Frédéric Magoulès
François-Xavier Roux

Calcul scientifique parallèle

Cours, exemples avec OpenMP et MPI et exercices corrigés

DUNOD

La série « Mathématiques pour le Master/SMAI » propose une nouvelle génération de livres adaptés aux étudiants de Master niveau M1 et aux élèves ingénieurs. Leur adéquation au cursus LMD et aux outils de calcul modernes sont au service de la qualité scientifique.

La SMAI (Société de Mathématiques Appliquées et Industrielles) assure la direction éditoriale grâce à un comité renouvelé périodiquement et largement représentatif des différents thèmes des mathématiques appliquées et de leur évolution : analyse numérique, probabilités appliquées, statistique, optimisation, systèmes dynamiques et commande, traitement d'images et du signal, finance, recherche opérationnelle, etc. Son ambition est de constituer un ensemble d'ouvrages de référence.

© Dunod, Paris, 2013
ISBN 978-2-10-058904-3

Table des matières

Avant-propos

Le calcul scientifique est devenu un outil indispensable dans de très nombreux domaines comme la physique, la mécanique, la biologie, la finance ou l'industrie. Il permet par exemple grâce à des algorithmes performants et adaptés aux ordinateurs actuels de simuler sans l'aide de maquettes ou d'expérimentations la déformation d'un toit sous le poids de la neige, le niveau sonore dans une salle de théâtre, ou l'écoulement d'un fluide autour d'une aile d'avion.

Le but de cet ouvrage est d'expliquer et d'illustrer par des exemples concrets les techniques récentes de calcul scientifique pour la simulation numérique de problèmes de grandes tailles issus de systèmes modélisés par des équations aux dérivées partielles. Les différentes méthodes de formation et de résolution des grands systèmes linéaires sont présentées. La conception des méthodes récentes et les algorithmes associés sont étudiés en détail. Des techniques de mise en œuvre et de programmation sont exposées pour les solveurs directs, les solveurs itératifs, et les méthodes de décomposition de domaine. Les techniques de programmation par échanges de messages et de parallélisme de boucle sont illustrées par des exemples avec MPI et OpenMP.

L'objectif principal de l'ouvrage est l'étude des techniques numériques appliquées au parallélisme pour des machines à très grands nombres de processeurs et à mémoire distribuée. Des connaissances en analyse numérique ainsi que des notions de base en informatique sont requises pour une compréhension optimale. Toutefois, même si le fonctionnement des calculateurs scientifiques est présenté, l'ouvrage ne va pas au-delà de ce qui est utile pour écrire des programmes efficaces. Le principe est donc de faire un exposé assez complet des méthodes numériques récentes en calcul scientifique en insistant évidemment sur leur adaptation au calcul parallèle. Un certain nombre d'exemples d'algorithmes parallèles plus ou moins classiques en calcul scientifique sont présentés. La plupart de ces exemples sont tirés de problèmes provenant de la mise en œuvre de la méthode des éléments finis.

Une approche didactique, qui consiste à introduire les notions mathématiques et informatiques au fur et à mesure que leur besoin se fait sentir, a été suivie. C'est le cas notamment du fait de l'introduction de nouvelles caractéristiques architecturales

des machines, ou de nouveaux problèmes de gestion du parallélisme posés par la complexité croissante des applications présentées en exemple. L'objectif de ce livre n'est donc pas d'apparaître comme un ouvrage de référence sur le calcul parallèle du point de vue informatique, mais d'être une introduction à cette question pour des utilisateurs de calculateurs scientifiques.

Cet ouvrage s'adresse principalement aux étudiants en Master de mathématiques appliquées mais aussi de mécanique numérique et plus généralement aux étudiants dans tous les domaines des sciences de l'ingénieur qui s'intéressent au calcul à haute performance. Il pourra également intéresser tout ingénieur confronté à la simulation numérique de problèmes de grandes tailles issus de systèmes modélisés par des équations aux dérivées partielles mais aussi plus généralement à la résolution de grands systèmes linéaires. Des parties de ce livre ont d'ailleurs été utilisées pendant plusieurs années par les auteurs comme support de cours sur le calcul scientifique à l'Université Pierre et Marie Curie, à l'Université Henri Poincaré, au Conservatoire National des Arts et Métiers, à l'École Centrale des Arts et Manufactures, à l'École Normale Supérieure de Cachan, à l'École Supérieure des Sciences et Technologies de l'Ingénieur de Nancy et à l'Institut des Sciences de l'Ingénieur de Toulon et du Var.

Introduction

Le développement récent des architectures informatiques (fréquence d'horloge, mémoire cache, mémoire hiérarchique, multi-cœur, etc.) a abouti ce jour à des calculateurs scientifiques disposant de plus de 200 000 cœurs, lesquels réalisent souvent plus de mille milliards d'opérations flottantes par seconde. À titre de comparaison, ce chiffre correspondrait à plus d'opérations en une seconde, que ne pourrait en faire la population mondiale pendant 48 heures, à raison d'une opération flottante par seconde par personne ! Toutefois, disposer de calculateurs scientifiques ayant une puissance de calcul aussi élevée n'est pas tout . . . encore faut-il disposer d'algorithmes adéquats pour utiliser de telles ressources.

Ce livre constitue une introduction au calcul à haute performance. Son objectif est de présenter quelques méthodes numériques permettant de résoudre sur calculateurs scientifiques certains problèmes issus des sciences de l'ingénieur qui ne peuvent être traités sur un ordinateur classique. Les questions courantes du calcul à haute performance sont successivement abordées : le parallélisme de données, la vectorisation, les tâches communicantes, la formation parallèle des matrices, la parallélisation du produit de matrices, les méthodes directes et itératives parallèles de résolution de grands systèmes linéaires, etc. La présentation de ces méthodes est rendue vivante par le recours systématique aux environnements de programmation MPI et OpenMP dont les principales commandes sont introduites progressivement. Tous les algorithmes sont ici présentés sous la forme de pseudo-codes. Ceci permet de visualiser très rapidement les propriétés de ces algorithmes, en particulier l'enchaînement des opérations, les dépendances entre les données, etc. Certains algorithmes présentés dans le livre ont été programmés en langage C et/ou langage Fortran par les auteurs et sont téléchargeable sur le site de la maison d'édition. La résolution de divers problèmes, souvent motivés par des applications concrètes, fait l'objet de nombreux exemples et exercices. Afin de guider le lecteur dans la résolution des exercices proposés, des pistes et des idées de corrections sont fournies. À la fin de chaque chapitre, une section présente des aspects plus avancés et fournit des indications bibliographiques qui permettront au lecteur d'approfondir les connaissances acquises. Dans ce but, l'ou-

vrage s'articule autour de quatre parties distinctes.

La première partie introduit dans le chapitre 1, l'architecture des calculateurs scientifiques, les différents types de parallélisme, et l'architecture mémoire des calculateurs scientifiques. Le chapitre 2 présente les modèles de programmation, les critères de performances et la parallélisation de données. Pour sa part, le chapitre 3, présente l'exemple concret du produit de matrices pour illustrer la démarche de parallélisation, de localisation temporelle et spatiale des données puis de distribution.

La deuxième partie consiste en un bref et succinct complément d'analyse numérique matricielle. Le chapitre 4 rappelle tout d'abord quelques notions de base en algèbre linéaire, les propriétés des matrices, ainsi que les notations utilisées dans la suite de l'ouvrage. Le chapitre 5 s'intéresse plus particulièrement aux matrices creuses, notamment à l'origine et à la formation en parallèle de ces matrices dans le cadre de méthodes d'éléments finis. Le chapitre 6 présente succinctement les principales méthodes de résolution des systèmes linéaires. La mise en œuvre parallèle de ces méthodes étant détaillée dans les parties suivantes.

La troisième partie étudie les méthodes de résolution de grands systèmes linéaires. Après avoir présenté dans le chapitre 7 le principe des méthodes directes (LU, Cholesky, Gauss-Jordan, factorisation de Crout), le chapitre 8, puis le chapitre 9 s'intéressent respectivement à la parallélisation des méthodes LU pour les matrices pleines, puis creuses.

La quatrième partie traite des méthodes itératives de résolution de grands systèmes linéaires par les méthodes de Krylov. Un rapide rappel des espaces de Krylov et de la construction de la base d'Arnoldi est détaillé au chapitre 10. Le chapitre 11 présente les méthodes de Krylov avec orthogonalisation complète pour des matrices symétriques définies positives. Le chapitre 12 s'intéresse aux méthodes avec orthogonalisation exacte pour des matrices quelconques, suivi au chapitre 13 des méthodes avec bi-orthogonalisation pour des matrices non symétriques. Les techniques de parallélisation des méthodes de Krylov sont abordées et détaillées au chapitre 14. Les techniques de préconditionnement et les méthodes hybrides de type décomposition de domaines sont ensuite brièvement décrites au chapitre 15.

L'utilisation efficace des calculateurs parallèles pour la simulation numérique exige un effort de conception des logiciels qui doit nécessairement s'appuyer sur la connaissance et l'analyse des méthodes mathématiques. Cette question des méthodes est primordiale et présente un champ très actif de la recherche dans le domaine des mathématiques appliquées ; comme par exemple les méthodes de décomposition de domaines introduites à la fin de l'ouvrage. Cet ouvrage aura atteint son but si le lecteur a compris d'une part pourquoi le parallélisme était incontournable pour les grandes applications et d'autre part que les problèmes posés par la parallélisation demandaient de repenser les méthodes mathématiques et algorithmes numériques.

Chapitre 1

Architectures des calculateurs

Ce chapitre n'a pas la prétention de présenter un cours d'informatique. Seules les caractéristiques architecturales qui ne sont pas transparentes pour l'utilisateur, c'est-à-dire celles qui doivent nécessairement être prises en compte dans les codes de calcul scientifique pour tirer les meilleures performances possibles des calculateurs, sont ici présentées. Il n'est donc pas question de rentrer dans le détail des technologies matérielles et logicielles des systèmes informatiques, mais juste de décrire les principes et notions indispensables à connaître.

1.1 Différents types de parallélisme

1.1.1 Recouvrement, concurrence et parallélisme

L'objectif de la simulation numérique est d'approcher le mieux possible la réalité physique à l'aide de modèles discrets. Plus le modèle est riche, plus il compte de paramètres et plus il requiert une quantité de calcul importante. La fonction des calculateurs scientifiques est de permettre la réalisation de ces calculs en un temps suffisamment court pour que l'outil de simulation soit exploitable dans le cadre d'un processus de conception ou de prévision.

Le critère naturel de performance pour un calculateur scientifique, est la vitesse de calcul, c'est-à-dire le nombre d'opérations arithmétiques réalisable par seconde. Ces opérations arithmétiques, somme ou différence, produit, quotient, portent sur des données de type réel ou complexe, représentées par des nombres en virgule flottante, *floating point* en anglais. Le terme de virgule flottante signifie qu'un nombre réel est représenté par deux entiers, une mantisse et un exposant. Un ordinateur travaille en base 2 ; la valeur d'un nombre réel représenté en virgule flottante est donc égale à la mantisse fois 2 puissance exposant. L'unité de vitesse de calcul est le flops, ce qui signifie *floating point operation per second*. Avec la fréquence des micro-processeurs

actuels, on parle plus fréquemment de Gflops (Giga = 10^9), milliard d'opérations par seconde, de Tflops (Tera = 10^{12}), mille milliards d'opérations par seconde, et même de Pflops (Peta = 10^{15}), million de milliards d'opérations par seconde. Cette vitesse est dépendante de la technologie des composants, qui peut elle-même se mesurer a priori par la fréquence des micro-processeurs. Jusqu'au début des années 2000, les améliorations de l'intégration des circuits de semi-conducteurs et de la finesse de leur gravure ont permis une augmentation moyenne de la fréquence des micro-processeurs d'un facteur deux tous les dix-huit mois, en moyenne. Depuis quelques années, la fréquence est restée bloquée à quelques GHz, l'augmentation de la fréquence de fonctionnement provoquant des effets thermiques qui entraînent une consommation excessive et des difficultés techniques très sérieuses pour évacuer la chaleur.

Les calculateurs scientifiques les plus rapides du début des années 1980 avaient des fréquences d'horloge de l'ordre de 100 MHz et des vitesses de pointe de 100 Mflops (Mega = 10^6), million d'opérations par seconde. Un peu plus de vingt ans plus tard, la fréquence est de quelques GHz et les vitesses de pointe de l'ordre de quelques Pflops. C'est-à-dire, pour schématiser, que le facteur d'augmentation de la vitesse dû à la seule évolution des composants électroniques de base a été de l'ordre de quelques dizaines, alors que l'augmentation de la puissance de calcul a été de quelques centaines de milliers. Comment cela est-il possible ? L'explication réside dans l'évolution des architectures des calculateurs et, plus précisément, dans le recours à ce que l'on appelle le parallélisme. La façon la plus naturelle de dépasser la limite de vitesse liée à la fréquence, consiste à dupliquer les unités fonctionnelles arithmétiques : avec deux additionneurs on pourra aller deux fois plus vite qu'avec un seul, si on les fait travailler simultanément. L'amélioration permanente de la technologie des semi-conducteurs ne conduit plus à augmenter la fréquence, mais elle permet une plus grande intégration, ce qui autorise à mettre de plus en plus d'unités fonctionnelles sur la même puce, jusqu'à dupliquer complètement le cœur du processeur. En poussant cette logique plus loin, il est aussi possible de multiplier les processeurs dans une même machine. Une architecture de calculateur présentant des unités fonctionnelles ou des processeurs multiples capables de fonctionner simultanément pour exécuter une application est qualifiée de "parallèle". Le terme de "calculateur parallèle" désigne généralement une machine disposant de plusieurs processeurs. C'est à ce type de système que s'intéressera principalement la suite de l'ouvrage.

Mais avant même de développer des architectures parallèles, les constructeurs se sont toujours préoccupés d'utiliser au mieux les ressources des calculateurs, et en particulier d'éviter autant que possible les temps morts. Cela passe par le recouvrement des temps d'exécution des différentes instructions. Pour effectuer plus vite un ensemble d'opérations qui utilisent successivement des composants distincts, comme la mémoire, le bus de données, les unités fonctionnelles arithmétiques, on peut commencer l'exécution d'une instruction complexe avant que la précédente ne soit termi-

née. On parle alors de recouvrement.

Plus généralement, il est parfois possible d'effectuer simultanément des opérations distinctes, comme accéder à une mémoire principale ou secondaire d'un côté et exécuter une série d'opérations arithmétiques dans le processeur d'un autre côté. Il s'agit alors de concurrence. Ce genre de technique est mis en œuvre depuis très longtemps dans tous les systèmes capables d'effectuer plusieurs tâches à la fois, en temps partagé. Il s'agit alors d'optimiser le rendement global du système, sans accélérer forcément le temps d'exécution propre de chaque tâche.

Les choses sont plus compliquées lorsqu'il s'agit d'accélérer l'exécution d'un seul programme, ce qui nous intéresse dans ce livre. Il va falloir alors exhiber des paquets d'instructions susceptibles de bénéficier d'un traitement concurrent. Ce qui va nécessiter non seulement des adaptations du matériel, mais aussi du logiciel. Le parallélisme est donc un mode particulier de fonctionnement concurrent dans le cas où des parties du processeur ou de la machine complète ont été dupliquées afin de pouvoir exécuter simultanément des instructions ou des paquets d'instructions souvent semblables participant à une même application.

1.1.2 Parallélisme temporel et spatial pour les unités arithmétiques

Le parallélisme introduit dans le paragraphe précédent est parfois qualifié de parallélisme spatial. Pour augmenter la production, on peut dupliquer les ateliers : avec trois unités, on peut tripler la production.

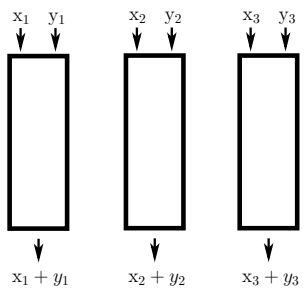

FIGURE 1.1 – Parallélisme spatial : duplication des unités.

Il existe aussi un parallélisme qualifié de "temporel", qui repose sur le recouvrement synchronisé d'instructions successives semblables. Le modèle est celui de la chaîne d'assemblage. Le principe de la chaîne d'assemblage consiste à tronçonner le processus de fabrication en séries d'opérations successives, de durée similaire. Si la chaîne possède trois niveaux, alors, lorsque les opérations du premier niveau sont terminées, on passe l'objet en cours d'assemblage au deuxième niveau et on commence immédiatement les opérations de premier niveau pour un nouvel objet, et ainsi de

suite. De sorte que, si le temps total de fabrication de chaque objet est de trois cycles, il sort en fait un nouvel objet de la chaîne tous les cycles. Schématiquement, on va aussi vite qu'avec trois ateliers complets fonctionnant simultanément. Ce mode de fonctionnement permet d'une part d'éviter de dupliquer tous les outils et d'autre part d'assurer un flux plus continu au niveau des approvisionnements et de la production. Ce mode de fonctionnement pour des unités fonctionnelles d'un ordinateur est qualifié de mode *pipe-line*. Ce terme vient de ce qu'un pipe-line fait effectivement du transport chaîné par rapport à un système de transport par des unités indépendantes, camions, trains ou bateaux.

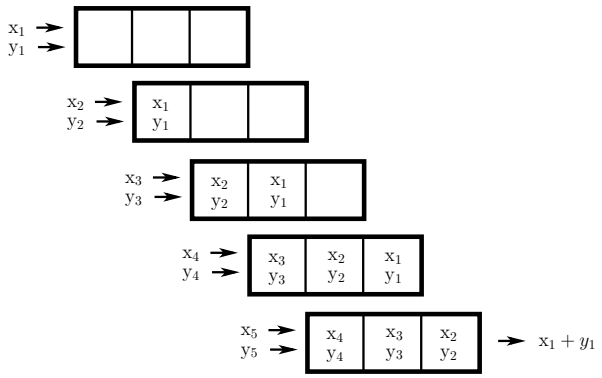

FIGURE 1.2 – Parallélisme temporel : pipe-line d'addition.

Pour illustrer le mode de fonctionnement pipe-line, considérons l'exemple de l'addition de nombres en virgule flottante. L'opération se fait en trois étapes. La première consiste à comparer les exposants, de façon à pouvoir aligner les mantisses, la seconde, à additionner les mantisses et la troisième, à normaliser le résultat en tronquant ou décalant la mantisse. Prenons un exemple, écrit en base 10 pour simplifier la lecture. On suppose que la mantisse possède 4 chiffres. Pour additionner 1234×10^{-4} et -6543×10^{-5}, on constate d'abord que $-4-(-5) = 1$. Il faut donc décaler la deuxième mantisse d'une case vers la droite. C'est exactement ce que l'on fait lorsque l'on écrit deux opérandes décimales l'une au dessus de l'autre en alignant la position de la virgule. Ensuite on calcule $1234 + (-0654) = 0580$. Enfin, la normalisation du résultat consiste à redécaler d'une case la mantisse vers la gauche en abaissant l'exposant de 1, ce qui donne pour résultat final 5800×10^{-5}. Au passage, on peut noter que, de même que la précision de la représentation décimale d'un nombre réel est limité par la taille de la mantisse, les opérations s'effectuent avec une approximation due à cette troncature, et ce, même si l'on étend momentanément la taille des mantisses pour limiter les erreurs d'arrondis. En effet, quelle est la bonne extension à cinq chiffres de 1234×10^1 : 12340, 12345 ou 12350 ?

Le potentiel d'amélioration des performances obtenues par des architectures pipe-linées est limité par la taille des opérations élémentaires que l'on peut exécuter en une seule période d'horloge, comme l'addition de deux entiers signés. Il ne servirait évidemment à rien de découper davantage. On trouvera donc simultanément du parallélisme temporel et spatial dans les calculateurs scientifiques présentant des architectures pipe-lines, c'est-à-dire tout simplement des unités pipe-linées multiples.

1.1.3 Parallélisme et mémoire

La vision de la performance basée uniquement sur la vitesse de réalisation des opérations arithmétiques oublie un ingrédient essentiel du calcul : les données.

Dans un code de calcul scientifique, la part la plus importante des calculs réside en général dans la phase de résolution des problèmes discrétisés, qui nécessite des opérations algébriques portant sur des tableaux à une ou plusieurs dimensions. Une opération emblématique de ce type de calcul est la combinaison linéaire de vecteurs :

for $i = 1$ **to** n
  $y(i) = y(i) + \alpha \star x(i)$
end for

À chaque itération de cette boucle, il faut, pour réaliser une addition et une multiplication, récupérer une donnée $x(i)$ en mémoire, puis une donnée $y(i)$, et enfin remettre à jour le résultat $y(i)$. La donnée α étant la même à chaque itération, elle pourra être conservée dans les cases mémoire tampon internes du processeur, les registres, tout au long de l'exécution de la boucle. Finalement, il faut trois accès à la mémoire, deux en lecture et un en écriture, pour deux opérations arithmétiques. Il ne sert donc à rien de multiplier les unités arithmétiques ou les processeurs, si l'on n'augmente pas en même temps le débit de la mémoire. Or, si l'on veut calculer plus vite, c'est pour pouvoir traiter des modèles avec plus de paramètres. On voudrait donc que la mémoire soit simultanément de grande taille, pour pouvoir contenir toutes les données, et capable d'avoir un débit élevé, afin d'alimenter toutes les unités fonctionnelles arithmétiques disponibles. Pour ce faire, il faudrait que la mémoire fonctionne à une cadence beaucoup plus élevée que les processeurs, ce qui est évidemment illusoire : la mémoire utilise la même technologie de semi-conducteurs.

Le point le plus important pour la réalisation de calculateurs scientifiques à haute performance est donc en réalité l'architecture de la mémoire.

1.2 Architecture mémoire

Comme nous venons de le voir, la puissance calcul obtenue est issue pour une grande part de la duplication des unités fonctionnelles. Ces unités fonctionnelles travaillent sur des données stockées en mémoire. Une question qui se pose naturelle-

ment est de savoir comment alimenter ces unités fonctionnelles afin d'obtenir les meilleures performances possibles.

1.2.1 Mémoire multi-banc entrelacée

Pour augmenter simultanément la taille et le débit de la mémoire, la solution qui s'impose consiste à dupliquer les unités mémoire. Ce faisant, la taille augmente évidemment, néanmoins le délai d'accès à une donnée unitaire est toujours le même. Pour que le débit augmente globalement, dans le cas de l'accès à une série de données d'un tableau, il faut que les différentes unités mémoire fonctionnent simultanément, en parallèle. Les différents éléments du tableau, qui occupent des adresses successives en mémoire, doivent donc être répartis dans les différentes unités mémoire, qualifiées de "bancs". La mémoire est alors dite "entrelacée".

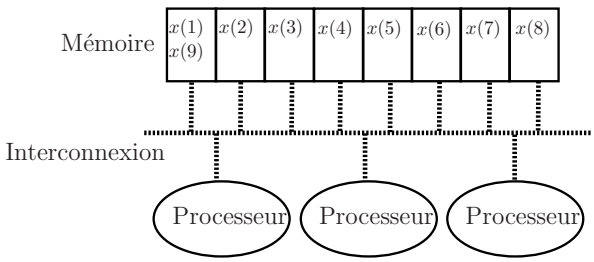

FIGURE 1.3 – Mémoire multi-banc entrelacée.

Supposons que le tableau x est réparti sur huit bancs comme représenté à la figure 1.3. Si le temps d'accès à un banc est de huit cycles d'horloge, alors, après une phase d'initialisation, la mémoire multi-banc entrelacée est capable de fournir une nouvelle valeur $x(i)$ à chaque cycle, puisque chaque banc n'est sollicité que tous les huit cycles.

Dans la réalité, le temps d'accès à un banc est plutôt de l'ordre de plusieurs dizaines de cycles. Pour alimenter de façon satisfaisante quelques ou quelques dizaines de processeurs disposant chacun de plusieurs jeux d'unités arithmétiques, il faut de quelques centaines à quelques milliers de bancs. Le contrôleur de la mémoire et le réseau d'interconnexion entre tous les bancs et tous les processeurs sont très complexes et donc très coûteux à réaliser.

Cette solution ne se trouve que dans des machines spécialisées dans le calcul scientifique, appelées super-calculateurs "vectoriels/parallèles". Le terme vectoriel indique que ces machines disposent, afin de faciliter la gestion du système et d'améliorer les performances, d'un jeu d'instructions portant non pas sur une donnée mais sur une série de données dans un tableau, qualifiée de *vecteur*. Pour stocker ces vec-

teurs, les processeurs disposent généralement de *registres vectoriels*, capables justement de stocker temporairement ces vecteurs.

Cette solution n'est pas *extensible*, dans ce sens que les performances du système n'augmentent pas linéairement avec le nombre de processeurs. En effet, pour mettre en œuvre efficacement plus de processeurs, il faut augmenter le débit mémoire, ce qui nécessite d'accroître le nombre de bancs. Sinon, les processeurs ne seront pas alimentés correctement et la vitesse de calcul totale n'augmentera pas comme le nombre de processeurs. Par ailleurs, si le nombre de processeurs et le nombre de bancs sont multipliés par un facteur p, alors la complexité de l'interconnexion entre la mémoire et les processeurs augmente par un facteur p^2. À un instant donné, l'état de la technologie impose donc une limite à la taille maximale des systèmes réalisables.

1.2.2 Mémoire hiérarchisée

Il est tout à fait possible de réaliser des unités mémoire à temps d'accès court, mais de capacité réduite. En particulier, l'augmentation du nombre et de la densité des circuits permet de disposer de mémoire sur la même puce que le processeur. Cette mémoire peut avoir un temps d'accès d'un seul cycle, mais en revanche elle est de taille limitée. Par ailleurs, le délai unitaire d'accès à une mémoire de grande capacité peut être réduit par la mise en place de procédures d'accès par blocs de données contiguës.

Entre la mémoire de grande taille et le processeur, on va donc trouver une mémoire rapide, appelée "cache", qui sert au stockage temporaire des données utilisées par le processeur. De manière à optimiser le débit de transfert entre la mémoire et le cache, ceux-ci vont s'effectuer par blocs de petite taille, appelés "lignes". Les lignes

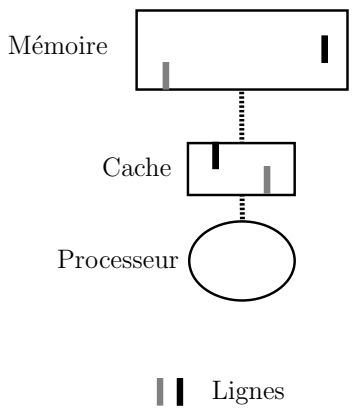

FIGURE 1.4 – Mémoire cache.

présentes dans le cache ne sont que des copies temporaires des lignes mémoire. Le

système gère la liste des lignes présentes dans le cache. Lorsque le processeur a besoin d'une donnée, deux possibilités se présentent :

- la ligne contenant la donnée est déjà dans le cache ; dans ce cas, favorable, le temps d'accès à la donnée sera celui du cache ;
- la ligne n'est pas dans le cache ; il faut la copier depuis la mémoire ; mais, pour ce faire, il faut libérer de la place dans le cache et donc renvoyer en mémoire une des lignes, de préférence celle dont la durée d'inactivité est la plus longue et qui est donc supposée être la moins utile ; la figure 1.5 illustre ce mécanisme.

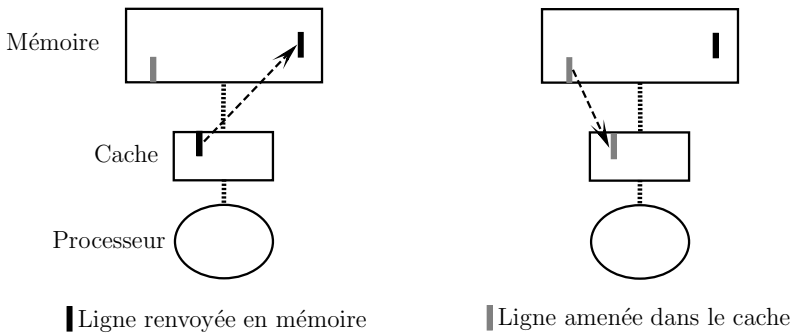

FIGURE 1.5 – Gestion de la mémoire cache.

Avec ce type de fonctionnement apparaît une notion nouvelle : le délai d'accès à la mémoire devient non uniforme. L'organisation des structures de données dans le code et de leurs schémas d'utilisation va fortement conditionner le bon fonctionnement du système mémoire et, par là, le niveau de performance.

Tout d'abord, le système repose sur des mécanismes de transfert par bloc (ou ligne) des données de la mémoire vers le cache, sensés améliorer le débit de celle-ci. Pour chaque accès à une donnée en mémoire, toute la ligne qui la contient est recopiée dans le cache, avant que la donnée ne soit enfin transmise au processeur. Si seule cette donnée est utilisée par la suite, il est évident que la procédure coûte plus cher qu'un accès direct unitaire à la mémoire. En revanche, si les autres données de la ligne sont utilisées par le processeur, dans la même instruction ou dans les instructions immédiatement suivantes, le mécanisme va s'avérer bénéfique. Il faut donc favoriser les accès à des données contiguës en mémoire, c'est-à-dire la "localisation spatiale" des données.

De plus, si le processeur utilise plusieurs fois de suite dans un bref délai des données d'une même ligne, celle-ci va rester dans le cache et les accès successifs aux données seront rapides. Il faut donc tâcher de regrouper dans le temps les accès successifs aux mêmes données, c'est-à-dire favoriser la "localisation temporelle" des données.