélisme (RAY [Moritz et al., 2018]) et les infrastructures distribuées (Mpi4Py [Smith, 2016]), ce qui s'avère être aujourd'hui un bon choix pour prototyper une MVP moderne, en phase avec les applications en IA et en science des données actuelles. De plus, les nouveaux types et structures de données introduits par des logiciels tels que TensorFlow et PyTorch peuvent servir de base pour un découpage efficace en *chunks* et ainsi optimiser l'espace mémoire et la disponibilité des données dans les calculs distribués, notamment pour l'apprentissage fédéré.

[*À propos de la propriété intellectuelle*] Le développement de SAT a été réalisé sur le modèle propriétaire, c'est-à-dire que le code source n'a pas fait l'objet d'une autorisation de publication sous licence publique. Cette stratégie est difficilement défendable dans le contexte de projets européens et constitue un frein évident pour la mise en place de collaborations, pour la reproductibilité des résultats publiés, ainsi que pour la pérennité de la brique technologique. Depuis une quinzaine d'années maintenant, être moteur de l'innovation, et à fortiori dans le milieu de la recherche, ne consiste plus à garder un logiciel secret : cela consiste au contraire à créer une masse critique autour du projet incluant la communauté scientifique, à construire l'expertise en interne et à capitaliser sur les retours d'expérience dans le temps long. De cette manière il est possible de provoquer l'émergence d'idées nouvelles tout en conservant une avance stratégique.

□

Chapitre 4

Conclusion et perspectives

4.1 Conclusion

Pour un projet de recherche, une dizaine d'années représente une période longue et courte à la fois. Cette période a été marquée par le développement rapide des systèmes embarqués à forte capacité calculatoire peu avant 2010, avec des processeurs économes en énergie pour les *smartphones*, les objets connectés et les véhicules autonomes, ainsi que des architectures massivement parallèles conçues pour optimiser le coût de calcul par watt. À partir de 2014 les architectures hétérogènes constituent une évolution logique de ces systèmes embarqués en agrégeant différentes unités de calcul dédiées à des tâches spécifiques. Enfin, l'avènement de l'intelligence artificielle dans tous les domaines du quotidien met l'accent sur des accélérateurs spécialisés ou reprogrammables dont il faut assurer une intégration efficace dans les systèmes existants. Dans cette temporalité, l'étude des machines parallèles, réparties et hétérogènes a donné lieu à de nombreuses thématiques et pistes de réflexion sur les aspects matériels et logiciels. Différentes architectures à l'état de l'art ont été traitées, du processeur massivement parallèle au micro-serveur distribué hétérogène. La nouveauté de ces architectures provient de deux stratégies différentes : l'accumulation massives des unités de traitement pour les many-cœurs en comparaison des multi-cœurs, et l'intégration de nombreux nœuds de calcul et de stockage hétérogènes dans un même rack pour les micro-serveurs en comparaison des architectures homogènes des grappes de calcul. Bien que conçues selon des mécanismes connus, l'accumulation et la diversification des ressources, ces deux architectures souffrent de la même problématique de programmabilité : comment exploiter efficacement les unités de calcul tout en masquant la complexité de l'architecture matérielle aux utilisateurs ?

Les travaux présentés dans ce manuscrit apportent des éléments de réponse sur deux plans : le modèle de programmation et le modèle de gestion des données partagées. Ces deux plans sont en pratique intimement liés et les contributions de l'un impliquent bien souvent des contributions dans le second. Pour les architectures many-cœurs, deux modèles de programmation ont été traités : le flot-de-données et la mémoire partagée. Ces deux modèles offrent une haute abstraction de la puce, le premier se focalisant sur le modèle de tâches et le second sur les accès mémoire. La complexité est nécessairement prise en charge par des éléments tiers tels que une chaîne de compilation, un intergiciel d'exécution ou un protocole de cohérence des caches. De tels outils tiers ont été proposés et validés par le tissu industriel (Kalray) et la communauté scientifique. Pour les architectures hétérogènes, un modèle en mémoire partagée a été proposé et une mémoire virtuellement partagée logicielle a été conçue, implémentée et validée dans des démonstrateurs sur plateforme industrielle (Christmann). L'animation de cette thématique ne s'est pas limitée à la MVP : de nombreuses contributions satellites ont été proposées pour faciliter son utilisation et son intégration dans un contexte opérationnel sobre en énergie (modèle de programmation haut-niveau, extension au modèle événementiel, exploration de la configuration du déploiement, maîtrise de l'activité des communications MPI et prise en compte de FPGA).

Le retour d'expérience obtenu sur l'utilisation des many-cœurs et des micro-serveurs est très similaire : proposer un outil d'exploitation de l'architecture de calcul n'est pas suffisant, des outils d'aide à la décision sont bien souvent nécessaires pour guider l'utilisateur. Ainsi, après avoir proposé plusieurs protocoles de cohérence des caches paramétrables, le choix du protocole et sa configuration ne peuvent être laissés à la charge du développeur : une chaîne de compilation pour l'aide à la décision a donc été proposée. Il en va de même pour le déploiement de la MVP SAT dont la topologie logique détermine la performance calculatoire et l'empreinte énergétique. Le choix de cette topologie est complexe et parfois contre-intuitif. Là encore, un outil d'aide à la décision est proposé pour orienter l'utilisateur vers les solutions offrant les meilleurs compromis. Il y a donc à chaque fois un enjeu majeur à offrir des systèmes d'aide à la décision permettant d'accompagner l'expert métier. Ces outils reposent aujourd'hui sur des techniques classiques de recherche opérationnelle : méthodes populationnelles et exploration du voisinage. Il semble cependant naturel que l'optimisation des systèmes pour le HPC et le HPEC bénéficient des méthodes prédictives offertes par l'IA.

4.2 Perspectives

Depuis une quinzaine d'années l'intelligence artificielle rencontre un nouvel essor rendu possible par des avancées notables sur le plan théorique, algorithmique et technologique. Ceci est visible à travers deux faits : 1) l'avènement de l'apprentissage profond, reconnu par un prix Turing en 2018 (LeCun, Bengio et Hinton) et 2) les avancées technologiques spectaculaires sur le plan des architectures de calcul. Ces nouvelles architectures, en partie étudiées dans ce manuscrit, permettent un apprentissage machine à partir d'un grand volume de données dans un temps devenu raisonnable. Elles sont aussi présentes au sein des systèmes embarqués en général, permettant de déployer des tâches d'inférence au plus proche du contexte opérationnel. L'une des particularités de l'IA est de s'immiscer *positivement* dans tous les domaines : la conception assistée par ordinateur, la recherche de solutions optimisées et bien sûr tous les applicatifs métier présentant des tâches fastidieuses, répétitives et suffisamment complexes pour ne pas pouvoir être réalisées par un programme informatique classique.

Fin 2019 je rejoins la Direction Générale de l'Armement (DGA Maîtrise de l'Information) dans un département en cours de création et traitant des problématiques liées à l'IA, à la RO et à la gestion de grands volumes de données (*datascience*). La création de ce département s'inscrit dans la volonté de moderniser les Armées, de faire le lien entre les univers académiques, industriels et les problématiques rencontrées par les opérationnels. Dans ce contexte, mes travaux s'inscrivent dans diverses missions : 1) une première mission est la mise en place de moyens de calculs conséquents dédiés à l'apprentissage machine, la fouille de données, la simulation et l'optimisation. Ceci inclut le choix du matériel à l'état de l'art et son intégration dans un environnement logiciel calqué sur les installations classiques des supercalculateurs. Ceci implique aussi d'adapter les pratiques liées aux activités IA aux outils permettant une exploitation efficace du parallélisme massif de l'infrastructure. 2) Une deuxième mission est la proposition et le suivi de projets au service de la modernisation des Armées. 3) Enfin une troisième mission consiste en l'animation scientifique, ce qui inclut la direction de thèse, le montage de projets avec des partenaires académiques et l'aide pour l'utilisation des mécanismes dédiés à l'innovation (AID).

Les activités menées au sein de la DGA constituent une continuation logique et cohérente des travaux effectués au CEA sur l'exploitation des machines parallèles, distribuées et hétérogènes. L'intelligence artificielle est par nature un client grand compte des environnements de calcul massivement répartis et hétérogènes. L'apprentissage machine nécessite de la puissance de calcul et des systèmes efficaces pour le partage des données : un exemple est l'identification des poids dans un réseau de neurones, ce qui s'effectue aujourd'hui principalement sur des machines multi-GPU à mémoire partagée. Lorsque de telles machines, complexes et donc coûteuses, ne suffisent plus, ce processus calculatoire est distribué sur plusieurs nœuds. L'**apprentissage machine fédéré** est une thématique de recherche en plein essor car deux aspects théoriques durs se rejoignent : les calculs probabilistes attachés aux réseaux de neurones et le non-déterminisme dans l'ordre des évènements inhérents aux systèmes répartis. Il en résulte un système dont les résultats sont difficilement reproductibles. Cette propriété de reproductibilité est pourtant fondamentale pour la **qualification et la certification** des applications IA dans les contextes opérationnels. Une initiative est menée à travers le projet franco-allemand E2Clab [Rosendo et al., 2020] visant à offrir une plateforme d'exploration et de dimensionnement des applications pour la **reproductibilité des calculs**.

E2Clab est aussi une plateforme qui doit permettre d'explorer différentes configurations de déploiement des applications IA. Que ce soit en apprentissage ou en inférence, les environnements de développement et de calcul IA sont particulièrement récents et reposent rarement sur les outils classiques du calcul haute-performance (par exemple MPI). Ils sont aussi pensés pour exprimer l'algorithmique a un plus haut-niveau d'abstraction, plus proche du métier d'expert en mégadonnées que de l'expert en HPC. Dès lors, les choix technologiques et leurs conséquences sur les divers critères de performances sont peu documentés. Il existe donc un besoin de fournir dans un futur proche des outils de décision pour choisir un environnement de développement et configurer son déploiement sur l'infrastructure de calcul. Dans ce travail préliminaire [Prigent et al., 2022], nous avons proposé une **méthodologie pour explorer des compromis** entre le temps de calcul, la qualité du résultat et la dépense énergétique. Cette méthodologie est appliquée sur un cas d'étude concret : elle permet d'orienter le choix de la politique d'apprentissage par renforcement pour paramétrer un système de guidage de parachute autonome.

Un deuxième axe de recherche est la capacité à délocaliser l'apprentissage machine vers les nœuds de calcul embarqués. Cette inversion du flux du savoir répond à plusieurs objectifs : 1) la possibilité de générer de l'information localement, par exemple pour de l'apprentissage en ligne, accessible dans un temps très court, 2) la rationalisation des communications, l'information échangée entre les participants revêtant un caractère sémantique plus concis, par exemple les poids d'un réseau de neurones plutôt qu'un flux d'images et 3) la possibilité de maîtriser la confidentialité des informations brutes localement et de remonter une information non protégée, par exemple un résultat de classification plutôt qu'un modèle. Aujourd'hui, avec la miniaturisation des composants électroniques, l'amélioration de l'autonomie des batteries et la maîtrise de la consommation d'énergie, l'IOT (*Internet of Things*) s'annonce comme un support naturel de décentralisation massive, replaçant le calcul au plus près des utilisateurs. Un exemple est le déploiement de tâches de reconnaissance et de classification d'objets dans le domaine du spatial : l'IA bord. Dans cet exemple, les contraintes liées à l'embarqué sont augmentées par les spécificités du domaine spatial qui impliquent de respecter un facteur de forme, une consommation d'énergie limitée et une architecture matérielle durcie à l'environnement (vibrations, température, radiations). Une approche étudiée actuellement est l'utilisation de FPGA dans les satellites

d'observation afin de bénéficier de son efficacité énergétique et des possibilités de reconfiguration de la circuiterie en cours de mission. L'une des problématiques est d'établir un compromis entre l'empreinte mémoire du noyau de calcul et la qualité des résultats. Dans ce contexte, la thèse de Cédric Gernigon dirigée par Olivier Sentieys (INRIA Rennes) à laquelle je participe aux cotés du CNES, explore les techniques de réduction de précision de la représentation des poids dans les réseaux de neurones pour en diminuer l'empreinte mémoire sur un FPGA. Si ces travaux concernent aujourd'hui les phases d'inférence, ils ouvrent cependant la voie à l'embarquabilité de tâches plus complexes sur plan calculatoire, et notamment l'**apprentissage machine en ligne**.

Un troisième axe de recherche consiste à étudier le **continuum informatique**, c'est-à-dire la continuité de la gestion des calculs et des données entre grands systèmes hétérogènes. Avec la miniaturisation des équipements numériques, plusieurs grandes architectures système ont émergées ces 20 dernières années : par exemple le *cloud computing* à l'échelle des centres de calcul, le *fog computing* à l'échelle des équipements intermédiaires dans le réseau et le *edge computing* à l'échelle des équipements au plus proche des utilisateurs. Ces systèmes diffèrent sur de nombreuses caractéristiques matérielles et environnementales. Une conséquence est de constater que les méthodes et outils utilisés pour les exploiter sont spécifiques. Dans ce contexte, les applications sont développées et déployées sur un unique système. Si des exemples montrent des interactions entre applications exécutées sur des systèmes différents, il est plus rare de trouver une même application déployée sur plusieurs systèmes hétérogènes. Certains scénarios reposent cependant sur la possibilité de migrer une application d'un système à un autre. Par exemple un apprentissage en ligne est initié sur un essaim de drones puis est migré dynamiquement sur un centre HPC, en relocalisant les tâches d'apprentissage sur les ressources du centre de calcul et en redirigeant les flux de données issus des capteurs des drones. Cette thématique sur le continuum informatique est l'objet de la thèse de Cédric Prigent dirigée par Gabriel Antoniu (INRIA Rennes) et que je co-encadre avec Alexandru Costan (INSA Rennes).

Un quatrième axe de recherche se situe à la frontière entre l'apprentissage fédéré et le continuum informatique : il semble raisonnable de penser que les géants du numérique ont un intérêt opérationnel et financier à déporter les calculs liés à l'apprentissage machine sur les terminaux mobiles des utilisateurs. Dans ce contexte, la question de la confidentialité des données pour un apprentissage fédéré entre terminaux utilisateurs est un sujet d'intérêt, pouvant bénéficier de techniques calculatoires basées sur le **chiffrement homomorphe**. Ce type de chiffrement permet d'effectuer des calculs directement sur le chiffré et non sur le clair. Si de récentes avancées théoriques ont permis de rendre cette technique soutenable sur le plan de la complexité calculatoire, il n'est cependant pas possible d'en généraliser l'usage sur une application distribuée complète. Cependant, dans le cadre de l'apprentissage machine, la simplicité des opérations effectuées pour mettre à jour les poids d'un réseau de neurones (multiplication-accumulation) rend l'approche envisageable [Sébert et al., 2021, Boudguiga et al., 2021].

Ces quatre axes de recherche s'inscrivent dans les trois thématiques présentées en début de manuscrit : le **parallélisme** des tâches, la **distribution** des calculs et l'**hétérogénéité** des architectures. Le prisme utilisé n'est cependant plus la gestion des données partagées, mais le support des applications en intelligence artificielle. Ces applications sont porteuses de grandes avancées pratiques, d'innovations techniques et d'usage, ainsi que d'interrogations sociétales touchant à l'éthique. Ces changements sont largement perceptibles dans le quotidien et de nombreux contextes opérationnels vont bénéficier de ces nouvelles approches de résolution de problème. Néanmoins, les systèmes calculatoires sous-jacents pour l'IA partagent les mêmes problématiques que les systèmes complexes plus anciens, notamment pour concevoir des architectures adaptées à la résolution de problèmes de recherche opérationnelle et pour utiliser efficacement les ressources de calcul. Ainsi, les techniques en IA peuvent déjà aider à concevoir les architectures de calcul de demain et générer automatiquement des programmes informatiques efficaces. La prochaine évolution majeure se situe peut-être dans le domaine de la physique : pour le calcul haute-performance en général, il est possible qu'une innovation de rupture provienne de l'**informatique quantique** à travers un paradigme de programmation et un modèle de calcul résolument novateurs. Si les conséquences sur la cryptographie et la résolution de certains problèmes de recherche opérationnelle durs commencent à être entrevues, il est possible que la mécanique quantique réserve de nouvelles singularités à exploiter, comme par exemple l'absence de causalité entre les événements à très petite échelle [Letertre, 2021]. Cette propriété interroge sur la capacité -ou la pertinence- à écrire un ordonnanceur de tâches et un protocole de cohérence des données à l'échelle quantique, ces deux objets constituant la base des systèmes calculatoires tels que nous les connaissons aujourd'hui.

□

Glossaire

API	Interface de programmation (*Application Programming Interface*) 21, 38, 58, 59, 63, 65, 71, 73, 78, 81
ARM	Architecture processeur basée sur un jeu d'instruction RISC (*Acorn/Advanced Risc Management/Machine*) 53
ASIC	Circuit intégré spécialisé (*Application-Specific Integrated Circuit*) 21, 53, 76
CAO	Conception assistée par ordinateur 77
CC	Cohérence de cache (*Cache Coherence*) 15, 16, 42, 43, 45–51
COTS	Composants sur étagère (*Components off-the-shelf*) 76
CPU	Unité centrale de traitement (*Central Processing Unit*) 53, 77, 78, 80
CSDF	*Cyclo-Static Data-Flow* 10, 20–23, 25
CSP	*Communicating Sequential Processes* 13
DAG	Graphe orienté acyclique (*Directed Acyclic Graph*) 68
DMA	Accès direct à la mémoire (*Direct Memory Access*) 25, 34, 35
DSL	Langage dédié (*Domain Specific Language*) 21, 32, 35, 52
DSP	Processeur spécialisé dans le traitement du signal (*Digital Signal Processor*) 76
EEE	Norme ethernet optimisée sur le plan énergétique (*Energy Efficient Ethernet*, IEEE 802.3az) 73
EoP	Equivalence de pointeur (*Equivalence of Pointer*) 33–35
FIFO	*First In First Out* 21, 60, 79, 80
FPGA	Circuit logique programmable (*Field-Programmable Gate Array*) 21, 25, 53–57, 63, 75–81, 83–85
GFLOPS	Nombre d'opérations en virgule flottante par seconde, en million 10^6 (*floating-point operations per second*) 16
GOPS	Nombre d'opérations par seconde, en million 10^6 (*operations per second*) 16
GPGPU	Calcul générique sur processeur graphique (*General-Purpose computing on Graphics Processing Unit*) 13, 15, 17, 21, 53, 78, 81
GPU	Processeur graphique (*Graphics Processing Unit*) 13, 25, 53, 54, 77, 78, 80, 84
HAL	Couche d'abstraction matérielle (*Hardware Abstraction Layer*) 20, 25
HBM	Mémoire à large bande passante (*High-Bandwidth Memory*) 80
HDL	Langage de description de matériel (*Hardware Description Language*) 76, 80
HN	Noeud référent (*Home-Node*) 40–42
HPC	Calcul haute performance (*High-Performance Computing*) 13, 15, 17, 21, 38, 40, 53, 57, 62, 65, 71, 72, 75, 77, 80, 81, 83–85
HPEC	Calcul haute performance embarqué (*High-Performance Embedded Computing*) 13, 65, 83
IA	Intelligence artificielle 15, 51–53, 70, 76, 77, 80, 82–85
IDE	Environnement de développement (*Integrated Development Environment*) 20
IOT	Internet des objets (*Internet of Things*) 84
IPS	Nombre d'instructions par seconde 16
ISS	Simulateur de jeu d'instructions (*Instruction Set Simulator*) 21, 34, 35
JIT	Compilation à la volée (*Just in time compilation*) 41
JTAG	Port d'accès de test (*Joint Test Action Group*) 76
KPN	*Kahn Process Network* 21
LoG	Filtre Laplacien d'une Gaussienne (*Laplacian of Gaussian*) 30–32
LVT	Temps logique vectoriel (*Logical Vector Time*) 23
MIPS	Nombre d'instructions par seconde, en million 10^6 16
ML	Apprentissage machine (*Machine Learning*) 15, 80
MP	Passage de message (*Message Passing*) 17, 20, 25, 65
MPI	*Message Passing Interface* 11, 20, 24, 59, 64–67, 71–75, 77, 78, 81–84
MPPA	Réseau de processeurs massivement parallèles (*Multi Purpose Processor Array*) 13, 15, 38, 52
MRSW	Lecteurs multiples, écrivain unique (*Multiple Readers, Single Writer*) 40

MVP Mémoire virtuellement partagée (*Distributed Shared Memory*) 20, 45, 51, 55, 57–71, 73–75, 77–83
NoC Réseau sur puce (*Network on Chip*) 13, 15, 17, 25, 28, 38–40, 42, 44–48, 65, 81
NUMA Architecture mémoire non-uniforme (*Non-Uniform Memory Architecture*) 15, 19, 20, 40, 55, 62, 70, 72, 75
PAAS Plate-forme en tant que service (*Platform as a Service*) 76
PCAM Modèle de déploiement d'applications parallèles de Ian Foster proposé en 1995 (*Partition Communicate Agglomerate Map*) 67–69
PCIe Bus local rapide d'interconnexion de cartes d'extension (*Peripheral Component Interconnect Express*) 37, 76
PE Unité de calcul, cœur (*Processing Element*) 15, 17, 20, 28, 30, 38, 44, 45, 67, 77, 79, 80
PGAS Espace d'addressage global partitionné (*Patitioned Global Address Space*) 55
PS Modèle de programmation évènementiel publier-s'abonner (*Publish-Subscribe*) 61, 62
REST Architecture logicielle pour les services web (*Representational State Transfer*) 74
RL Apprentissage par renforcement (*Reinforcement Learning*) 75
RO Recherche Opérationnelle 49, 70, 84
SDF *Static Data-Flow* 22, 23, 25
SSI Système à image unique (*Single System Image*) 20
TCO Coût total de possession (*Total Cost of Ownership*) 19
TGCC Très Grand Centre de Calcul du CEA accueillant des super-calculateurs 67
TLB Traduction d'adresses virtuelles en adresses physiques (*Translation Lookaside Buffer*) 78
TLM Modèle transactionnel (*Transaction Level Modelling*) 46, 47
TPU Processeur de traitement des tenseurs, des structures de données vectorielles (*Tensor Processing Unit*) 53

Bibliographie

[Aba et al., 2020] Aba, M. A., Zaourar, L., and Munier, A. (2020). Efficient algorithm for scheduling parallel applications on hybrid multicore machines with communications delays and energy constraint. *Concurr. Comput. Pract. Exp.*, 32(15).

[Abdelouahab et al., 2018] Abdelouahab, K., Pelcat, M., Sérot, J., and Berry, F. (2018). Accelerating CNN inference on fpgas : A survey. *CoRR*, abs/1806.01683.

[Adapteva Inc., 2017] Adapteva Inc. (2017). A tflops scale 16nm 1024-core 64-bit risc-array processor. Hot Chips : A Symposium on High Performance Chips (HC29). https://www.hotchips.org/wp-content/uploads/hc_archives/hc29/HC29.23-Posters-Pub/HC29.23.p60-Adapteva.pdf.

[Akiba et al., 2019] Akiba, T., Sano, S., Yanase, T., Ohta, T., and Koyama, M. (2019). Optuna : A next-generation hyperparameter optimization framework. In Teredesai, A., Kumar, V., Li, Y., Rosales, R., Terzi, E., and Karypis, G., editors, *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2019, Anchorage, AK, USA, August 4-8, 2019*, pages 2623–2631. ACM.

[Almaless, 2014] Almaless, G. (2014). *Conception et réalisation d'un système d'exploitation pour des processeurs many-cores. (Operating System Design and Implementation for Single-Chip cc-NUMA Many-Core).* PhD thesis, Pierre and Marie Curie University, Paris, France.

[Amarasinghe et al., 2005] Amarasinghe, S. P., Gordon, M. I., Karczmarek, M., Lin, J., Maze, D., Rabbah, R. M., and Thies, W. (2005). Language and compiler design for streaming applications. *Int. J. Parallel Program.*, 33(2-3) :261–278.

[Amazon Web Service, 2016] Amazon Web Service (2016). Aws ec2 fpga development kit. GitHub Repository. https://github.com/aws/aws-fpga.

[AMD Inc., 2019] AMD Inc. (2019). Amd epyc 7742. AMD website. https://www.amd.com/fr/products/cpu/amd-epyc-7742.

[Andres et al., 2009] Andres, E., Widhalm, M., and Caloto, A. (2009). Achieving high speed cfd simulations : Optimization, parallelization, and fpga acceleration for the unstructured dlr tau code.

[Antoniu et al., 2005] Antoniu, G., Bougé, L., and Jan, M. (2005). Juxmem : An adaptive supportive platform for data sharing on the grid. *Scalable Comput. Pract. Exp.*, 6(3).

[Antoniu et al., 2007] Antoniu, G., Caron, E., Desprez, F., Fèvre, A., and Jan, M. (2007). Towards a transparent data access model for the gridrpcparadigm. In Aluru, S., Parashar, M., Badrinath, R., and Prasanna, V. K., editors, *High Performance Computing - HiPC 2007, 14th International Conference, Goa, India, December 18-21, 2007, Proceedings*, volume 4873 of *Lecture Notes in Computer Science*, pages 269–284. Springer.

[Arap and Swany, 2014] Arap, O. and Swany, M. (2014). Offloading MPI parallel prefix scan (mpi_scan) with the netfpga. *CoRR*, abs/1408.4939.

[ARM, 2017] ARM (2017). AMBA AXI and ACE Protocol Specification. ARM IHI 0022H (ID040120). https://developer.arm.com/-/media/Arm%20Developer%20Community/PDF/IHI0022H_amba_axi_protocol_spec.pdf.

[arm Limited, 2004] arm Limited (2004). ARM1176JZF-S technical reference manual. arm Developer. https://developer.arm.com/documentation/ddi0301/h/.

[Arnold et al., 2012] Arnold, S. S., Nuzzaci, R., and Gordon-Ross, A. (2012). Energy budgeting for cubesats with an integrated fpga. In *2012 IEEE Aerospace Conference*, pages 1–14.

[Astudillo-Salinas et al., 2016] Astudillo-Salinas, F., Barrera-Salamea, D., Rodas, A. V., and Solano-Quinde, L. D. (2016). Minimizing the power consumption in raspberry pi to use as a remote WSN gateway. In *8th IEEE Latin-American Conference on Communications, LATINCOM 2016, Medellin, Colombia, November 15-17, 2016*, pages 1–5. IEEE.

[Augonnet et al., 2012] Augonnet, C., Aumage, O., Furmento, N., Namyst, R., and Thibault, S. (2012). Starpu-mpi : Task programming over clusters of machines enhanced with accelerators. In Träff, J. L., Benkner, S., and Dongarra, J. J., editors, *Recent Advances in the Message Passing Interface - 19th European MPI Users' Group Meeting, EuroMPI 2012, Vienna, Austria, September 23-26, 2012. Proceedings*, volume 7490 of *Lecture Notes in Computer Science*, pages 298–299. Springer.

[Augonnet et al., 2011] Augonnet, C., Thibault, S., Namyst, R., and Wacrenier, P. (2011). Starpu : a unified platform for task scheduling on heterogeneous multicore architectures. *Concurr. Comput. Pract. Exp.*, 23(2) :187–198.

[Bach et al., 2010] Bach, M., Charney, M., Cohn, R., Demikhovsky, E., Devor, T., Hazelwood, K. M., Jaleel, A., Luk, C., Lyons, G., Patil, H., and Tal, A. (2010). Analyzing parallel programs with pin. *Computer*, 43(3) :34–41.

[Bader and JáJá, 1999] Bader, D. A. and JáJá, J. (1999). SIMPLE : A methodology for programming high performance algorithms on clusters of symmetric multiprocessors (smps). *J. Parallel Distributed Comput.*, 58(1) :92–108.

[Bailey et al., 1991] Bailey, D. H., Barszcz, E., Barton, J. T., Browning, D. S., Carter, R. L., Dagum, L., Fatoohi, R. A., Frederickson, P. O., Lasinski, T. A., Schreiber, R., Simon, H. D., Venkatakrishnan, V., and Weeratunga, S. (1991). The nas parallel benchmarks. *Int. J. High Perform. Comput. Appl.*, 5(3) :63–73.

[Banâtre et al., 2000] Banâtre, J., Fradet, P., and Métayer, D. L. (2000). Gamma and the chemical reaction model : Fifteen years after. In Calude, C., Paun, G., Rozenberg, G., and Salomaa, A., editors, *Multiset Processing, Mathematical, Computer Science, and Molecular Computing Points of View [Workshop on Multiset Processing, WMP 2000, Curtea de Arges, Romania, August 21-25, 2000]*, volume 2235 of *Lecture Notes in Computer Science*, pages 17–44. Springer.

[Banâtre et al., 2004] Banâtre, J., Fradet, P., and Radenac, Y. (2004). Higher-order chemical programming style. In Banâtre, J., Fradet, P., Giavitto, J., and Michel, O., editors, *Unconventional Programming Paradigms, International Workshop UPP 2004, Le Mont Saint Michel, France, September 15-17, 2004, Revised Selected and Invited Papers*, volume 3566 of *Lecture Notes in Computer Science*, pages 84–95. Springer.

[Banâtre and Métayer, 1990] Banâtre, J. and Métayer, D. L. (1990). The GAMMA model and its discipline of programming. *Sci. Comput. Program.*, 15(1) :55–77.

[Barlas, 2015] Barlas, G. (2015). Chapter 1 - introduction. In Barlas, G., editor, *Multicore and GPU Programming*, pages 1 – 26. Morgan Kaufmann, Boston.

[Bartenstein and Liu, 2013] Bartenstein, T. and Liu, Y. D. (2013). Green streams for data-intensive software. In Notkin, D., Cheng, B. H. C., and Pohl, K., editors, *35th International Conference on Software Engineering, ICSE '13, San Francisco, CA, USA, May 18-26, 2013*, pages 532–541. IEEE Computer Society.

[Bell et al., 2008] Bell, S., Edwards, B., Amann, J., Conlin, R., Joyce, K., Leung, V., MacKay, J., Reif, M., Bao, L., III, J. F. B., Mattina, M., Miao, C., Ramey, C., Wentzlaff, D., Anderson, W., Berger, E., Fairbanks, N., Khan, D., Montenegro, F., Stickney, J., and Zook, J. (2008). TILE64 - processor : A 64-core soc with mesh interconnect. In *2008 IEEE International Solid-State Circuits Conference, ISSCC 2008, Digest of Technical Papers, San Francisco, CA, USA, February 3-7, 2008*, pages 88–89. IEEE.

[Benabderrahmane et al., 2010] Benabderrahmane, M., Pouchet, L., Cohen, A., and Bastoul, C. (2010). The polyhedral model is more widely applicable than you think. In Gupta, R., editor, *Compiler Construction, 19th International Conference, CC 2010, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2010, Paphos, Cyprus, March 20-28, 2010. Proceedings*, volume 6011 of *Lecture Notes in Computer Science*, pages 283–303. Springer.

[Benazouz et al., 2013] Benazouz, M., Kordon, A. M., Hujsa, T., and Bodin, B. (2013). Liveness evaluation of a cyclo-static dataflow graph. In *The 50th Annual Design Automation Conference 2013, DAC '13, Austin, TX, USA, May 29 - June 07, 2013*, pages 3 :1–3 :7. ACM.

[Beri et al., 2017] Beri, T., Bansal, S., and Kumar, S. (2017). The unicorn runtime : Efficient distributed shared memory programming for hybrid CPU-GPU clusters. *IEEE Trans. Parallel Distributed Syst.*, 28(5) :1518–1534.

[Bershad et al., 1993] Bershad, B. N., Zekauskas, M. J., and Sawdon, W. A. (1993). The Midway distributed shared memory system. In *Proceedings of the 38th IEEE International Computer Conference (COMPCON Spring '93)*, pages 528–537, Los Alamitos, CA.

[Bielert et al., 2015] Bielert, M., Ciorba, F. M., Feldhoff, K., Ilsche, T., and Nagel, W. E. (2015). HAEC-SIM : a simulation framework for highly adaptive energy-efficient computing platforms. In Theodoropoulos, G., editor, *Proceedings of the 8th International Conference on Simulation Tools and Techniques, Athens, Greece, August 24-26, 2015*, pages 129–138. ICST/ACM.

[Bilsen et al., 1996] Bilsen, G., Engels, M., Lauwereins, R., and Peperstraete, J. (1996). Cycle-static dataflow. *IEEE Transactions on Signal Processing*, 44(2) :397–408.

[Bohnenstiehl et al., 2017] Bohnenstiehl, B., Stillmaker, A., Pimentel, J. J., Andreas, T., Liu, B., Tran, A., Adeagbo, E., and Baas, B. M. (2017). Kilocore : A 32-nm 1000-processor computational array. *IEEE J. Solid State Circuits*, 52(4) :891–902.

[Bolze et al., 2006] Bolze, R., Cappello, F., Caron, E., Daydé, M. J., Desprez, F., Jeannot, E., Jégou, Y., Lanteri, S., Leduc, J., Melab, N., Mornet, G., Namyst, R., Primet, P., Quétier, B., Richard, O., Talbi, E., and Touche, I. (2006). Grid'5000 : A large scale and highly reconfigurable experimental grid testbed. *Int. J. High Perform. Comput. Appl.*, 20(4) :481–494.

[Boudguiga et al., 2021] Boudguiga, A., Stan, O., Fazzat, A., Labiod, H., and Clet, P. (2021). Privacy preserving services for intelligent transportation systems with homomorphic encryption. In Mori, P., Lenzini, G., and Furnell, S., editors, *Proceedings of the 7th International Conference on Information Systems Security and Privacy, ICISSP 2021, Online Streaming, February 11-13, 2021*, pages 684–693. SCITEPRESS.

[Buck et al., 2004] Buck, I., Foley, T., Horn, D. R., Sugerman, J., Fatahalian, K., Houston, M., and Hanrahan, P. (2004). Brook for gpus : stream computing on graphics hardware. *ACM Trans. Graph.*, 23(3) :777–786.

[Carter et al., 1991] Carter, J. B., Bennett, J. K., and Zwaenepoel, W. (1991). Implementation and performance of munin. In Levy, H. M., editor, *Proceedings of the Thirteenth ACM Symposium on Operating System Principles, SOSP 1991, Asilomar Conference Center, Pacific Grove, California, USA, October 13-16, 1991*, pages 152–164. ACM.

[CEA, 2017] CEA (2017). N2d2, première plateforme européenne de deep learning en open source. CEA Espace Presse. https://www.cea.fr/presse/Pages/actualites-communiques/ntic/plateforme-n2d2-deep-learning.aspx.

[Cenedese et al., 2017] Cenedese, A., Tramarin, F., and Vitturi, S. (2017). An energy efficient ethernet strategy based on traffic prediction and shaping. *IEEE Trans. Commun.*, 65(1) :270–282.

[Cho et al., 2003] Cho, J., Oh, S., Kim, J., Lee, H. H., and Lee, J. (2003). Neighbor caching in multi-hop wireless ad hoc networks. *IEEE Commun. Lett.*, 7(11) :525–527.

[Christodoulis et al., 2018] Christodoulis, G., Broquedis, F., Muller, O., Selva, M., and Desprez, F. (2018). An FPGA target for the starpu heterogeneous runtime system. In Niar, S. and Saghir, M. A. R., editors, *13th International Symposium on Reconfigurable Communication-centric Systems-on-Chip, ReCoSoC 2018, Lille, France, July 9-11, 2018*, pages 1–8. IEEE.

[Creveuil, 1991] Creveuil, C. (1991). *Techniques d'analyse et de mise en oeuvre des programmes gamma.* PhD thesis. Thèse de doctorat dirigée par Le Métayer, Daniel Informatique Rennes 1 1991.

[Culler et al., 1999] Culler, D. E., Singh, J. P., and Gupta, A. (1999). *Parallel computer architecture - a hardware / software approach.* Morgan Kaufmann.

[Cutress and Shilov, 2019] Cutress, I. and Shilov, A. (2019). The larrabee chapter closes : Intel's final xeon phi processors now in eol. AnandTech. https://www.anandtech.com/show/14305/intel-xeon-phi-knights-mill-now-eol.

[Czajkowski et al., 2012] Czajkowski, T. S., Aydonat, U., Denisenko, D., Freeman, J., Kinsner, M., Neto, D., Wong, J., Yiannacouras, P., and Singh, D. P. (2012). From opencl to high-performance hardware on FPGAS. In Koch, D., Singh, S., and Tørresen, J., editors, *22nd International Conference on Field Programmable Logic and Applications (FPL), Oslo, Norway, August 29-31, 2012*, pages 531–534. IEEE.

[Dahmani, 2015] Dahmani, S. (2015). *Modèles et protocoles de cohérence de données, décision et optimisation à la compilation pour des architectures massivement parallèles. (Data Consistency Models and Protocols, Decision and Optimization at Compile Time for Massively Parallel Architectures).* PhD thesis, University of Southern Brittany, Morbihan, France.

[Davis and Hu, 2011] Davis, T. A. and Hu, Y. (2011). The university of florida sparse matrix collection. *ACM Trans. Math. Softw.*, 38(1) :1 :1–1 :25.

[de Dinechin, 2015] de Dinechin, B. D. (2015). Kalray mppa® : Massively parallel processor array : Revisiting DSP acceleration with the kalray MPPA manycore processor. In *2015 IEEE Hot Chips 27 Symposium (HCS), Cupertino, CA, USA, August 22-25, 2015*, pages 1–27. IEEE.

[de Dinechin et al., 2013a] de Dinechin, B. D., Ayrignac, R., Beaucamps, P., Couvert, P., Ganne, B., de Massas, P. G., Jacquet, F., Jones, S., Chaisemartin, N. M., Riss, F., and Strudel, T. (2013a). A clustered manycore processor architecture for embedded and accelerated applications. In *IEEE High Performance Extreme Computing Conference, HPEC 2013, Waltham, MA, USA, September 10-12, 2013*, pages 1–6. IEEE.

[de Oliveira Castro et al., 2010] de Oliveira Castro, P., Louise, S., and Barthou, D. (2010). A multidimensional array slicing DSL for stream programming. In Barolli, L., Xhafa, F., Vitabile, S., and Hsu, H., editors, *CISIS 2010, The Fourth International Conference on Complex, Intelligent and Software Intensive Systems, Krakow, Poland, 15-18 February 2010*, pages 913–918. IEEE Computer Society.

[Dennard et al., 1974] Dennard, R. H., Gaensslen, F. H., Yu, H., Rideout, V. L., Bassous, E., and LeBlanc, A. R. (1974). Design of ion-implanted mosfet's with very small physical dimensions. *IEEE Journal of Solid-State Circuits*, 9(5) :256–268.

[Denolf et al., 2007] Denolf, K., Bekooij, M. J. G., Cockx, J., Verkest, D., and Corporaal, H. (2007). Exploiting the expressiveness of cyclo-static dataflow to model multimedia implementations. *EURASIP J. Adv. Signal Process.*, 2007.

[Design Spot, 2018] Design Spot (2018). FACE : Démonstrateur de l'architecture électronique du véhicule de demain. Design Spot, Université Paris-Saclay. https://www.designspot.fr/portfolio/face/.

[Dimakopoulos and Hadjidoukas, 2011] Dimakopoulos, V. V. and Hadjidoukas, P. E. (2011). HOMPI : A hybrid programming framework for expressing and deploying task-based parallelism. In Jeannot, E., Namyst, R., and Roman, J., editors, *Euro-Par 2011 Parallel Processing - 17th International Conference, Euro-Par 2011, Bordeaux, France, August 29 - September 2, 2011, Proceedings, Part II*, volume 6853 of *Lecture Notes in Computer Science*, pages 14–26. Springer.

[Djidi et al., 2021] Djidi, N. E. H., Gautier, M., Courtay, A., and Berder, O. (2021). Mees-wur : Minimum energy coding with early shutdown for wake-up receivers. *IEEE Trans. Green Commun. Netw.*, 5(3) :1502–1513.

[Domke et al., 2019] Domke, J., Matsumura, K., Wahib, M., Zhang, H., Yashima, K., Tsuchikawa, T., Tsuji, Y., Podobas, A., and Matsuoka, S. (2019). Double-precision fpus in high-performance computing : An embarrassment of riches ? In *2019 IEEE International Parallel and Distributed Processing Symposium, IPDPS 2019, Rio de Janeiro, Brazil, May 20-24, 2019*, pages 78–88. IEEE.

[Dongarra, 2016] Dongarra, J. (2016). Report on the sunway taihulight system. University of Tennessee, Oak Ridge National Laboratory, Tech Report UT-EECS-16-742. http://www.netlib.org/utk/people/JackDongarra/PAPERS/sunway-report-2016.pdf.

[Dongarra, 1987] Dongarra, J. J. (1987). The LINPACK benchmark : An explanation. In Houstis, E. N., Papatheodorou, T. S., and Polychronopoulos, C. D., editors, *Supercomputing, 1st International Conference, Athens, Greece, June 8-12, 1987, Proceedings*, volume 297 of *Lecture Notes in Computer Science*, pages 456–474. Springer.

[Dubrulle et al., 2012] Dubrulle, P., Louise, S., Sirdey, R., and David, V. (2012). A low-overhead dedicated execution support for stream applications on shared-memory cmp. In Jerraya, A., Carloni, L. P., Maraninchi, F., and Regehr, J., editors, *Proceedings of the 12th International Conference on Embedded Software, EMSOFT 2012, part of the Eighth Embedded Systems Week, ESWeek 2012, Tampere, Finland, October 7-12, 2012*, pages 143–152. ACM.

[Dybdahl and Stenström, 2007] Dybdahl, H. and Stenström, P. (2007). An adaptive shared/private NUCA cache partitioning scheme for chip multiprocessors. In *13st International Conference on High-Performance Computer Architecture (HPCA-13 2007), 10-14 February 2007, Phoenix, Arizona, USA*, pages 2–12. IEEE Computer Society.

[Eastep et al., 2017] Eastep, J., Sylvester, S., Cantalupo, C., Geltz, B., Ardanaz, F., Al-Rawi, A., Livingston, K., Keceli, F., Maiterth, M., and Jana, S. (2017). Global extensible open power manager : A vehicle for HPC community collaboration on co-designed energy management solutions. In Kunkel, J. M., Yokota, R., Balaji, P., and Keyes, D. E., editors, *High Performance Computing - 32nd International Conference, ISC High Performance 2017, Frankfurt, Germany, June 18-22, 2017, Proceedings*, volume 10266 of *Lecture Notes in Computer Science*, pages 394–412. Springer.

[Eker and Janneck, 2003] Eker, J. and Janneck, J. W. (2003). Cal language report specification of the cal actor language. Technical Report UCB/ERL M03/48, EECS Department, University of California, Berkeley. http://www.eecs.berkeley.edu/Pubs/TechRpts/2003/4186.html.

[Eskandari et al., 2006] Eskandari, H., Geiger, C. D., and Lamont, G. B. (2006). Fastpga : A dynamic population sizing approach for solving expensive multiobjective optimization problems. In Obayashi, S., Deb, K., Poloni, C., Hiroyasu, T., and Murata, T., editors, *Evolutionary Multi-Criterion Optimization, 4th International Conference, EMO 2007, Matsushima, Japan, March 5-8, 2007, Proceedings*, volume 4403 of *Lecture Notes in Computer Science*, pages 141–155. Springer.

[Filgueras et al., 2014] Filgueras, A., Gil, E., Jiménez-González, D., Álvarez, C., Martorell, X., Langer, J., Noguera, J., and Vissers, K. A. (2014). Ompss@zynq all-programmable soc ecosystem. In Betz, V. and Constantinides, G. A., editors, *The 2014 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, FPGA '14, Monterey, CA, USA - February 26 - 28, 2014*, pages 137–146. ACM.

[Foster, 1995] Foster, I. T. (1995). *Designing and building parallel programs - concepts and tools for parallel software engineering*. Addison-Wesley.

[Foster and Kesselman, 2004] Foster, I. T. and Kesselman, C., editors (2004). *The Grid 2 - Blueprint for a New Computing Infrastructure, Second Edition*. The Morgan Kaufmann Series in Computer Architecture and Design. Morgan Kaufmann.

[Friese et al., 2013] Friese, R., Khemka, B., Maciejewski, A. A., Siegel, H. J., Koenig, G. A., Powers, S., Hilton, M., Rambharos, J., Okonski, G., and Poole, S. W. (2013). An analysis framework for investigating the trade-offs between system performance and energy consumption in a heterogeneous computing environment. In *2013 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum, Cambridge, MA, USA, May 20-24, 2013*, pages 19–30. IEEE.

[Fujitsu Limited, 2020] Fujitsu Limited (2020). A64FX microarchitecture manual. Technical Report Edition 1.3, Fujitsu Limited. https://www.stonybrook.edu/commcms/ookami/support/_docs/A64FX_Microarchitecture_Manual_en_1.3.pdf.

[Gabriel et al., 2004] Gabriel, E., Fagg, G. E., Bosilca, G., Angskun, T., Dongarra, J. J., Squyres, J. M., Sahay, V., Kambadur, P., Barrett, B., Lumsdaine, A., Castain, R. H., Daniel, D. J., Graham, R. L., and Woodall, T. S. (2004). Open MPI : goals, concept, and design of a next generation MPI implementation. In Kranzlmüller, D., Kacsuk, P., and Dongarra, J. J., editors, *Recent Advances in Parallel Virtual Machine and Message Passing Interface, 11th European PVM/MPI Users' Group Meeting, Budapest, Hungary, September 19-22, 2004, Proceedings*, volume 3241 of *Lecture Notes in Computer Science*, pages 97–104. Springer.

[Galea and Sirdey, 2012] Galea, F. and Sirdey, R. (2012). A parallel simulated annealing approach for the mapping of large process networks. In *26th IEEE International Parallel and Distributed Processing Symposium Workshops & PhD Forum, IPDPS 2012, Shanghai, China, May 21-25, 2012*, pages 1787–1792. IEEE Computer Society.

[Gao et al., 2009] Gao, S., Schmidt, A. G., and Sass, R. (2009). Hardware implementation of mpi_barrier on an FPGA cluster. In Danek, M., Kadlec, J., and Nelson, B. E., editors, *19th International Conference on Field Programmable Logic and Applications, FPL 2009, August 31 - September 2, 2009, Prague, Czech Republic*, pages 12–17. IEEE.

[Gelado et al., 2010] Gelado, I., Cabezas, J., Navarro, N., Stone, J. E., Patel, S. J., and Hwu, W. W. (2010). An asymmetric distributed shared memory model for heterogeneous parallel systems. In Hoe, J. C. and Adve, V. S., editors, *Proceedings of the 15th International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2010, Pittsburgh, Pennsylvania, USA, March 13-17, 2010*, pages 347–358. ACM.

[Gordon et al., 2006] Gordon, M. I., Thies, W., and Amarasinghe, S. P. (2006). Exploiting coarse-grained task, data, and pipeline parallelism in stream programs. In Shen, J. P. and Martonosi, M., editors, *Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2006, San Jose, CA, USA, October 21-25, 2006*, pages 151–162. ACM.

[Gordon et al., 2002] Gordon, M. I., Thies, W., Karczmarek, M., Lin, J., Meli, A. S., Lamb, A. A., Leger, C., Wong, J., Hoffmann, H., Maze, D., and Amarasinghe, S. P. (2002). A stream compiler for communication-exposed architectures. In Gharachorloo, K. and Wood, D. A., editors, *Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-X), San Jose, California, USA, October 5-9, 2002*, pages 291–303. ACM Press.

[Goubier et al., 2014] Goubier, T., Couroussé, D., and Azaiez, S. (2014). τ C : C with process network extensions for embedded manycores. In Abramson, D., Lees, M., Krzhizhanovskaya, V. V., Dongarra, J. J., and Sloot, P. M. A., editors, *Proceedings of the International Conference on Computational Science, ICCS 2014, Cairns, Queensland, Australia, 10-12 June, 2014*, volume 29 of *Procedia Computer Science*, pages 1100–1112. Elsevier.

[Goubier et al., 2020a] Goubier, T., Martinovic, J., Dubrulle, P., Ganne, L., Louise, S., Martinovic, T., and Slaninová, K. (2020a). Real-time model of computation over hpc/cloud orchestration - the LEXIS approach. In Barolli, L., Poniszewska-Maranda, A., and Enokido, T., editors, *Complex, Intelligent and Software Intensive Systems - Proceedings of the 14th International Conference on Complex, Intelligent and Software Intensive Systems (CISIS-2020), Lodz, Poland, 1-3 July 2020*, volume 1194 of *Advances in Intelligent Systems and Computing*, pages 255–266. Springer.

[Goubier et al., 2020b] Goubier, T., Rakowsky, N., and Harig, S. (2020b). Fast tsunami simulations for a real-time emergency response flow. In *IEEE/ACM HPC for Urgent Decision Making, UrgentHPC@SC 2020, Atlanta, GA, USA, November 13, 2020*, pages 21–26. IEEE.

[Goubier et al., 2011] Goubier, T., Sirdey, R., Louise, S., and David, V. (2011). Σc : A programming model and language for embedded manycores. In Xiang, Y., Cuzzocrea, A., Hobbs, M., and Zhou, W., editors, *Algorithms and Architectures for Parallel Processing - 11th International Conference, ICA3PP, Melbourne, Australia, October 24-26, 2011, Proceedings, Part I*, volume 7016 of *Lecture Notes in Computer Science*, pages 385–394. Springer.

[Griessl et al., 2014] Griessl, R., Peykanu, M., Hagemeyer, J., Porrmann, M., Krupop, S., vor dem Berge, M., Kiesel, T., and Christmann, W. (2014). A scalable server architecture for next-generation heterogeneous compute clusters. In *12th IEEE International Conference on Embedded and Ubiquitous Computing, EUC 2014, Milano, Italy, August 26-28, 2014*, pages 146–153. IEEE Computer Society.

[Hascoet et al., 2017] Hascoet, J., Desnos, K., Nezan, J., and de Dinechin, B. D. (2017). Hierarchical dataflow model for efficient programming of clustered manycore processors. In *28th IEEE International Conference on Application-specific Systems, Architectures and Processors, ASAP 2017, Seattle, WA, USA, July 10-12, 2017*, pages 137–142. IEEE Computer Society.

[Hashemi et al., 2018] Hashemi, M., Swersky, K., Smith, J. A., Ayers, G., Litz, H., Chang, J., Kozyrakis, C., and Ranganathan, P. (2018). Learning memory access patterns. In Dy, J. G. and Krause, A., editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 1924–1933. PMLR.

[Heinrich et al., 2017] Heinrich, F. C., Cornebize, T., Degomme, A., Legrand, A., Carpen-Amarie, A., Hunold, S., Orgerie, A., and Quinson, M. (2017). Predicting the energy-consumption of MPI applications at scale using only a single node. In *2017 IEEE International Conference on Cluster Computing, CLUSTER 2017, Honolulu, HI, USA, September 5-8, 2017*, pages 92–102. IEEE Computer Society.

[Hennessy and Patterson, 2012] Hennessy, J. L. and Patterson, D. A. (2012). *Computer Architecture - A Quantitative Approach, 5th Edition*. Morgan Kaufmann.

[Herrero et al., 2010] Herrero, E., González, J., and Canal, R. (2010). Elastic cooperative caching : an autonomous dynamically adaptive memory hierarchy for chip multiprocessors. In Seznec, A., Weiser, U. C., and Ronen, R., editors, *37th International Symposium on Computer Architecture (ISCA 2010), June 19-23, 2010, Saint-Malo, France*, pages 419–428. ACM.

[Hilbert, 1935] Hilbert, D. (1935). Über die stetige abbildung einer linie auf ein flächenstück. In *Dritter Band : Analysis · Grundlagen der Mathematik · Physik Verschiedenes : Nebst Einer Lebensgeschichte*, pages 1–2, Berlin, Heidelberg. Springer Berlin Heidelberg.

[Holmedahl et al., 1998] Holmedahl, V., Smith, B., and Yang, T. (1998). Cooperative caching of dynamic content on a distributed web server. In *Proceedings of the Seventh IEEE International Symposium on High Performance Distributed Computing, HPDC '98, Chicago, Illinois, USA, July 28-31, 1998*, pages 243–250. IEEE Computer Society.

[Howard et al., 2011] Howard, J., Dighe, S., Vangal, S. R., Ruhl, G., Borkar, N., Jain, S., Erraguntla, V., Konow, M., Riepen, M., Gries, M., Droege, G., Lund-Larsen, T., Steibl, S., Borkar, S., De, V. K., and der Wijngaart, R. F. V. (2011). A 48-core IA-32 processor in 45 nm CMOS using on-die message-passing and DVFS for performance and power scaling. *IEEE J. Solid State Circuits*, 46(1) :173–183.

[HPE, 2014] HPE (2014). Pioneering enterprise-class 64-bit arm server technology : Hp moonshot system on 64-bit arm. Hewlett-Packard Development Company Solution Brief. http://www.hp.com/hpinfo/newsroom/press_kits/2014/ARMmomentum/HP_ARM_SolutionBrief.pdf.

[Huang et al., 2016] Huang, M., Wu, D., Yu, C. H., Fang, Z., Interlandi, M., Condie, T., and Cong, J. (2016). Programming and runtime support to blaze FPGA accelerator deployment at datacenter scale. In Aguilera, M. K., Cooper, B., and Diao, Y., editors, *Proceedings of the Seventh ACM Symposium on Cloud Computing, Santa Clara, CA, USA, October 5-7, 2016*, pages 456–469. ACM.

[Intel, 2009] Intel (2009). An introduction to the Intel QuickPath Interconnect. Document Number 320412-001US. https://www.intel.fr/content/dam/doc/white-paper/quick-path-interconnect-introduction-paper.pdf.

[Irofti, 2020] Irofti, D. (2020). An anticipatory protocol to reach fast consensus in multi-agent systems. *Autom.*, 113 :108776.

[ITRS, 2011] ITRS (2011). System driver chapter 2010 updates. International Technology Roadmap for Semiconductors, International Technology Working Group. http://www.itrs.net.

[ITRS2.0, 2015] ITRS2.0 (2015). Executive report. International Technology Roadmap for Semiconductors 2.0, International Technology Working Group. https://www.semiconductors.org/resources/2015-international-technology-roadmap-for-semiconductors-itrs/.

[Kahn, 1974] Kahn, G. (1974). The semantics of a simple language for parallel programming. In Rosenfeld, J. L., editor, *Information Processing, Proceedings of the 6th IFIP Congress 1974, Stockholm, Sweden, August 5-10, 1974*, pages 471–475. North-Holland.

[Karp et al., 1967] Karp, R. M., Miller, R. E., and Winograd, S. (1967). The organization of computations for uniform recurrence equations. *J. ACM*, 14(3) :563–590.

[Kathail, 2020] Kathail, V. (2020). Xilinx vitis unified software platform. In Neuendorffer, S. and Shannon, L., editors, *FPGA '20 : The 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Seaside, CA, USA, February 23-25, 2020*, pages 173–174. ACM.

[Kaxiras et al., 2015] Kaxiras, S., Klaftenegger, D., Norgren, M., Ros, A., and Sagonas, K. (2015). Turning centralized coherence and distributed critical-section execution on their head : A new approach for scalable distributed shared memory. In Kielmann, T., Hildebrand, D., and Taufer, M., editors, *Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing, HPDC 2015, Portland, OR, USA, June 15-19, 2015*, pages 3–14. ACM.

[Keleher et al., 1994] Keleher, P. J., Cox, A. L., Dwarkadas, S., and Zwaenepoel, W. (1994). Treadmarks : Distributed shared memory on standard workstations and operating systems. In *USENIX Winter 1994 Technical Conference, San Francisco, California, USA, January 17-21, 1994, Conference Proceedings*, pages 115–132. USENIX Association.

[Keltcher et al., 2002] Keltcher, C., Ahmed, A., Clark, M., Gao, H., Goveas, K., Haddad, R., Holloway, B., Hughes, B., Klass, R., Kroesche, D., Madrid, P., McGrath, K., Tan, T., White, S., Wood, T., and Zuraski, G. (2002). The amd hammer processor core. HotChips 14. https://webpages.eng.wayne.edu/~ad5781/ECECourses/ECE5620/Manuals/Hammer2_AMD.pdf.

[Kirchner et al., 2015] Kirchner, F., Kosmatov, N., Prevosto, V., Signoles, J., and Yakobowski, B. (2015). Frama-c : A software analysis perspective. *Formal Aspects Comput.*, 27(3) :573–609.

[Kubiatowicz et al., 2000] Kubiatowicz, J., Bindel, D., Chen, Y., Czerwinski, S. E., Eaton, P. R., Geels, D., Gummadi, R., Rhea, S. C., Weatherspoon, H., Weimer, W., Wells, C., and Zhao, B. Y. (2000). Oceanstore : An architecture for global-scale persistent storage. In Rudolph, L. and Gupta, A., editors, *ASPLOS-IX Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems, Cambridge, MA, USA, November 12-15, 2000*, pages 190–201. ACM Press.

[Kuon and Rose, 2007] Kuon, I. and Rose, J. (2007). Measuring the gap between fpgas and asics. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, 26(2) :203–215.

[Lamport, 1978] Lamport, L. (1978). Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21(7) :558–565.

[Lamport, 1979] Lamport, L. (1979). How to make a multiprocessor computer that correctly executes multiprocess programs. *IEEE Trans. Computers*, 28(9) :690–691.

[Lamport, 1987] Lamport, L. (1987). Distribution. Email message sent to a DEC SRC bulletin board at 12 :23 :29 PDT on 28 May 87.

[Lee and Parks, 1995] Lee, E. A. and Parks, T. M. (1995). Dataflow process networks. *Proceedings of the IEEE*, 83(5) :773–801.

[Legrand et al., 2004] Legrand, A., Renard, H., Robert, Y., and Vivien, F. (2004). Mapping and load-balancing iterative computations. *IEEE Trans. Parallel Distributed Syst.*, 15(6) :546–558.

[Lenormand, 2022] Lenormand, E. (2022). *Unification des mémoires réparties dans les systèmes hétérogènes.* PhD thesis. Thèse de doctorat dirigée par Charles, Henri-Pierre Informatique Université Paris-Saclay 2022.

[Lepak et al., 2017] Lepak, K., Talbot, G., White, S., Beck, N., and Naffziger, S. (2017). The next generation AMD enterprise server product architecture. HotChips 29. https://old.hotchips.org/wp-content/uploads/hc_archives/hc29/HC29.22-Tuesday-Pub/HC29.22.90-Server-Pub/HC29.22.921-EPYC-Lepak-AMD-v2.pdf.

[Letertre, 2021] Letertre, L. (2021). Causalité quantique : et si l'ordre des événements disparaissait à très petite échelle ? The Conversation. https://theconversation.com/causalite-quantique-et-si-lordre-des-evenements-disparaissait-a-tres-petite-echelle-161948.

[Li, 1988] Li, K. (1988). IVY : A shared virtual memory system for parallel computing. In *Proceedings of the International Conference on Parallel Processing, ICPP '88, The Pennsylvania State University, University Park, PA, USA, August 1988. Volume 2 : Software*, pages 94–101. Pennsylvania State University Press.

[Lim et al., 2006] Lim, M. Y., Freeh, V. W., and Lowenthal, D. K. (2006). MPI and communication - adaptive, transparent frequency and voltage scaling of communication phases in MPI programs. In *Proceedings of the ACM/IEEE SC2006 Conference on High Performance Networking and Computing, November 11-17, 2006, Tampa, FL, USA*, page 107. ACM Press.

[Lin et al., 2012] Lin, F. X., Wang, Z., LiKamWa, R., and Zhong, L. (2012). Reflex : using low-power processors in smartphones without knowing them. In Harris, T. and Scott, M. L., editors, *Proceedings of the 17th International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2012, London, UK, March 3-7, 2012*, pages 13–24. ACM.

[LIP6, 2009] LIP6 (2009). Tsar (tera-scale architecture). LIP6 website. https://www-soc.lip6.fr/trac/tsar.

[Litz et al., 2009] Litz, H., Fröning, H., and Brüning, U. (2009). A hypertransport 3 physical layer interface for fpgas. In Becker, J., Woods, R. F., Athanas, P. M., and Morgan, F., editors, *Reconfigurable Computing : Architectures, Tools and Applications, 5th International Workshop, ARC 2009, Karlsruhe, Germany, March 16-18, 2009. Proceedings*, volume 5453 of *Lecture Notes in Computer Science*, pages 4–14. Springer.

[Louise, 2017] Louise, S. (2017). An openmp backend for the Σc streaming language. In Koumoutsakos, P., Lees, M., Krzhizhanovskaya, V. V., Dongarra, J. J., and Sloot, P. M. A., editors, *International Conference on Computational Science, ICCS 2017, 12-14 June 2017, Zurich, Switzerland*, volume 108 of *Procedia Computer Science*, pages 1073–1082. Elsevier.

[Louise et al., 2014] Louise, S., Dubrulle, P., and Goubier, T. (2014). A model of computation for real-time applications on embedded manycores. In *IEEE 8th International Symposium on Embedded Multicore/Manycore SoCs, MCSoC 2014, Aizu-Wakamatsu, Japan, September 23-25, 2014*, pages 333–340. IEEE Computer Society.

[Lu et al., 1995] Lu, H., Dwarkadas, S., Cox, A. L., and Zwaenepoel, W. (1995). Message passing versus distributed shared memory on networks of workstations. In Karin, S., editor, *Proceedings Supercomputing '95, San Diego, CA, USA, December 4-8, 1995*, page 37. ACM.

[Marândola Kofuji, 2011] Marândola Kofuji, J. (2011). *Optimized method for cache coherence architecture based on multicore embedded systems.* PhD thesis. Thèse de doctorat dirigée par Zuffo, Marcelo Knörich, Escola Politécnica da Universidade de São Paulo.

[Mattern, 1988] Mattern, F. (1988). Virtual time and global states of distributed systems. In *PARALLEL AND DISTRIBUTED ALGORITHMS*, pages 215–226. North-Holland.

[Matthew Mattina, 2013] Matthew Mattina (2013). Architecture and performance of the tilera tile-gx8072 manycore processor. InsideHPC Report. https://www.hotchips.org/wp-content/uploads/hc_archives/hc29/HC29.23-Posters-Pub/HC29.23.p60-Adapteva.pdf.

[Mazzucco et al., 2009] Mazzucco, M., Morgan, G., Panzieri, F., and Sharp, C. (2009). Engineering distributed shared memory middleware for java. In Meersman, R., Dillon, T. S., and Herrero, P., editors, *On the Move to Meaningful Internet Systems : OTM 2009, Confederated International Conferences, CoopIS, DOA, IS, and ODBASE 2009, Vilamoura, Portugal, November 1-6, 2009, Proceedings, Part I*, volume 5870 of *Lecture Notes in Computer Science*, pages 531–548. Springer.

[Michalska et al., 2015] Michalska, M., Boutellier, J., and Mattavelli, M. (2015). A methodology for profiling and partitioning stream programs on many-core architectures. In Koziel, S., Leifsson, L. Þ., Lees, M., Krzhizhanovskaya, V. V., Dongarra, J. J., and Sloot, P. M. A., editors, *Proceedings of the International Conference on Computational Science, ICCS 2015, Computational Science at the Gates of Nature, Reykjavík, Iceland, 1-3 June, 2015, 2014*, volume 51 of *Procedia Computer Science*, pages 2962–2966. Elsevier.

[Mnih et al., 2013] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. A. (2013). Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602.

[Moritz et al., 2018] Moritz, P., Nishihara, R., Wang, S., Tumanov, A., Liaw, R., Liang, E., Elibol, M., Yang, Z., Paul, W., Jordan, M. I., and Stoica, I. (2018). Ray : A distributed framework for emerging AI applications. In Arpaci-Dusseau, A. C. and Voelker, G., editors, *13th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2018, Carlsbad, CA, USA, October 8-10, 2018*, pages 561–577. USENIX Association.

[Mulnix et al., 2017] Mulnix, D., Kumar, A., and Ould-ahmed-vall, E. (2017). Intel xeon processor scalable family technical overview. https://software.intel.com/content/www/us/en/develop/articles/intel-xeon-processor-scalable-family-technical-overview.html.

[Mutnury et al., 2010] Mutnury, B., Paglia, F., Mobley, J., Singh, G. K., and Bellomio, R. (2010). Quickpath interconnect (qpi) design and analysis in high speed servers. In *19th Topical Meeting on Electrical Performance of Electronic Packaging and Systems*, pages 265–268.

[Nelson et al., 2015] Nelson, J., Holt, B., Myers, B., Briggs, P., Ceze, L., Kahan, S., and Oskin, M. (2015). Latency-tolerant software distributed shared memory. In Lu, S. and Riedel, E., editors, *2015 USENIX Annual Technical Conference, USENIX ATC '15, July 8-10, Santa Clara, CA, USA*, pages 291–305. USENIX Association.

[Németh et al., 2006] Németh, Z., Pérez, C., and Priol, T. (2006). Distributed workflow coordination : molecules and reactions. In *20th International Parallel and Distributed Processing Symposium (IPDPS 2006), Proceedings, 25-29 April 2006, Rhodes Island, Greece*. IEEE.

[Nicolae et al., 2011] Nicolae, B., Antoniu, G., Bougé, L., Moise, D., and Carpen-Amarie, A. (2011). Blobseer : Next-generation data management for large scale infrastructures. *J. Parallel Distributed Comput.*, 71(2) :169–184.

[Nitzberg and Lo, 1991] Nitzberg, B. and Lo, V. M. (1991). Distributed shared memory : A survey of issues and algorithms. *Computer*, 24(8) :52–60.

[Okazaki et al., 2020] Okazaki, R., Tabata, T., Sakashita, S., Kitamura, K., Takagi, N., Sakata, H., Ishibashi, T., Nakamura, T., and Ajima, Y. (2020). Supercomputer Fugaku CPU A64FX Realizing High Performance, High-Density Packaging, and Low Power Consumption. Technical Report Fujistu Technical Review, Fujitsu Limited. https://www.fujistu.com/global/documents/about/resources/publications/technicalreview/2020-03/article03.pdf.

[Patterson, 2011] Patterson, B. (2011). Under the hood : The iphone's gaming mettle. TouchArcade. https://toucharcade.com/2008/07/07/under-the-hood-the-iphones-gaming-mettle/.

[Patterson and Hennessy, 2012] Patterson, D. A. and Hennessy, J. L. (2012). *Computer Organization and Design - The Hardware / Software Interface (Revised 4th Edition)*. The Morgan Kaufmann Series in Computer Architecture and Design. Academic Press.

[Penna et al., 2019] Penna, P. H., Francis, D., and Souto, J. (2019). The Hardware Abstraction Layer of Nanvix for the Kalray MPPA-256 Lightweight Manycore Processor. In *Conférence d'Informatique en Parallélisme, Architecture et Système*, Anglet, France.

[Pérache et al., 2008] Pérache, M., Jourdren, H., and Namyst, R. (2008). MPC : A unified parallel runtime for clusters of NUMA machines. In Luque, E., Margalef, T., and Benitez, D., editors, *Euro-Par 2008 - Parallel Processing, 14th International Euro-Par Conference, Las Palmas de Gran Canaria, Spain, August 26-29, 2008, Proceedings*, volume 5168 of *Lecture Notes in Computer Science*, pages 78–88. Springer.

[PEZY Computing, 2020] PEZY Computing (2020). PEZY computing. https://www.pezy.co.jp/en/.

[Preferred Networks, 2020] Preferred Networks (2020). Mn-core - prefered networks. Prefered Networks Website. https://projects.preferred.jp/mn-core/en/#mn-core.

[Putnam et al., 2015] Putnam, A., Caulfield, A. M., Chung, E. S., Chiou, D., Constantinides, K., Demme, J., Esmaeilzadeh, H., Fowers, J., Gopal, G. P., Gray, J., Haselman, M., Hauck, S., Heil, S., Hormati, A., Kim, J., Lanka, S., Larus, J. R., Peterson, E., Pope, S., Smith, A., Thong, J., Xiao, P. Y., and Burger, D. (2015). A reconfigurable fabric for accelerating large-scale datacenter services. *IEEE Micro*, 35(3) :10–22.

[Radenac, 2007] Radenac, Y. (2007). *Programmation "chimique" d'ordre supérieur*. PhD thesis. Thèse de doctorat dirigée par Banâtre, Jean-Pierre Informatique Rennes 1 2007.

[Rahman, 2013] Rahman, R. (2013). *Xeon Phi Cache and Memory Subsystem*, pages 65–80. Apress, Berkeley, CA.

[Raspberry PI Foundation, 2017] Raspberry PI Foundation (2017). BCM2835 - raspberry pi documentation. Raspberry Pi Documentation. https://www.raspberrypi.org/documentation/hardware/raspberrypi/bcm2835/README.md.

[Raynal, 1987] Raynal, M. (1987). A distributed algorithm to prevent mutual drift between n logical clocks. *Inf. Process. Lett.*, 24(3) :199–202.

[Raynal, 2013] Raynal, M. (2013). *Distributed Algorithms for Message-Passing Systems*. Springer.

[Richie and Ross, 2018] Richie, D. A. and Ross, J. A. (2018). Architecture emulation and simulation of future many-core epiphany RISC array processors. In Shi, Y., Fu, H., Tian, Y., Krzhizhanovskaya, V. V., Lees, M. H., Dongarra, J. J., and Sloot, P. M. A., editors, *Computational Science - ICCS 2018 - 18th International Conference, Wuxi, China, June 11-13, 2018, Proceedings, Part II*, volume 10861 of *Lecture Notes in Computer Science*, pages 289–300. Springer.

[Richie et al., 2017] Richie, D. A., Ross, J. A., and Infantolino, J. K. (2017). A distributed shared memory model and C++ templated meta-programming interface for the epiphany RISC array processor. In Koumoutsakos, P., Lees, M., Krzhizhanovskaya, V. V., Dongarra, J. J., and Sloot, P. M. A., editors, *International Conference on Computational Science, ICCS 2017, 12-14 June 2017, Zurich, Switzerland*, volume 108 of *Procedia Computer Science*, pages 1093–1102. Elsevier.

[Rinard et al., 1992] Rinard, M. C., Scales, D. J., and Lam, M. S. (1992). Heterogeneous parallel programming in jade. In Werner, R., editor, *Proceedings Supercomputing '92, Minneapolis, MN, USA, November 16-20, 1992*, pages 245–256. IEEE Computer Society.

[Rosendo et al., 2020] Rosendo, D., Silva, P., Simonin, M., Costan, A., and Antoniu, G. (2020). E2clab : Exploring the computing continuum through repeatable, replicable and reproducible edge-to-cloud experiments. In *IEEE International Conference on Cluster Computing, CLUSTER 2020, Kobe, Japan, September 14-17, 2020*, pages 176–186. IEEE.

[Roussel et al., 2017] Roussel, C., Keller, K., Gaalich, M., Bautista Gomez, L., and Bigot, J. (2017). PDI, an approach to decouple I/O concerns from high-performance simulation codes. working paper or preprint.

[Saldaña and Chow, 2006] Saldaña, M. and Chow, P. (2006). TMD-MPI : an MPI implementation for multiple processors across multiple fpgas. In *Proceedings of the 2006 International Conference on Field Programmable Logic and Applications (FPL), Madrid, Spain, August 28-30, 2006*, pages 1–6. IEEE.

[Sanaullah and Herbordt, 2018] Sanaullah, A. and Herbordt, M. C. (2018). FPGA HPC using opencl : Case study in 3d FFT. In *Proceedings of the 9th International Symposium on Highly-Efficient Accelerators and Reconfigurable Technologies, HEART 2018, Toronto, ON, Canada, June 20-22, 2018*, pages 7 :1–7 :6. ACM.

[Sarkauskas et al., 2021] Sarkauskas, N., Bayatpour, M., Tran, T., Ramesh, B., Subramoni, H., and Panda, D. K. (2021). Large-message nonblocking mpi_iallgather and MPI ibcast offload via bluefield-2 DPU. In *28th IEEE International Conference on High Performance Computing, Data, and Analytics, HiPC 2021, Bengaluru, India, December 17-20, 2021*, pages 388–393. IEEE.

[Scales and Gharachorloo, 1997] Scales, D. J. and Gharachorloo, K. (1997). Towards transparent and efficient software distributed shared memory. In Banâtre, M., Levy, H. M., and Waite, W. M., editors, *Proceedings of the Sixteenth ACM Symposium on Operating System Principles, SOSP 1997, St. Malo, France, October 5-8, 1997*, pages 157–169. ACM.

[Sébert et al., 2021] Sébert, A. G., Pinot, R., Zuber, M., Gouy-Pailler, C., and Sirdey, R. (2021). SPEED : secure, private, and efficient deep learning. *Mach. Learn.*, 110(4) :675–694.

[Sebexen and Sohmers, 2015] Sebexen, P. and Sohmers, T. (2015). Software techniques for scratchpad memory management. In Jacob, B. L., editor, *Proceedings of the 2015 International Symposium on Memory Systems, MEMSYS 2015, Washington DC, DC, USA, October 5-8, 2015*, pages 98–102. ACM.

[Sepúlveda et al., 2007] Sepúlveda, J., Strum, M., and Wang, J. (2007). A tlm-based network-on-chip performance evaluation framework. *Proc. 3rd Symposium on Circuits and Systems, Colombian Chapter*, pages 54–60.

[Serra, 2020] Serra, Y. (2020). Le français kalray lance un cpu qui booste réseau, stockage et ia. LeMagIT. https://www.lemagit.fr/actualites/252476856/Le-francais-Kalray-lance-un-CPU-qui-booste-reseau-stockage-et-IA.

[Sidhu and Prasanna, 2001] Sidhu, R. P. S. and Prasanna, V. K. (2001). Fast regular expression matching using fpgas. In *The 9th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, FCCM 2001, Rohnert Park, California, USA, April 29 - May 2, 2001*, pages 227–238. IEEE Computer Society.

[Smith, 2016] Smith, R. (2016). Performance of MPI codes written in python with numpy and mpi4py. In Schreiber, A., Scullin, W., Spotz, W. F., and Terrel, A. R., editors, *6th Workshop on Python for High-Performance and Scientific Computing, PyHPC@SC 2016, Salt Lake, UT, USA, November 14, 2016*, pages 45–51. IEEE.

[Sommer et al., 2017] Sommer, L., Korinth, J., and Koch, A. (2017). Openmp device offloading to FPGA accelerators. In *28th IEEE International Conference on Application-specific Systems, Architectures and Processors, ASAP 2017, Seattle, WA, USA, July 10-12, 2017*, pages 201–205. IEEE Computer Society.

[Sozzo et al., 2017] Sozzo, E. D., Tucci, L. D., and Santambrogio, M. D. (2017). A highly scalable and efficient parallel design of n-body simulation on FPGA. In *2017 IEEE International Parallel and Distributed Processing Symposium Workshops, IPDPS Workshops 2017, Orlando / Buena Vista, FL, USA, May 29 - June 2, 2017*, pages 241–246. IEEE Computer Society.

[Stuecheli et al., 2018] Stuecheli, J., Starke, W. J., Irish, J. D., Arimilli, L. B., Dreps, D. M., Blaner, B., Wollbrink, C., and Allison, B. (2018). IBM POWER9 opens up a new era of acceleration enablement : Opencapi. *IBM J. Res. Dev.*, 62(4/5) :8 :1–8 :8.

[Tachyum, 2020] Tachyum (2020). Tachyum prodigy t16128. Tachyum website. https://www.tachyum.com/products.

[Tang et al., 2020] Tang, X., Giacomin, E., Chauviere, B., Alacchi, A., and Gaillardon, P. (2020). Openfpga : An open-source framework for agile prototyping customizable fpgas. *IEEE Micro*, 40(4) :41–48.

[Tatebe et al., 2002] Tatebe, O., Morita, Y., Matsuoka, S., Soda, N., and Sekiguchi, S. (2002). Grid datafarm architecture for petascale data intensive computing. In *2nd IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2002), 22-24 May 2002, Berlin, Germany*, pages 102–110. IEEE Computer Society.

[Teixeira Monteiro, 2016] Teixeira Monteiro, F. M. (2016). Clock synchronization for many-core processors. Master's thesis, Mestrado Integrado em Engenharia Eletrotécnica e de Computadores.

[Thakur and Gropp, 2007] Thakur, R. and Gropp, W. (2007). Test suite for evaluating performance of MPI implementations that support mpi_thread_multiple. In Cappello, F., Hérault, T., and Dongarra, J. J., editors, *Recent Advances in Parallel Virtual Machine and Message Passing Interface, 14th European PVM/MPI User's Group Meeting, Paris, France, September 30 - October 3, 2007, Proceedings*, volume 4757 of *Lecture Notes in Computer Science*, pages 46–55. Springer.

[Tiwari et al., 2012] Tiwari, A., Laurenzano, M., Peraza, J., Carrington, L., and Snavely, A. (2012). Green queue : Customized large-scale clock frequency scaling. In Liu, J., Chen, J., and Xu, G., editors, *2012 Second International Conference on Cloud and Green Computing, CGC 2012, Xiangtan, Hunan, China, November 1-3, 2012*, pages 260–267. IEEE Computer Society.

[TOP500, 2020] TOP500 (2020). TOP500 the list. https://www.top500.org/.

[Turan and Verbauwhede, 2021] Turan, F. and Verbauwhede, I. (2021). Trust in fpga-accelerated cloud computing. *ACM Comput. Surv.*, 53(6) :128 :1–128 :28.

[Uçar et al., 2006] Uçar, B., Aykanat, C., Kaya, K., and Ikinci, M. (2006). Task assignment in heterogeneous computing systems. *J. Parallel Distributed Comput.*, 66(1) :32–46.

[Vangal et al., 2008] Vangal, S. R., Howard, J., Ruhl, G., Dighe, S., Wilson, H., Tschanz, J. W., Finan, D., Singh, A. P., Jacob, T., Jain, S., Erraguntla, V., Roberts, C., Hoskote, Y., Borkar, N., and Borkar, S. (2008). An 80-tile sub-100-w teraflops processor in 65-nm CMOS. *IEEE J. Solid State Circuits*, 43(1) :29–41.

[Venkatesh et al., 2015] Venkatesh, A., Vishnu, A., Hamidouche, K., Tallent, N. R., Panda, D. K., Kerbyson, D. J., and Hoisie, A. (2015). A case for application-oblivious energy-efficient MPI runtime. In Kern, J. and Vetter, J. S., editors, *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2015, Austin, TX, USA, November 15-20, 2015*, pages 29 :1–29 :12. ACM.

[Vivet et al., 2020] Vivet, P., Guthmuller, E., Thonnart, Y., Pillonnet, G., Moritz, G., Miro-Panadès, I., Tortolero, C. F., Durupt, J., Bernard, C., Varreau, D., Pontes, J. J. H., Thuries, S., Coriat, D., Harrand, M., Dutoit, D., Lattard, D., Arnaud, L., Charbonnier, J., Coudrain, P., Garnier, A., Berger, F., Gueugnot, A., Greiner, A., Meunier, Q. L., Farcy, A., Arriordaz, A., Cheramy, S., and Clermidy, F. (2020). 2.3 A 220gops 96-core processor with 6 chiplets 3d-stacked on an active interposer offering 0.6ns/mm latency, 3tb/s/mm^2 inter-chiplet interconnects and 156mw/mm^2@ 82%-peak-efficiency DC-DC converters. In *2020 IEEE International Solid- State Circuits Conference, ISSCC 2020, San Francisco, CA, USA, February 16-20, 2020*, pages 46–48. IEEE.

[Völker, 2013] Völker, L. (2013). Some/ip – die middleware für ethernet-basierte kommunikation. *Hanser automotive networks*.

[Walker, 1992] Walker, D. W. (1992). Standards for message-passing in a distributed memory environment.

[Watt, 2009] Watt, D. (2009). *Programming XC on XMOS Devices*. XMOS Limited.

[Weil et al., 2006] Weil, S. A., Brandt, S. A., Miller, E. L., Long, D. D. E., and Maltzahn, C. (2006). Ceph : A scalable, high-performance distributed file system. In Bershad, B. N. and Mogul, J. C., editors, *7th Symposium on Operating Systems Design and Implementation (OSDI '06), November 6-8, Seattle, WA, USA*, pages 307–320. USENIX Association.

[Werstein et al., 2003] Werstein, P., Pethick, M., and Huang, Z. (2003). A performance comparison of dsm, pvm, and mpi. In Shen, H. and Fan, P., editors, *Proceedings of the Fourth International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT), Chengdu, China, August 27-29, 2003*, pages 476–482. IEEE.

[WikiChip, 2017] WikiChip (2017). Matrix-2000 - NUDT. WikiChip - Chips & Semi, Semiconductors and Computer Engineering. https://en.wikichip.org/wiki/nudt/matrix-2000.

[WikiChip, 2020] WikiChip (2020). PEZY-SCx (PEZY-SuperComputerx). WikiChip - Chips & Semi, Semiconductors and Computer Engineering. https://en.wikichip.org/wiki/pezy/pezy-scx.

[Wikipedia, 2020] Wikipedia (2020). Xeon phi. Wikipedia website. https://en.wikipedia.org/wiki/Xeon_Phi.

[Wikipedia, 2021] Wikipedia (2021). Transputer. Wikipedia website. https://en.wikipedia.org/wiki/Transputer.

[Williams et al., 2006] Williams, J. A., Syed, I., Wu, J., and Bergmann, N. W. (2006). A reconfigurable cluster-on-chip architecture with MPI communication layer. In *14th IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM 2006), 24-26 April 2006, Napa, CA, USA, Proceedings*, pages 351–352. IEEE Computer Society.

[Yigit, 2003] Yigit, O. (2003). Hash functions. http://www.cse.yorku.ca/~oz/hash.html.

[Zeni et al., 2021] Zeni, A., O'Brien, K., Blott, M., and Santambrogio, M. D. (2021). Optimized implementation of the HPCG benchmark on reconfigurable hardware. In Sousa, L., Roma, N., and Tomás, P., editors, *Euro-Par 2021 : Parallel Processing - 27th International Conference on Parallel and Distributed Computing, Lisbon, Portugal, September 1-3, 2021, Proceedings*, volume 12820 of *Lecture Notes in Computer Science*, pages 616–630. Springer.

[Zhou et al., 1992] Zhou, S., Stumm, M., Li, K., and Wortman, D. B. (1992). Heterogeneous distributed shared memory. *IEEE Trans. Parallel Distributed Syst.*, 3(5) :540–554.

[Zohouri et al., 2018] Zohouri, H. R., Podobas, A., and Matsuoka, S. (2018). Combined spatial and temporal blocking for high-performance stencil computation on fpgas using opencl. In Anderson, J. H. and Bazargan, K., editors, *Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, FPGA 2018, Monterey, CA, USA, February 25-27, 2018*, pages 153–162. ACM.

Publications

Journaux internationaux (2)

[Cudennec, 2020] Cudennec, L. (2020). Adaptive message passing polling for energy efficiency : Application to software-distributed shared memory over heterogeneous computing resources. *Concurr. Comput. Pract. Exp.*, 32(24).

[Oleksiak et al., 2017] Oleksiak, A., Kierzynka, M., Piatek, W., Agosta, G., Barenghi, A., Brandolese, C., Fornaciari, W., Pelosi, G., Cecowski, M., Plestenjak, R., Cinkelj, J., Porrmann, M., Hagemeyer, J., Griessl, R., Lachmair, J., Peykanu, M., Tigges, L., vor dem Berge, M., Christmann, W., Krupop, S., Carbon, A., Cudennec, L., Goubier, T., Philippe, J., Rosinger, S., Schlitt, D., Pieper, C., Adeniyi-Jones, C., Setoain, J., Ceva, L., and Janssen, U. (2017). M2DC - modular microserver datacentre with heterogeneous hardware. *Microprocess. Microsystems*, 52 :117–130.

Chapitre de livre (1)

[Oleksiak et al., 2019] Oleksiak, A., Kierzynka, M., Piatek, W., vor dem Berge, M., Christmann, W., Krupop, S., Porrmann, M., Hagemeyer, J., Griessl, R., Peykanu, M., Tigges, L., Rosinger, S., Schlitt, D., Pieper, C., Janssen, U., Rauchfuss, H., Agosta, G., Barenghi, A., Brandolese, C., Fornaciari, W., Pelosi, G., Pita Costa, J., Cecowski, M., Plestenjak, R., Cinkelj, J., Cudennec, L., Goubier, T., Philippe, J.-M., Adeniyi-Jones, C., Setoain, J., and Ceva, L. (2019). *M2DC—A Novel Heterogeneous Hyperscale Microserver Platform*, pages 109–128. Springer International Publishing, Cham.

Conférences internationales (20)

[Antoniu et al., 2008] Antoniu, G., Cudennec, L., Ghareeb, M., and Tatebe, O. (2008). Building hierarchical grid storage using the gfarmglobal file system and the juxmemgrid data-sharing service. In Luque, E., Margalef, T., and Benitez, D., editors, *Euro-Par 2008 - Parallel Processing, 14th International Euro-Par Conference, Las Palmas de Gran Canaria, Spain, August 26-29, 2008, Proceedings*, volume 5168 of *Lecture Notes in Computer Science*, pages 456–465. Springer.

[Antoniu et al., 2007] Antoniu, G., Cudennec, L., Jan, M., and Duigou, M. (2007). Performance scalability of the JXTA P2P framework. In *21th International Parallel and Distributed Processing Symposium (IPDPS 2007), Proceedings, 26-30 March 2007, Long Beach, California, USA*, pages 1–10. IEEE.

[Antoniu et al., 2006a] Antoniu, G., Cudennec, L., and Monnet, S. (2006a). Extending the entry consistency model to enable efficient visualization for code-coupling grid applications. In *Sixth IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2006), 16-19 May 2006, Singapore*, pages 552–560. IEEE Computer Society.

[Antoniu et al., 2006b] Antoniu, G., Cudennec, L., and Monnet, S. (2006b). A practical evaluation of a data consistency protocol for efficient visualization in grid applications. In Daydé, M. J., Palma, J. M. L. M., Coutinho, A. L. G. A., Pacitti, E., and Lopes, J. C., editors, *High Performance Computing for Computational Science - VECPAR 2006, 7th International Conference, Rio de Janeiro, Brazil, June 10-13, 2006, Revised Selected and Invited Papers*, volume 4395 of *Lecture Notes in Computer Science*, pages 692–706. Springer.

[Aubry et al., 2013] Aubry, P., Beaucamps, P., Blanc, F., Bodin, B., Carpov, S., Cudennec, L., David, V., Dore, P., Dubrulle, P., de Dinechin, B. D., Galea, F., Goubier, T., Harrand, M., Jones, S., Lesage, J., Louise, S., Chaisemartin, N. M., Nguyen, T., Raynaud, X., and Sirdey, R. (2013). Extended cyclostatic dataflow program compilation and execution for an integrated manycore processor. In Alexandrov, V. N., Lees, M., Krzhizhanovskaya, V. V., Dongarra, J. J., and Sloot, P. M. A., editors, *Proceedings of the International Conference on Computational Science, ICCS 2013, Barcelona, Spain, 5-7 June, 2013*, volume 18 of *Procedia Computer Science*, pages 1624–1633. Elsevier.

[Carpov et al., 2013] Carpov, S., Cudennec, L., and Sirdey, R. (2013). Throughput constrained parallelism reduction in cyclo-static dataflow applications. In Alexandrov, V. N., Lees, M., Krzhizhanovskaya, V. V., Dongarra, J. J., and Sloot, P. M. A., editors, *Proceedings of the International Conference on Computational Science, ICCS 2013, Barcelona, Spain, 5-7 June, 2013*, volume 18 of *Procedia Computer Science*, pages 30–39. Elsevier.

[Cecowski et al., 2016] Cecowski, M., Agosta, G., Oleksiak, A., Kierzynka, M., vor dem Berge, M., Christmann, W., Krupop, S., Porrmann, M., Hagemeyer, J., Griessl, R., Peykanu, M., Tigges, L., Rosinger, S., Schlitt, D., Pieper, C., Brandolese, C., Fornaciari, W., Pelosi, G., Plestenjak, R., Cinkelj, J., Cudennec, L., Goubier, T., Philippe, J., Janssen, U., and Adeniyi-Jones, C. (2016). The M2DC project : Modular microserver datacentre. In Kitsos, P., editor, *2016 Euromicro Conference on Digital System Design, DSD 2016, Limassol, Cyprus, August 31 - September 2, 2016*, pages 68–74. IEEE Computer Society.

[Cudennec et al., 2016a] Cudennec, L., Dahmani, S., Gogniat, G., Maignan, C., and Sepúlveda, M. J. (2016a). A fast evaluation approach of data consistency protocols within a compilation toolchain. In Connolly, M., editor, *International Conference on Computational Science 2016, ICCS 2016, 6-8 June 2016, San Diego, California, USA*, volume 80 of *Procedia Computer Science*, pages 2297–2301. Elsevier.

[Cudennec et al., 2016b] Cudennec, L., Dahmani, S., Gogniat, G., Maignan, C., and Sepúlveda, M. J. (2016b). Network contention-aware method to evaluate data coherency protocols within a compilation toolchain. In *10th IEEE International Symposium on Embedded Multicore/Many-core Systems-on-Chip, MCSOC 2016, Lyon, France, September 21-23, 2016*, pages 249–256. IEEE Computer Society.

[Cudennec et al., 2014] Cudennec, L., Dubrulle, P., Galea, F., Goubier, T., and Sirdey, R. (2014). Generating code and memory buffers to reorganize data on many-core architectures. In Abramson, D., Lees, M., Krzhizhanovskaya, V. V., Dongarra, J. J., and Sloot, P. M. A., editors, *Proceedings of the International Conference on Computational Science, ICCS 2014, Cairns, Queensland, Australia, 10-12 June, 2014*, volume 29 of *Procedia Computer Science*, pages 1123–1133. Elsevier.

[Cudennec and Goubier, 2015] Cudennec, L. and Goubier, T. (2015). A short overview of executing Γ chemical reactions over the Σc and τ C datafl w programming models. In Koziel, S., Leifsson, L. T., Lees, M., Krzhizhanovskaya, V. V., Dongarra, J. J., and Sloot, P. M. A., editors, *Proceedings of the International Conference on Computational Science, ICCS 2015, Computational Science at the Gates of Nature, Reykjavík, Iceland, 1-3 June, 2015, 2014*, volume 51 of *Procedia Computer Science*, pages 1413–1422. Elsevier.

[Cudennec and Sirdey, 2012] Cudennec, L. and Sirdey, R. (2012). Parallelism reduction based on pattern substitution in dataflow oriented programming languages. In Ali, H. H., Shi, Y., Khazanchi, D., Lees, M., van Albada, G. D., Dongarra, J. J., and Sloot, P. M. A., editors, *Proceedings of the International Conference on Computational Science, ICCS 2012, Omaha, Nebraska, USA, 4-6 June, 2012*, volume 9 of *Procedia Computer Science*, pages 146–155. Elsevier.

[Dahmani et al., 2014] Dahmani, S., Cudennec, L., Louise, S., and Gogniat, G. (2014). Using the spring physical model to extend a cooperative caching protocol for many-core processors. In *IEEE 8th International Symposium on Embedded Multicore/Manycore SoCs, MCSoC 2014, Aizu-Wakamatsu, Japan, September 23-25, 2014*, pages 303–310. IEEE Computer Society.

[Kierzynka et al., 2016] Kierzynka, M., Oleksiak, A., Agosta, G., Brandolese, C., Fornaciari, W., Pelosi, G., vor dem Berge, M., Christmann, W., Krupop, S., Cecowski, M., Plestenjak, R., Cinkelj, J., Porrmann, M., Hagemeyer, J., Griessl, R., Peykanu, M., Tigges, L., Cudennec, L., Goubier, T., Philippe, J., Rosinger, S., Schlitt, D., Pieper, C., Adeniyi-Jones, C., Janssen, U., and Ceva, L. (2016). Data centres for iot applications : The M2DC approach (invited paper). In Najjar, W. A. and Gerstlauer, A., editors, *International Conference on Embedded Computer Systems : Architectures, Modeling and Simulation, SAMOS 2016, Agios Konstantinos, Samos Island, Greece, July 17-21, 2016*, pages 293–299. IEEE.

[Lenormand et al., 2020] Lenormand, E., Goubier, T., Cudennec, L., and Charles, H.-P. (2020). A combined fast/cycle accurate simulation tool for reconfigurable accelerator evaluation : application to distributed data management. In *2020 International Symposium on Rapid System Prototyping, RSP 2020, Virtual Conference, September 24-25, 2020*. IEEE.

[Marandola et al., 2016] Marandola, J., Louise, S., and Cudennec, L. (2016). Pattern based cache coherency architecture for embedded manycores. In Connolly, M., editor, *International Conference on Computational Science 2016, ICCS 2016, 6-8 June 2016, San Diego, California, USA*, volume 80 of *Procedia Computer Science*, pages 1542–1553. Elsevier.

[Marandola et al., 2012] Marandola, J., Louise, S., Cudennec, L., Acquaviva, J., and Bader, D. A. (2012). Enhancing cache coherent architectures with access patterns for embedded manycore systems. In *2012 International Symposium on System on Chip, ISSoC 2012, Tampere, Finland, October 10-12, 2012*, pages 1–7. IEEE.

[Pfrommer et al., 2016] Pfrommer, J., Schleipen, M., Azaiez, S., Boc, M., Cudennec, L., Kchir, S., Tortech, T., and Klinge, X. (2016). Deploying software functionality to manufacturing resources safely at runtime. In *21st IEEE International Conference on Emerging Technologies and Factory Automation, ETFA 2016, Berlin, Germany, September 6-9, 2016*, pages 1–7. IEEE.

[Stan et al., 2020] Stan, O., Cudennec, L., and Syoën, L. (2020). Attribute-based encryption and its application to a software-distributed shared memory. In Samarati, P., di Vimercati, S. D. C., Obaidat, M. S., and Ben-Othman, J., editors, *Proceedings of the 17th International Joint Conference on e-Business and Telecommunications, ICETE 2020 - Volume 2 : SECRYPT, Lieusaint, Paris, France, July 8-10, 2020*, pages 625–631. ScitePress.

[Marandola and Cudennec, 2011] Marandola, J. and Cudennec, L. (2011). Co-Designed Cache Coherency Architecture for Embedded Multicore Systems. In *IP-Embedded System Conference and Exhibition*, volume 20, Grenoble, France. Design and Reuse.

Workshops internationaux (13)

[Almousa Almaksour et al., 2007] Almousa Almaksour, A., Antoniu, G., Bougé, L., Cudennec, L., and Gançarski, S. (2007). Building a DBMS on top of the JuxMem Grid Data-Sharing Service. In *HiPerGRID Workshop*, Brasov, Romania. Held in conjunction with Parallel Architectures and Compilation Techniques 2007 (PACT2007).

[Azaiez et al., 2016] Azaiez, S., Boc, M., Cudennec, L., Simoes, M. D. S., Haupert, J., Kchir, S., Klinge, X., Labidi, W., Nahhal, K., Pfrommer, J., Schleipen, M., Schulz, C., and Tortech, T. (2016). Towards flexibility in future industrial manufacturing : A global framework for self-organization of production cells. In Shakshuki, E. M., editor, *The 7th International Conference on Ambient Systems, Networks and Technologies (ANT 2016) / The 6th International Conference on Sustainable Energy Information Technology (SEIT-2016) / Affiliated Workshops, May 23-26, 2016, Madrid, Spain*, volume 83 of *Procedia Computer Science*, pages 1268–1273. Elsevier.

[Chaker et al., 2015] Chaker, H., Cudennec, L., Dahmani, S., Gogniat, G., and Sepúlveda, M. J. (2015). Cycle-based model to evaluate consistency protocols within a multi-protocol compilation tool-chain. In Wang, Z., Petoumenos, P., and Leather, H., editors, *Proceedings of the 2015 International Workshop on Code Optimisation for Multi and Many Cores, COSMIC@CGO 2015, San Francisco Bay Area, CA, USA, February 8, 2015*, pages 8 :1–8 :10. ACM.

[Cudennec, 2017] Cudennec, L. (2017). Software-distributed shared memory over heterogeneous micro-server architecture. In Heras, D. B., Bougé, L., Mencagli, G., Jeannot, E., Sakellariou, R., Badia, R. M., Barbosa, J. G., Ricci, L., Scott, S. L., Lankes, S., and Weidendorfer, J., editors, *Euro-Par 2017 : Parallel Processing Workshops - Euro-Par 2017 International Workshops, Santiago de Compostela, Spain, August 28-29, 2017, Revised Selected Papers*, volume 10659 of *Lecture Notes in Computer Science*, pages 366–377. Springer.

[Cudennec, 2018] Cudennec, L. (2018). Merging the publish-subscribe pattern with the shared memory paradigm. In Mencagli, G., Heras, D. B., Cardellini, V., Casalicchio, E., Jeannot, E., Wolf, F., Salis, A., Schifanella, C., Manumachu, R. R., Ricci, L., Beccuti, M., Antonelli, L., Sánchez, J. D. G., and Scott, S. L., editors, *Euro-Par 2018 : Parallel Processing Workshops - Euro-Par 2018 International Workshops, Turin, Italy, August 27-28, 2018, Revised Selected Papers*, volume 11339 of *Lecture Notes in Computer Science*, pages 469–480. Springer.

[Cudennec et al., 2008] Cudennec, L., Antoniu, G., and Bougé, L. (2008). CoRDAGe : towards transparent management of interactions between applications and ressources. In *International Workshop on Scalable Tools for High-End Computing (STHEC 2008)*, pages 13–24, Kos, Greece. Michael Gerndt and Jesus Labarta and Barton Miller.

[Cudennec and Trabelsi, 2020] Cudennec, L. and Trabelsi, K. (2020). Experiments using a software-distributed shared memory, MPI and 0mq over heterogeneous computing resources. In Balis, B., Heras, D. B., Antonelli, L., Bracciali, A., Gruber, T., Hyun-Wook, J., Kuhn, M., Scott, S. L., Unat, D., and Wyrzykowski, R., editors, *Euro-Par 2020 : Parallel Processing Workshops - Euro-Par 2020 International Workshops, Warsaw, Poland, August 24-25, 2020, Revised Selected Papers*, volume 12480 of *Lecture Notes in Computer Science*, pages 237–248. Springer.

[Dahmani et al., 2015] Dahmani, S., Carpov, S., Cudennec, L., and Gogniat, G. (2015). A multiobjective consistency decision engine for a manycore compilation platform using an evolutionary approach. In *Evolutionary Multiobjective Optimization (EMO 2015), held in conjunction with the 23rd Intl. Conf. on Multiple Criteria Decision Making (MCDM 2015)*, Hamburg, Germany.

[Dahmani et al., 2013] Dahmani, S., Cudennec, L., and Gogniat, G. (2013). Introducing a data sliding mechanism for cooperative caching in manycore architectures. In *2013 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum, Cambridge, MA, USA, May 20-24, 2013*, pages 335–344. IEEE.

[Lenormand et al., 2021] Lenormand, E., Goubier, T., Cudennec, L., and Charles, H.-P. (2021). Data management model to program irregular compute kernels on fpga : application to heterogeneous distributed system. In *Euro-Par 2021 : Parallel Processing Workshops - Euro-Par 2021 International Workshops, Lisbon, Portugal, August 30 - September 3, 2021, Revised Selected Papers*, Lecture Notes in Computer Science. Springer.

[Prigent et al., 2022] Prigent, C., Cudennec, L., Costan, A., and Antoniu, G. (2022). A methodology to build decision analysis tools applied to distributed reinforcement learning. In *2022 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum, Lyon, France, 2022, To Appear*. IEEE.

[Trabelsi et al., 2019] Trabelsi, K., Cudennec, L., and Bennour, R. (2019). Application topology definition and tasks mapping for efficient use of heterogeneous resources. In Schwardmann, U., Boehme, C., Heras, D. B., Cardellini, V., Jeannot, E., Salis, A., Schifanella, C., Manumachu, R. R., Schwamborn, D., Ricci, L., Sangyoon, O., Gruber, T., Antonelli, L., and Scott, S. L., editors, *Euro-Par 2019 : Parallel Processing Workshops - Euro-Par 2019 International Workshops, Göttingen, Germany, August 26-30, 2019, Revised Selected Papers*, volume 11997 of *Lecture Notes in Computer Science*, pages 258–269. Springer.

Conférences nationales (7)

[Lenormand et al., 2019] Lenormand, E., Cudennec, L., and Charles, H.-P. (2019). Unification des mémoires réparties dans un système hétérogène avec accélérateur reconfigurable : exposé de principe. In *Conférence d'informatique en*

Parallélisme, Architecture et Système (Compas'2019), Anglet, France. LIUPPA - Laboratoire d'Informatique de l'Université de Pau et des Pays de l'Adour (EA 3000).

[Chaker et al., 2016] Chaker, H., Cudennec, L., Dahmani, S., Gogniat, G., Maignan, C., and Sepùlveda, M. J. (2016). Modèles temporels d'évaluation des protocoles de gestion des données. In *Conférence d'informatique en Parallélisme, Architecture et Système (Compas'2016)*, Lorient, France.

[Cudennec, 2008] Cudennec, L. (2008). Un service hiérarchique distribué de partage de données pour grille. In *Rencontres francophones du Parallélisme (RenPar '18)*, Fribourg, Switzerland. Haute Ecole Spécialisée de Suisse Occidentale (HES-SO).

[Cudennec and Goubier, 2015] Cudennec, L. and Goubier, T. (2015). Compilation itérative pour l'exécution de programmes chimiques sur une chaîne de compilation flot de données. In *16ème conférence ROADEF Société Française de Recherche Opérationnelle et Aide à la Décision*, Marseille, France.

[Cudennec and Monnet, 2005] Cudennec, L. and Monnet, S. (2005). Extension du modèle de cohérence à l'entrée pour la visualisation dans les applications de couplage de codes sur grilles. In *Actes des Journées francophones sur la Cohérence des Données en Univers Réparti*, Paris, France.

[Dahmani et al., 2015] Dahmani, S., Cudennec, L., and Gogniat, G. (2015). Aide à la décision pour le choix et le paramétrage de protocoles de cohérence des données. In *16ème conférence ROADEF Société Française de Recherche Opérationnelle et Aide à la Décision*, Marseille, France.

[Lenormand et al., 2021] Lenormand, E., Goubier, T., Cudennec, L., and Charles, H.-P. (2021). Modèle de programmation pour noyaux de calcul irréguliers sur accélérateur reconfigurable dans un système distribué hétérogène. In *Conférence d'informatique en Parallélisme, Architecture et Système (Compas'2021)*, Lyon, France.

Autres (7)

[Cudennec, 2005] Cudennec, L. (2005). Modèles et protocoles de cohérence des données en environnement volatile. Rapport de fin d'études. INSA de Rennes, INRIA Centre Bretagne Atlantique, IRISA.

[Cudennec, 2009] Cudennec, L. (2009). *CoRDAGe : Un service générique de co-déploiement et redéploiement d'applications sur grilles. (CoRDAGe : a generic service for co-deploying and re-deploying grid applications).* PhD thesis, Université de Rennes 1, France.

[Cudennec, 2020] Cudennec, L. (2020). Software-distributed shared memory for heterogeneous machines : Design and use considerations. *CoRR*, abs/2009.01507.

[Cudennec et al., 2012] Cudennec, L., Kofuji, J., Acquaviva, J.-T., and Camier, J.-S. (2012). Multi-core system and method of data consistency. Patent FR2970794 (A1), Commissariat à l'Énergie Atomique.

[Cudennec et al., 2009] Cudennec, L., Antoniu, G., and Bougé, L. (2009). Experimentations With CoRDAGe, A Generic Service For Co-Deploying and Re-Deploying Applications On Grids. Research Report RR-7086, INRIA.

[Dahmani et al., 2013] Dahmani, S., Cudennec, L., and Gogniat, G. (2013). Adaptive Cooperative Caching for Many-cores systems. Proceedings of the Ninth International Summer School on Advanced Computer Architecture and Compilation for High-Performance and Embedded Systems (HiPEAC ACACES 2013).

[Le Guillou and Cudennec, 2008] Le Guillou, X. and Cudennec, L. (2008). Vers la classification darwinienne d'un processeur fossile. In Heliot, R. and Zimmermann, A., editors, *The Third Review of April Fool's day Transactions (RAFT'2008)*, pages 9–21, Grenoble, France. This paper has been awarded "Editor's Choice" of the RAFT 2008 proceedings.

Contributions

Bibliographie	Actes	Rang	Init.	Idées	Impl.	Exp.	Rédac.	Prés.
[Prigent et al., 2022]	ScaDL@IPDPS		★	■	□	□	■	□
[Lenormand et al., 2021]	Heteropar@Euro-Par		★	■	□	□	□	□
[Lenormand et al., 2020]	RSP@ESWEEK	C/B2	★	■	□	□	■	□
[Cudennec and Trabelsi, 2020]	ParaMo@Euro-Par		■	■	■	■	■	■
[Cudennec, 2020]	Wiley & Sons CPE	IF 1.536	■	■	■	■	■	
[Stan et al., 2020]	SECRYPT	B/B4	★	■	■	□	■	□
[Trabelsi et al., 2019]	Heteropar@Euro-Par		★	■	□	■	■	□
[Cudennec, 2018]	Heteropar@Euro-Par		■	■	■	■	■	■
[Oleksiak et al., 2017]	Elsevier MICPRO	IF 1.525	□	□	□	□	■	
[Cudennec, 2017]	Heteropar@Euro-Par		■	■	■	■	■	■
[Pfrommer et al., 2016]	ETFA	B1	□	■	■	□	■	□
[Cecowski et al., 2016]	DSD	B1	□	□	□	□	■	□
[Cudennec et al., 2016b]	MCSoC		★	■	□	□	■	■
[Cudennec et al., 2016a]	ICCS	A/A2	★	■	□	□	■	■
[Marandola et al., 2016]	ICCS	A/A2	★	■	□	□	■	□
[Kierzynka et al., 2016]	SAMOS	B2	□	□	□	□	■	□
[Azaiez et al., 2016]	RAMCOM@ANT		□	■	■	□	■	□
[Dahmani et al., 2015]	EMO@MCDM	C	★	■	□	□	■	□
[Cudennec and Goubier, 2015]	ICCS	A/A2	■	■	■	■	■	■
[Chaker et al., 2015]	COSMIC@CGO		★	■	□	□	■	□
[Dahmani et al., 2014]	MCSoC		★	■	□	□	■	□
[Cudennec et al., 2014]	ICCS	A/A2	■	■	■	■	■	■
[Aubry et al., 2013]	ICCS	A/A2	□	■	■	□	■	□
[Carpov et al., 2013]	ICCS	A/A2	□	■	□	□	□	□
[Dahmani et al., 2013]	HIPS@IPDPS	C/B4	★	■	□	□	■	□
[Marandola et al., 2012]	SOC	B4	★	■	□	□	■	□
[Cudennec and Sirdey, 2012]	ICCS	A/A2	■	■	■	■	■	■
[Antoniu et al., 2008]	Euro-Par	A/A2	★	■	■	■	■	■
[Cudennec et al., 2008]	STHEC@ICS		■	■	■	■	■	■
[Almousa Almaksour et al., 2007]	HiPerGRID		★	■	□	□	■	□
[Antoniu et al., 2007]	IPDPS	A/A1	□	■	■	■	■	□
[Antoniu et al., 2006b]	VECPAR	B/B4	■	■	■	■	■	□
[Antoniu et al., 2006a]	CCGrid	A/A1	■	■	■	■	■	■
Implication			76%	91%	48%	39%	94%	35%

Grille de lecture :
— **Initiative** (Init.) Définition du sujet de recherche. ★ Encadrement doctorant ou étudiant en Master.
— **Idées** Contribution aux idées présentées dans la publication.
— **Implémentation** (Impl.) Contribution à l'implémentation des idées.
— **Expérimentations** (Exp.) Contribution à la réalisation des expérimentations.
— **Rédaction** (Rédac.) Contribution à la rédaction de la publication.
— **Présentation** (Prés.) Présentation de la publication en conférence.