Une approche expérimentale pour l'algorithmique parallèle sur grappes et grilles de calcul

Christophe Cérin
Université de Picardie Jules Verne,
LaRIA, Bat CURI, 5 rue du moulin neuf, 80000
AMIENS - France
cerin@laria.u-picardie.fr

Compilation du 4 décembre 2002

Table des matières

Première partie

Introduction

CHAPITRE 1

Calcul parallèle intensif

NOS RECHERCHES portent sur quatre thèmes : l'architecture des machines multiflots, l'algorithmique sur mémoire commune, l'algorithmique et la programmation de grappes (ou réseaux de stations de travail) et les grilles de calcul. Les travaux de recherche présentés portent sur l'algorithmique parallèle expérimentale principalement sur réseaux de stations de travail et sur grilles de calcul. Le travail en matière d'architecture des machines ainsi qu'en algorithmique parallèle en mémoire commune est présenté dans un degré de détail moindre vis à vis du travail effectué sur les grappes et les grilles.

Notre projet de recherche part d'un double constat :

- Le besoin croissant d'applications manipulant de grandes masses de données dans le domaine du calcul scientifique et technique (calcul des structures, simulation, conception assistée par ordinateur), médical (imagerie, génomique), ou commercial (fouille de données),
- L'émergence d'un nouveau type d'architecture bon marché basé sur des PC standards. Ces PC sont connectés en particulier à Internet.

De manière générale, les recherches en parallélisme se sont concentrées sur les aspects calcul et communication : les nouvelles architectures parallèles sont aujourd'hui capables d'atteindre des puissances de calcul de l'ordre du Téraflops (mille milliard d'opérations par seconde). Un point crucial pour exploiter pleinement ces machines est de pouvoir traiter des masses de données qui se mesurent en Gigaoctets, voire en Téraoctets.

Ces masses d'information, que l'on rencontre souvent dans les applications scientifiques (simulation par exemple), financières ou commerciales (fouille) sont largement supérieures à la mémoire centrale disponible sur les machines parallèles et obligent à utiliser les disques pour les accueillir. Les problèmes induits concernent alors l'organisation et l'accès aux données

Les réseaux de stations ou *clusters* sont des architectures parallèles qui ont émergé pendant les années 90. Ils offrent une alternative raisonnable en terme de coûts aux super-calculateurs. Le nombre de conférences internationales de haut niveau complètement dédiées ou offrant des sessions autour du thème montre que ces architectures ont aujourd'hui trouvé toute leurs places dans la communauté scientifique comme objet d'étude. La notion de grille est apparue vers la fin des années 90 pour rassembler dans un même dispositif de calcul et de stockage des machines interconnectées par Internet. La grille constitue l'architecture sur laquelle nous travaillons principalement à ce jour.

Notre démarche est à double sens. Elle se veut pragmatique et expérimentale dans le sens où les expérimentations se font « à grandeur réelle » (ce ne sont pas des simulations) avec la (dernière) technologie qu'il faut maîtriser si l'on veut tirer vers le haut les performances de l'architecture afin d'en isoler les points forts.

Elle se veut plus théorique quant il s'agit de développer des algorithmes optimaux sur un type d'architecture et qu'il faut assurer des bornes sur le temps d'exécution ou sur l'équilibrage des charges. On ne peut pas raisonnablement penser que tous nos maux seront résolus par une technologie plus performante d'année en année. Bien au contraire, le problème en lui même doit être bien saisi et formulé, certes, en termes pas trop éloignés d'une certaine réalité matérielle.

Notre projet scientifique a pour objectif d'étudier et de définir les concepts et outils nécessaires pour résoudre efficacement les problèmes qui gèrent de grandes masses de données sur les architectures de grappes et de grilles.

Dans ce projet nous étudions les aspects architecturaux des processeurs et des machines (architecture et évaluation des processeurs, communication, gestion d'entrées/-sorties parallèles), les outils (bibliothèques de communication) et les algorithmes de calcul. Les applications visées sont principalement les problèmes de calcul numérique.

Nous affirmons une démarche intégratrice qui vise à dégager des techniques d'usage général pour traiter aussi bien un problème en mémoire centrale que sur disque. Notre démarche établi un plan d'intégration. Premièrement il y a comparaison de l'existant afin de mettre en évidence des conflits par rapport à des objectifs assignés. Deuxièmement il y a une mise en conformité qui permet la résolution des conflits. Troisièmement nous fusionnons les schémas que nous avons estimés les plus représentatifs. Enfin, nous restructurons : c'est l'amélioration du schéma global

Pour illuster nos idées et mesurer nos apports nous avons choisi un problème significatif : le tri. Notre démarche d'intégration permet d'obtenir un schéma global de tri qui fonctionne bien dans la pratique et qui a de très bonnes propriétés aussi bien pour le cas en mémoire que pour le cas sur disques.

1. Thèmatique du tri

Pour le calcul scientifique haute performance, l'intérêt du tri tient dans le fait que les accès à la mémoire sont imprédictibles au contraire de beaucoup de problèmes d'algèbre linéaire comme le produit de matrice, la factorisation... ce qui en fait un problème difficile à paralléliser.

Le tri est une application que l'on rencontre dans de nombreux outils existants. Par exemple, `Fastlink` [CIS93] est un outil développé depuis 1993 pour l'analyse statistique des gènes et pour dériver une approximation de « l'endroit » où le gène

est défaillant. `Fastlink` a été utilisé comme outils dans plus de 400 publications scientifiques.

Le code est disponible dans le domaine public[1]. Nous y trouvons effectivement un tri... qui est implémenté avec un tri bulle. Ce n'est pas à priori le meilleur algorithme de tri mais tout dépend de la taille du sous-problème à traiter !

Que ce soit en mémoire ou sur disque, si l'on veut obtenir des performances il faut s'intéresser à la *hiérarchie mémoire*. Pour le tri séquentiel en mémoire nous nous sommes intéressés par exemple à l'impact du cache de premier niveau. Pour le tri externe (sur disque) nous nous sommes intéressés au tri parallèle en milieu hétérogène. C'est la première fois à notre connaissance que le problème est abordé dans ce contexte architectural.

Le problème du tri en parallèle a été étudié pour fixer une technique générale qui est *l'échantillonnage*. Cette idée consiste à choisir des valeurs dans l'entrée que l'on appelle des *pivots* de sorte qu'ils la partitionnent en groupes de tailles approximativement égales. Elle est déclinée et validée dans différents contextes de hiérarchie mémoire : sur disques ou en mémoire et/ou dans une grappe homogène de processeurs ou dans une grappe hétérogène. Pour chaque architecture nous donnons des bornes théoriques à l'équilibrage des charges des processeurs. En pratique les mesures effectuées confirment un équilibrage qui est seulement éloigné de 10% (environ et dans le pire cas) de l'optimal.

Le problème du *tri parallèle d'entiers* est un problème à tiroirs parce qu'il nous fait rebondir vers de nombreuses situations pour lesquels des liens existent. Par exemple la *fouille de données* pose le problème de l'exploitation intelligente de grandes masses de connaissances et d'informations accessibles sur ordinateurs, souvent dans des sites géographiques différents interconnectés par Internet. Nous nous intéressons à la partie << collecte >> d'informations : le tri. Le tri en parallèle est un problème important pour les systèmes de gestion de bases de données [SK96] ou encore dans les moteurs de recherche sur Internet.

Pour notre part, concernant les architectures de grappes nous avons apporté les contributions suivantes :

Tris hétérogènes: certains processeurs vont plus vite que les autres. Ce dernier point constitue sans doute la part la plus novatrice de notre travail sur le tri parallèle. Il s'agit à notre connaissance de la première fois où l'on traite ce problème et que l'on y apporte des solutions effectives. Le travail peut se diviser en deux :

(1) *Tri hétérogène sur disque* : voir la publication [Cér02] réalisé pour la conférence IPDPS'2002,

1. `http://www.ncbi.nlm.nih.gov/CBBresearch/Schaffer/fastlink.html`

(2) *Tri hétérogène en mémoire* : voir [CG00e, CG00f] pour deux techniques distinctes et publiées respectivement pour Cluster'2000 et HiPC'2000 ; Voir aussi [CG02] pour un article de synthèse ;

Tris avec des distributions de données particulières : par exemple une distribution avec << beaucoup >> de doublons [CG99b] ; Voir aussi les références [CG01, CG00d] qui concernent l'évaluation d'une grappe au moyen du tri.

Chaque idée, chaque algorithme présenté ici a été implanté, testé selon différentes entrées sur différentes versions de réseaux de stations. L'ensemble des codes que nous avons développés peut s'obtenir sur :

`http://www.laria.u-picardie.fr/~cerin/=paladin` .

Nous ouvrons ainsi des pistes pour implémenter efficacement des bases de données qui fonctionneraient sur une *machine distribuée hétérogène*. En effet, il est connu que l'opération de *jointure* par exemple, peut s'implémenter par un tri.

La problématique des disques est traitée de manière intensive dans ce rapport et sous différents angles. En effet, le travail collaboratif, les bases de données, le calcul intensif (génomique, résolution matricielle...), la fouille de données utilisent de grandes masses d'information, souvent plusieurs téra-octets. Ces thématiques revêtent une importance primordiale pour le futur où n'importe quel poste de travail sera connecté à Internet et il offrira de fait des ressources disques par exemple.

Notre positionnement scientifique sur le thème des entrées sorties (parallèles) est aussi dû aux réalités suivantes : depuis plus de 5 ans on observe une croissance de l'ordre de 60 à 80% par an de la capacité des disques grands publics alors que les temps d'accès n'augmentent que de 10% par an ! Comme la vitesse des processeurs double tous les 18 mois, l'écart entre les temps d'accès aux registres du processeur vis à vis du temps d'accès à un disque ne fait que grandir au fil des ans. Le temps d'accès aux disques demeure plus que jamais un enjeu majeur. Négliger les entrées sorties est une erreur fondamentale.

2. Thématique architecture

La conception des processeurs a connu de profondes évolutions au cours de la dernière décennie. Ces évolutions sont dûes en partie à des facteurs technologiques : on est maintenant capable d'intégrer de l'ordre de 50 millions de transistors sur quelques millimètres carrés de silicium. C'est considérable. Soit on utilise ces transistors, pour intégrer au sein du circuit, de la décompression vidéo, une carte son ; soit on les utilise pour faire du parallélisme.

En matière d'architecture des machines, nous nous sommes intéressés dans [CP98] aux *machines multiflots* et au concept de << multi-threading >> . Il s'agit de machines

qui autorisent à changer de contexte de processus très rapidement grâce à un matériel adapté [I+94], [GGB93]. Assez souvent dans la pratique on arrive à masquer les temps de changement de contexte et/ou à jouer sur le nombre de processus actifs simultanément.

Par exemple, la machine MTA de Cray est capable de changer de contexte tous les cycles processeur et en un seul cycle. Elle gère jusqu'à 128 contextes de processus grâce au matériel. Elle n'a pas de cache de données. C'est surprenant ! La latence mémoire est masquée par le changement de contexte. Dans la pratique il est nécessaire d'avoir au moins 25 processus pour que le recouvrement mémoire / calcul soit total. Le processeur MTA peut gérer jusqu'à 8 accès mémoire concurrents par contexte.

Il est très difficile de capter les performances attendues de ce genre de machines. On manque à la fois de modèles et d'outils. Peu de travail a finalement été accompli pendant la dernière décennie sur le thème des performances de ces machines. Cependant, ces machines sont potentionnelement très intéressantes pour apporter des performances de calcul. Ce type d'architecture est souvent cité pour sa capacité à atteindre potentiellement le Pétaflop.

Nous nous sommes intéressés aux performances des machines multiflots mais contrairement aux études probabilistes basées sur les files d'attentes [OH97, Aga89, SBC91, Squ94] et qui produisent des << valeurs moyennes >>, nous avons proposé un modèle algébrique au sens des automates. Mesurer les performances consiste à trouver des << mots parallèles >> parmi l'ensemble des mots possibles décrivant toutes les exécutions de la machine multiflot.

Les mots sont constitués soit de lettres matérialisant que le processeur calcule, soit de lettres matérialisant qu'il est en attente d'une opération mémoire, soit qu'il effectue un changement de contexte, soit qu'il est inactif.

Nous avons montré comment calculer effectivement les performances << maximale et minimale >> que l'on pouvait théoriquement atteindre pour une machine multiflot. Pour un mot on calcule le rapport entre le nombre de fois où l'on a une opération de calcul et la longueur du mot. Cette notion est étendue à un langage de mots.

C'est la première fois que l'on décrit les performances de machines par des langages et que l'on donne des bornes à l'efficacité de ce genre de machines et pas des estimations de comportements en moyenne.

En matière d'architecture nous nous sommes également intéressé à l'impact des caches dans l'obtention de performances. C'est une problématique de hiérarchie mémoire. Nous avons étudié le tri séquentiel et nous avons proposé un algorithme qui bat en terme de temps d'exécution les algorithmes connus de tri qui tiennent compte du cache.

Nos mesures expérimentales sont effectuées en observant les compteurs de performances des processeurs. C'est la première fois que l'on mesure les performances des

algorithmes de tri de cette façon : les précédentes études sont faites avec des simulateurs. Mais comme la conception des circuits s'est considérablement complexifiée avec par exemple l'apparition du concept d'exécution dans le désordre, un simulateur ne peut donner, à l'heure actuelle, qu'une approximation grossière des facteurs de performance.

Nos observations sur sept algorithmes de tri permettent de dégager la conjecture suivante : « Soient X et Y deux programmes répondant à la même spécification. Si le programme X a une IPC (nombre d'Instructions Par Cycle) au moins deux fois plus grand que Y mais qu'il a un nombre de défauts de cache L1 et un nombre d'instructions exécutées deux fois plus important que Y, alors le programme X a quand même une exécution en temps meilleure que Y ».

Cette conjecture ouvre des pistes de travail afin de construire des algorithmes qui exhiberont suffisamment d'instructions indépendantes pour alimenter les différentes unités d'exécution des processeurs. Les algorithmes actuels (QuickSort...) ne les exploitent pas de manière efficace.

3. Thématique du calcul en mémoire commune

Le champs de recherche du calcul en mémoire commune a été également investi car il correspond à une thématique traitée à Amiens depuis longtemps.

Le parallélisme a souvent été abordé avec la vision que tous les processeurs peuvent accéder à la mémoire qui est partagée par tous. Cette vision est celle du modèle PRAM (Parallel Random Access Memory). Le modèle à été étudié intensément et un grand nombre d'algorithmes ont été développés et synthétisés dans de nombreux ouvrages [Jáj92], [Ble93], [Lei92] [Akl97] et depuis de nombreuses années.

Nous présentons dans ce rapport nos résultats obtenus en algorithmique parallèle sur des séquences (vecteurs) d'entiers et de chaînes de caractères. Il s'agit principalement des problèmes de la mise en correspondance de parenthèses, et du segment de somme maximale. Ces problèmes sont traités selon différents modèles de calcul.

La mise en correspondance de parenthèses est un problème très important car il intervient dans l'évaluation en parallèle des expressions arithmétiques.

Nous avons donné un algorithme parallèle [BC97b] de mise en correspondance de parenthèses qui égale la complexité en temps du meilleur algorithme connu à ce jour mais qui offre une présentation favorable à une implantation, par exemple avec les librairies BSP (Bulk Synchronous Parallel Model).

Le problème du segment de plus grande somme [BC97a] est quant à lui abordé dans le cadre du modèle BSR (Broadcast with Selective Reduction) que l'on peut voir, en première approximation comme un modèle à mémoire commune plus une instruction de diffusion qui est supposée s'effectuer en temps constant. Nous donnons

différentes formulations à la solution de ce problème. Ces formulations utilisent moins d'opérations de diffusion que les formulations précédemment connues.

À travers l'exemple du segment de somme maximale, nous en retirons une méthodologie de conception de programmes BSR.

Le travail autour de PRAM a donné lieu à la thèse de Laurent Bergogne[2] et à deux publications : [BC97b, BC97a]. Les deux points précédents ont donné lieu également à l'encadrement de cinq stages du DEA d'informatique d'Amiens entre 1994 et 1999.

Nous pouvons rajouter que les résultats concernant PRAM et BSR sont très fortement résumés dans ce rapport afin de nous concentrer sur le thème du tri et de ses perspectives ainsi que sur les entrées sorties performantes.

4. Contexte technologique

Dans la communauté scientifique, la demande en puissance de calcul et en taille mémoire va grandissante. Par exemple, une << petite >> expérience de simulation numérique effectuée en parallèle peut prendre 2000 heures de calcul sur 20 machines. Comme la ressource de calcul est partagée et que des plages journalières de calcul sont attribuées à différents utilisateurs pour pouvoir lancer plusieurs expériences, on est obligé de stocker sur disques des calculs intermédiaires puis à les recharger en mémoire quand la machine est de nouveau attribuée à l'expérience.

On constate depuis plus de trois ans un recul des machines massivement parallèles au profit de plate-formes distribuées et hétérogènes à base de grappes de stations de travail que l'on appelle aussi des clusters. On parle à présent de grille de calcul (computational grid) dans laquelle on accède à la puissance de calcul comme on accède à l'électricité. Par ailleurs, on constate que le développement d'applications se fait de plus en plus grâce à des composants logiciels. Dans le cadre du calcul scientifique, l'utilisation de bibliothèques de haut niveau est maintenant courante et des versions parallèles (Netsolve, Scilab, Blas...) sont disponibles pour la plupart des architectures. En revanche l'accès aux ressources distribuées à grande échelle pose des difficultés. Une solution intéressante consiste à utiliser des serveurs fournissant des services à des clients qui s'y connectent. Dans la plus grande généralité, ces services ne sont pas limités uniquement au calcul scientifique et de nombreuses applications peuvent en bénéficier.

Le paradigme qui vient d'être décrit afin de répondre à un service s'appelle le paradigme ASP (Application Service Provider). Le paradigme ASP est illustré de manière

2. Laurent Bergogne, après avoir occupé un important poste de responsable des systèmes au sein de l'agence pour le développement des nouvelles technologies de l'information et de le communication en Picardie et s'être occupé de Renater à l'université, a un poste au conseil régional de Picardie. Il peut être contacté à l'adresse `lbergogne@cr-picardie.fr`

plus fine aux Figures 1 et 2. Un utilisateur veut réaliser un calcul numérique. L'utilisateur adresse une requête a un tiers (un agent) afin de réaliser l'opération `OP(A,B,C)`. L'agent a la charge de déterminer le << meilleur >> serveur qui peut répondre à la requête de calcul. Il s'agit ici du serveur 2. Sur la Figure 2, l'utilisateur envoie directement l'opération et ses opérandes (A,B,C) au serveur 2 ; puis le résultat (C) est retourné directement à l'utilisateur.

Nos objets d'étude sont les services de calcul que doivent rendre les serveurs. Nous nous intéressons plus particulièrement aux serveurs de type << clusters >> qui sont construits à partir de matériel commun (des PC grand public, interconnectés par une technologie grand public de type Ethernet). Le cluster est supposé dédié. Nous avons étudié plus particulièrement le service de tri externe lorsque le cluster est hétérogène mais aussi le cas du tri parallèle en mémoire en hétérogène.

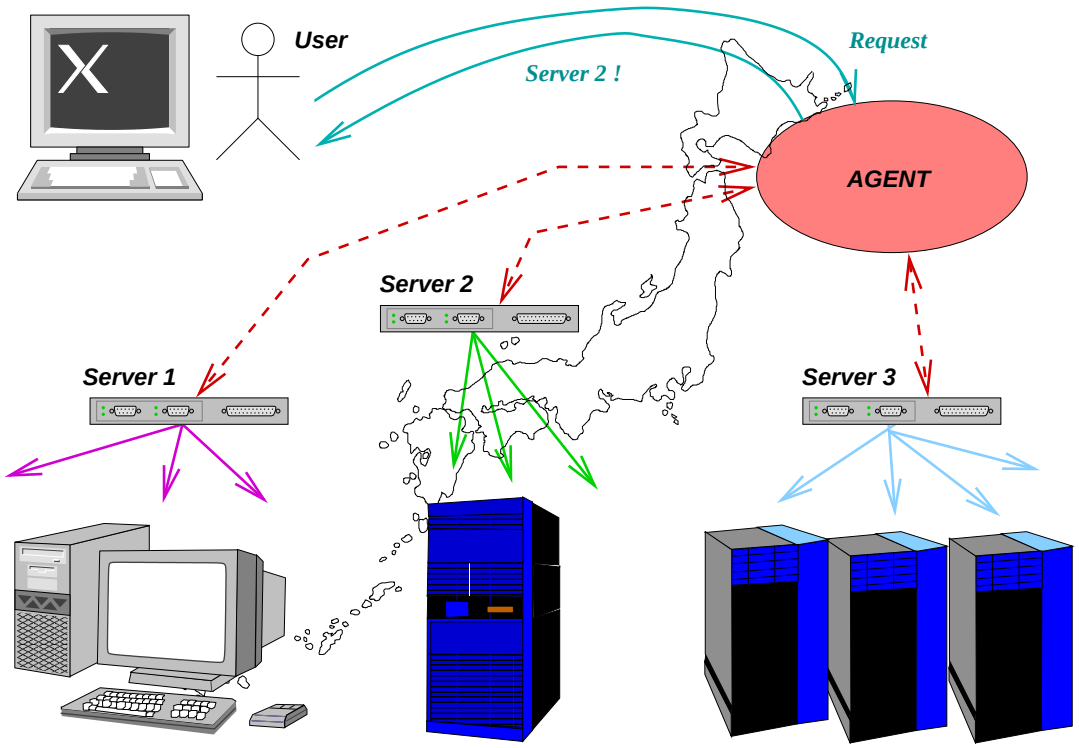

FIG. 1: *Le paradigme ASP (première partie).*

5. Présentation de l'architecture des clusters

Sur la fin des années 90, les *réseaux de stations de travail* ont émergés comme une variété d'architecture distribuée partageant "more than nothing" comme on le dit fréquemment. Le partage peut consister à partager la configuration des machines ou de manière plus technique, l'espace d'adressage des fichiers, de la mémoire afin de globaliser les ressources physiques des différentes machines.

L'architecture cible sur laquelle nos algorithmes ont été implantés est une architecture de *réseaux de stations de travail*. Cette architecture est une alternative raisonnable [Buy99a, Buy99b] aux machines parallèles dédiées. Il est maintenant admis que les principaux facteurs ayant conduit à cette acceptation sont liés à des facteurs économiques : le rapport entre les coûts de la machine et les performances sont très avantageux lorsqu'on peut compter sur des composants produits en masse.

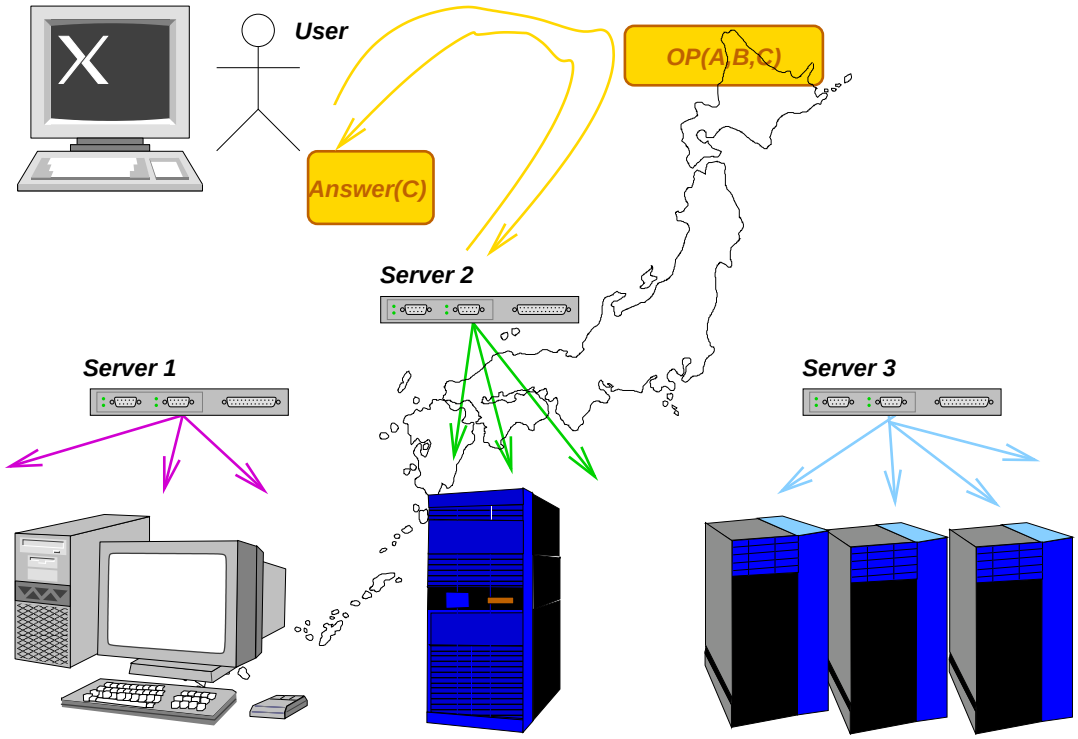

FIG. 2: *Le paradigme ASP (deuxième partie).*

Une grappe de stations de travail (ou *cluster*) telle que nous l'avons expérimentée est composé :

- de composants standards (microprocesseurs, mémoire vive DRAM, disques...) tels qu'on les achète chez de grands distributeurs ;
- de cartes mère grand public (on trouve aussi des cartes bi-processeurs Intel Celeron pour 150 €) ;
- de réseaux d'interconnexion basés sur un standard éprouvé comme Ethernet décliné en Fast-Ethernet (100Mbits/s) et bientôt Gigabit-Ethernet (1Gbits/s) ou des réseaux dédiés comme Myrinet qui offre des communications point à point de plus de 1Gbits/s mesuré avec une latence de l'ordre de 10µs. Ces performances sont du même ordre que celles des réseaux des calculateurs parallèles comme l'IBM SP... alors que le coût d'une carte Myrinet est d'environ 1000USD (seulement) ;
- de bibliothèques de programmation parallèle, généralement par envois de messages comme MPI ou BspLib [MR94] ou BUP7 [BJvOR99] qui implémentent le modèle BSP [Val90a].

Outre l'aspect économique avantageux, les réseaux de stations permettent le *passage à l'échelle* : l'extension d'un système est aisément réalisée et bien souvent, là aussi, à un coût très raisonnable. En effet, dans une architecture de type client/serveur, quand le nombre de clients augmente, le serveur est le goulet d'étranglement puisqu'il est par définition le seul à rendre un service. De même dans une architecture à mémoire partagée, le trafic sur le bus mémoire devient vite un goulet d'étranglement, généralement à partir d'une dizaine de nœuds. Dans une architecture de type grappe, les données sont réparties sur l'ensemble des machines constituant la grappe ce qui réduit potentiellement les problèmes du goulet d'étranglement : on peut par exemple privilégier des interactions deux à deux sans passer par l'entremise d'un serveur.

Les architectures de type grappe ont des répercussions certaines sur la façon de concevoir les algorithmes. En effet, les réseaux rapides comme Myrinet ont des performances entre celles de la mémoire (en étant très proches puisque le débit d'une mémoire est de l'ordre du gigabit par seconde alors que la latence est de l'ordre de la microseconde) et celles des disques (le débit d'un disque de PC est de l'ordre du mégabit par seconde alors que le temps moyen d'accès est de l'ordre de la milliseconde). Les disques distants deviennent alors accessibles au coût d'un disque local ; la donnée dans une mémoire distante est accessible plus rapidement que si elle résidait sur le disque local[3]. Le fait de bâtir une stratégie de résolution en se souciant de la place où se trouve une donnée en mémoire n'aurait plus de raison d'être ?

3. Voir des mesures sur les pages de Myricom : `http://www.myri.com/`

6. Contraintes architecturales

Nous nous plaçons en premier lieu dans le contexte où les architectures matérielles des processeurs sont compatibles. Les processeurs sont reliés par un seul type de réseau de communication. Le cluster est complétement dédié à l'application.

La poussée des clusters[4], comme nous l'avons précédemment dit, a été indéniable les cinq dernières années. La plupart des clusters existants[5] sont homogènes dans le sens où tous les nœuds du cluster sont identiques. Par exemple, à l'automne 2002, le classement des << meilleurs >> clusters donne en première position une machine située au Japon et fabirquée par NEC. Il s'agit du projet "Earth-Simulator" qui utilise 5120 processeurs.

Nous nous sommes intéressés ensuite au cas particulier des *clusters non homogènes* qui sont définis ici comme des clusters habituels (réseau de communication unique, même système d'exploitation installé, même processeur, même montant de place mémoire et de place disque) mais les processeurs peuvent être à différentes vitesses. Il s'agit donc d'un premier niveau d'hétérogénéité. De manière générale, dans un système parallèle la notion d'hétérogénéité touche potentiellement toutes les composantes du système :

- Pour le réseau, la notion d'hétérogénéité concerne les possibilités d'utiliser différents média de communication, différentes méthodes de signalisation, différentes interfaces et protocoles, différents types de bus pour construire des réseaux de réseaux ;
- Pour la mémoire, la notion d'hétérogénéité concerne les possibilités d'utiliser la mémoire locale ainsi que la mémoire distante, de gérer différents niveaux de cache avec différentes politiques de gestion ;
- Pour les processeurs, la notion d'hétérogénéité concerne les possibilités d'utiliser des processeurs de différents fabricants, des processeurs à différentes vitesses, des processeurs d'architectures internes différentes (RISC (reduced instruction set computer), VLIW (Very Long Instruction Width processor), superscalaire, multithreadés)
- Pour les logiciels tournant sur les systèmes, la notion d'hétérogénéité concerne les possibilités d'utiliser plusieurs systèmes d'exploitation ou plusieurs binaires d'un même programme ;
- Pour les disques, la notion d'hétérogénéité concerne les possibilités d'utiliser plusieurs systèmes de fichiers, différents média d'archivage (disques, floppy, cartouche), différents protocoles (IDE ATA, SCSI, PCMCIA).

4. Voir IEEE task force on cluster computing sur `http://www.ieeetfcc.org`

5. Voir un classement des performances sur `http://clusters.top500.org/`. Le premier cluster français est le icluster de l'IMAG (`http://icluster.imag.fr`)

Les problèmes parallèles en milieu hétérogène tel que nous venons de le définir, constituent des problèmes intéressants et particulièrement utiles si l'utilisateur d'un cluster ne peut pas changer d'un seul coup tous les processeurs de son cluster mais doit composer avec plusieurs versions d'un même processeur à différentes vitesses.

Il n'existe pas beaucoup de littérature (à notre connaissance) pour cette classe d'architecture, tous les articles sur le tri sur clusters traitent du cas homogène uniquement. Notre travail est donc novateur. Quant au problème de mesurer effectivement la vitesse relative des processeurs entre eux, nous supposons l'existence de techniques précises pour le réaliser. Par exemple, pour mesurer les performances d'un nøeud on peut utiliser des tests. Voir à ce sujet la liste des liens Internet en Annexe à ce rapport.

Nous verrons également que dans le cas du tri sur disque, un cluster hétérogène fera référence à des nœuds avec des performances qui différent par une constante. C'est la première fois à notre connaissance où l'on trie dans ce contexte. Nous prétendons que le schéma que nous avons développé pour trier sur des clusters hétérogènes fonctionne très bien en pratique à la fois pour trier en mémoire et aussi pour trier sur disque. Cette généricité du schéma de calcul fait toute l'originalité de nos études.

7. Notre méthodologie de conception

Puisque les processeurs ne vont pas tous à la même vitesse, les techniques classiques de parallélisation automatique de programmes séquentiels (comme le dépliage de boucles) qui cherchent à exhiber des morceaux de code indépendants que l'on fait exécuter en paralèle ne sont plus applicables. En effet les techniques se concentrent sur les dépendances entre les données alors qu'en milieu hétérogène il est aussi nécessaire de s'intéresser à la quantité de travail effectué dans le temps par les processeurs afin de garantir un équilibrage des charges des processeurs.

Il est aussi bien compris en algorithmique et programmation parallèle que le gain d'un algorithme parallèle vis à vis d'un algorithme séquentiel résolvant le même problème est largement dépendant des stratégies pour réduire les latences mémoire et de communication et aussi des façons dont sont traitées les synchronisations et l'ordonnancement des tâches. À ce titre, la distribution du travail et sa gestion peuvent être effectuées soit dynamiquement [WT98] soit statiquement, soit en mixant les deux stratégies. Les stratégies dynamiques utilisent souvent une approche naïve tel que le schéma « maître/esclave » ou des paradigmes dont le fondement se trouve dans l'idée que « le passé immédiat sert à prédire le futur immédiat ». Comme il est montré dans un récent rapport prospectif de J.F Méhaut et Y. Robert[6], ce genre de stratégie peut conduire dans certains cas à ralentir les calculs d'algèbre linéaire du fait d'une prédiction qui n'est pas garantie.

6. Voir : `http://www.ens-lyon.fr/~yrobert/journals/RR99-36.ps.gz`

Par opposition, les stratégies statiques pré-déterminent une distribution des données et des communications de façon déterministe. Pour des applications comme notre problématique du tri, nous prétendons que les stratégies statiques jouent un rôle central dans l'obtention de bonnes propriétés d'équilibrage de charge et de bons résultats d'exécution. Nous sommes en effet très intéressés par le temps parallèle d'exécution mais aussi par la façon dont les processeurs sont chargés. Pour un problème de taille n avec $p > 1$ processeurs, on cherche à faire en sorte que le travail de chaque processeur ne soit pas supérieur à n/p ce qui procure un gain maximal et l'équilibrage optimal. Dans notre travail, puisque les processeurs ont des vitesses différentes, nous commençons par leur assigner une quantité de données inversement proportionnelle à leur vitesse.

Il s'agit ensuite de faire en sorte que les redistributions de données par les communications ne chargent pas les processeurs avec un nombre de données supérieur à ces quantités initiales, sous peine de surcharger certains processeurs. La difficulté consiste donc à trouver des redistributions qui préservent l'équilibrage initial du travail. Et en plus, il s'agit de faire en sorte que le coût des re-distributions soit aussi faible que possible vis à vis du travail de comparaison qui est l'essence même des algorithmes de tri (tout du moins beaucoup d'entre eux).

Au final les objectifs poursuivis dans ce document autour du tri peuvent se résumer ainsi :

1 : Proposer des schémas pour trier sur des architectures hétérogènes de réseaux de stations dédiées ; les cas en mémoire et avec les disques sont envisagés ;

2 : Étudier plus particulièrement l'équilibrage des charges : il s'agit de borner la quantité de travail que chaque processeur (hétérogène) aura à accomplir ;

3 : Garantir que le tri externe utilise vraiment les disques : pour cela on impose que les algorithmes fassent un usage limité de la mémoire RAM et que le nombre de fichiers intermédiaires soit aussi bas que possible. On cherche donc à construire des algorithmes qui font un usage limité des ressources du système. Ces contraintes sont imposées car pour effectuer des comparatifs il vaut mieux se placer dans des contextes difficiles. De plus, l'usage d'un cluster pré-suppose que les ressources ne sont pas infinies (par rapport à la taille des problèmes traitées) et donc qu'il faut rationaliser leurs usages.

4 : Mettre en perspective les techniques de tri par échantillonnage dans le contexte où le réseaux d'interconnexion est Internet. Ici le tri est vu comme une application permettant de discuter des points à mettre en œuvre en recherche dans les dispositifs de calcul globaux. Le tri n'est pas la seule application envisagée à terme mais elle nous parait suffisante pour présenter dans ce document ce que nous voulons faire dans le futur : l'étude des plate-formes ou grilles de calculs et en particulier les problématiques du stockage.

On montrera également que le tri n'est pas toujours bien utilisé dans la littérature comme test (benchmark) de dispositifs logiciels.

Nos conditions d'expérimentation sont assez strictes. Par exemple, si nous utilisons 10% de la taille disque du PC pour stocker des données alors nous souhaitons utiliser pas plus de 10% de la RAM de ce même PC. Ainsi nous nous plaçons dans le cas d'une utilisation des disques pour calculer : un tampon disque de taille $\ll$ 10% de la taille du disque $\gg$ ne pourra aller se loger que dans un tampon de même taille dans la RAM.

Lorsque le tri s'effectue en chargeant tout le fichier en mémoire, puis en triant en mémoire avant d'écrire sur disque les résultats, ceci correspond plutôt à un cas de figure où l'on utilise 100% de la RAM disponible et 10% du disque. Dans le monde (réel) de calcul global présenté dans le paragraphe de perspectives, nous ne pouvons pas nous permettre de réserver 100% de la RAM d'un utilisateur de PC.

En effet, un de nos objectifs à terme sera de n'utiliser qu'une fraction des ressources disponibles du PC. En effet, dans les perspectives, nous donnerons des pistes pour la réalisation d'un système de calcul global (dont un des services serait le tri) déployé sur Internet avec des nœuds de calcul chez le particulier. L'environnement cible ne sera plus les grappes mais des systèmes ouverts avec d'autres types de contraintes.

L'idée maîtresse à poursuivre est alors de ne pas occuper le processeur lors d'un échange d'un disque à un autre. On compte pour cela utiliser des bibliothèques comme `Read`[2] [CRU02] ou des systèmes où les disques ont une certaine intelligence et peuvent prendre de décision sans le contrôle du processeur.

Les algorithmes développés dans le cadre externe (sur disque) font effectivement un usage limité aux ressources systèmes. Par exemple, la taille mémoire maximale utilisée est de l'ordre de 32K octets et ce qu'elle que soit la taille du problème. De plus le nombre de fichiers intermédiaires utilisés dans nos expérimentations est compris entre 15 et 150 au plus.

8. Grilles de calcul

Le second paradigme qui change notre façon de calculer est, après le paradigme des réseaux de stations, le paradigme du *calcul sur une grille*. La grille doit être vue comme un réseau d'interconnexion grand public, à savoir Internet mais aussi comme le réseau RENATER pour les chercheurs en France. Les grilles sont des environnements logiciels qui autorisent les applications à intégrer, à afficher, à intrumentaliser des ressources qui sont détenues par des organismes ou des individus situés dans différentes zones géographiques.

Les deux paradigmes ont en commun le souci de partager des ressources de calcul dans un but d'économie des moyens mis en œuvre. Les clusters utilisent des machines à faible coût comme composants matériels d'un super-calculateur alors que la « grille » autorise une meilleure utilisation des ressources de calcul, mais aussi de stockage via Internet. Il s'agit de mutualiser l'ensemble des ressources et pas seulement le temps CPU.

Il est généralement reconnu que les domaines d'application des « grilles de calcul » se divisent essentiellement en cinq grandes classes :

(1) le calcul intensif distribué : on utilise les grilles de calcul pour additionner les capacités de plusieurs machines afin de résoudre un problème donné. Ce peut être l'agrégation de plusieurs super-calculateurs ou simplement de stations de travail. Les applications typiques sont la simulation [GG99], que ce soit de situations complexes (manoeuvres militaires par exemple) ou du déroulement précis de processus physiques ;
(2) le calcul à haut débit : on utilise les grilles de calcul pour ordonnancer un grand nombre de tâches, peu ou pas du tout couplées c'est à dire indépendantes, sur des machines inutilisées ;
(3) le calcul à la demande : il permet une utilisation temporaire de ressources dont la possession permanente ne serait pas rentable. Ici, c'est le rapport qualité-prix qui est l'argument principal quant à l'adoption des grilles, plutôt que les performances absolues ;
(4) le traitement intensif de données : on produit de nouvelles informations à partir de données géographiquement distribuées. C'est le cas typique de la physique des particules, des systèmes de prévisions météorologiques, des programmes d'observation terrestre. Le projet européen DataGrid [7] qui a débuté en janvier 2001 est un bon exemple de ce qui est attendu en utilisant des grilles, en particulier pour le stockage des données et le traitement des entrées sorties. Par exemple, en physique des particules, il est prévu à l'horizon 2006 qu'une seule expérience génèrera de l'ordre de 5 Péta octets de données par an !
(5) le calcul collaboratif : il privilégie la mise en place d'interactions entre personnes humaines pour l'exploration conjointe de bases de données, des systèmes de réalité virtuelle.

La construction d'environnements logiciels pour les grilles nécessitent alors de s'intéresser aux thèmes généraux suivants :

– intergiciel pour les grilles ;

7. Voir : `http://marianne.in2p3.fr/datagrid/wp6-fr/` pour la partie concernant la france

- systèmes d'interconnexion et protocoles ;
- systèmes de fichiers, entrées sorties et bases de données ;
- gestion des ressources ;
- ordonnancement et équilibrage des charges ;
- gestion de grandes masses de données distribuées ;
- langages de programmation et modèles ;
- outils ;
- évaluation de performances ;
- applications.

La partie intergiciel s'intéresse à développer une architecture logicielle capable d'ordonnancer les tâches et de gérer les ressources disponibles de mettre en place des outils de supervision de ressources et de services, de gérer les infrastructures locales à chaque site de la grille, de fournir une interface uniforme pour l'accès aux données.

Concernant la gestion de grandes masses de données distribuées, les biologistes par exemple font face à des problèmes cruciaux lorsqu'ils essayent d'accéder aux données. En effet, les bases de données génériques de biologie sont maintenant accessibles au public et rendues disponibles par leurs fournisseurs. Mais d'autres banques avec des informations très spécialisées sont seulement disponibles sur des sites spécifiques (habituellement, ces bases contiennent des résultats d'expériences disponibles sur le site web des laboratoires où elles ont lieu). Comme il n'y a pas de moteur de recherche générique qui pourrait aider le biologiste à trouver l'information sur un objet biologique donné, la recherche d'information se fait un peu au hasard !

De plus, il n'y a pas de format standard pour les bases de données à cause de la façon dont elles sont crées et maintenues (habituellement, les bases de données sont disponibles sous forme de fichiers « à plat » qui entraînent de nombreux parcours séquentiels de fichiers ce qui est coûteux en temps). Par ailleurs, il n'existe pas de standard de langage de requêtes qui pourrait aider le biologiste à réaliser des requêtes.

Le dernier problème majeur rencontré par les biologistes lorsqu'ils désirent accéder aux bases de données est le temps de réponse. Il n'existe que quelques centres de stockage pour l'instant. La communauté des biologistes qui analysent des données génomiques est très large (estimée à 30 000). Cela provoque la congestion des principaux centres de ressources. C'est pourquoi les biologistes préfèrent télécharger localement les bases de données dont ils ont besoin. Mais il se pose alors le problème de la cohérence des informations détenues localement vis à vis de la base.

En raison du format particulier des bases de données spécifiques, une base peut être présente dans différents formats dans un centre de ressources. De plus, les programmes d'analyse peuvent utiliser un format d'entrée spécifique.

En raison de la taille croissance des données, les mises à jour des bases de données de biologies peuvent être très longues. Par exemple, la mise à jour de l'indexation de la base EMBL[8] demande 24 heures.

La synchronisation de la mise à jour des bases dans les centres de ressources principaux est également un problème. Chaque centre collecte plusieurs mises à jour de bases. Régulièrement, ces centres échangent leur mises à jour pour synchroniser leurs banques. Ainsi, selon la base de données et le moment d'accès, un biologiste peut obtenir différents résultats. De plus les objets peuvent être nommés différemment dans les bases de données créant des ambiguïtés pour les biologistes.

Les problèmes rencontrés par les biologistes sont représentatifs des problèmes rencontrés dans d'autres champs disciplinaires comme l'imagerie, le multimédia, la CAO (conception assistée par ordinateur), la logistique.

9. Plan

L'architecte qui conçoit des machines multiflots se pose en particulier la question de savoir si, par exemple, un circuit capable de gérer 128 processus au niveau matériel et qui comporte 8 unités de traitement parallèle offre plus de performance qu'un circuit capable de gérer 64 processus au niveau matériel avec 16 unités de traitement parallèle.

Nous définissons des mesures qui permettent de capter le maximum de parallélisme et le minimum de parallélisme d'un circuit multiflot. En première appoximation, le parallélisme d'un circuit est déterminé par le rapport en le nombre de fois ou le circuit, modélisé par un automate, est dans l'état "running" vis à vis du nombre de fois ou il est dans les autres états et ceci pour toutes les executions possibles avec le circuit. Les mesures sont calculables en utilisant des outils de la théorie des langages formels ce qui constitue une approche radicalement différente des approches précédentes de l'évaluation des machines multiflots qui utilisent la théorie des files d'attentes.

Nos résultats offrent à l'architecte de machines la possibilité d'avoir un premier retour à priori sur les performances qu'il pourra atteindre au mieux et au pire avec sa conception du circuit multiflot.

Nos résultats en matière de gestion de cache montrent que les tris séquentiels actuels (Quicksort...) ne conduisent pas à une utilisation optimale des ressources des processeurs. En particulier, d'après nos mesures, les unités fonctionnelles de calcul sont sous exploitées.

Un modèle permettant de capter à la fois les propriétés du cache et celles des unités fonctionnelles parait un à priori souhaitable pour augmenter les performances des tris

8. Voir les sites `http://www.embl-heidelberg.de` et `http://www.embl-grenoble.fr`

séquentiels sur les processeurs actuels. Des techniques algorithmiques comme la compression de données couplées au respect des propriétés essentielles que sont la localité spaciale et temporelle permettront une utilisation bien meilleur des processeurs dans le futur.

Nos travaux en mémoire commune, en particulier pour le problème de la mise en correspondance de parenthèses, permettent d'envisager que notre algorithme soit implémenté dans un compilateur afin de générer une version parallèle de l'évaluation d'expressions arithmétiques. L'idée ici est aussi d'utiliser au mieux les différentes unités de calcul disponibles sur un processeur.

Le modèle BSR (Broadcast with Selective Reduction) conduit à des spécifications parallèles qui s'expriment en quelques lignes car le modèle le facilite, en particulier grâce à l'opération de diffusion. Aucun circuit BSR n'a été réalisé à ce jour bien que des descriptions << sur papier >> existent. Elles montrent qu'en terme de circuiterie, le modèle BSR s'implémente au même coût qu'une PRAM. Peut être que l'avènement des PIMs (Processor in Memory) qui consiste en de la DRAM avec des capacité de calcul, permettrait la construction d'un circuit à coût raisonnable.

Nos résultats obtenus pour les tris parallèles permettent d'envisager leurs intégrations dans différents secteurs : en base de données car l'opération de jointure peut s'implémenter par un tri, dans des logicielles pour la biologie, pour les problèmes de mises en correspondance (pattern matching et autres). La mise en correspondance de parenthèses est un problème qui intervient dans l'évaluation en parallèle des expressions arithmétiques. Le problème de la somme maximale a été introduit pour répondre à des problèmes de recherche de motifs dans des images.

Les points forts des algorithmes sont qu'ils ont été développés dans un contexte de machines hétérogènes et pour des problèmes dont les données tiennent en mémoire ou au contraire sur disques. Le problème du tri parallèle peut maintenant se traiter efficacement pour une palette d'architectures parallèles plus large qu'auparavent.

L'organisation de ce document est alors la suivante. Dans la partie 2 nous présentons notre travail en matière d'architecture des machines multiflots qui consiste en la définition et au calcul d'une mesure d'efficacité des machines multiflots. Il s'agit aussi de présenter nos études et résultats en matière de gestion des caches pour le tri séquentiel.

Dans la partie 3 nous présentons les résultats obtenus sur le modèle PRAM et une de ses variantes (BSR). Les problèmes traités concernent des problèmes sur des << séquences de >> (mise en correspondance de parenthèses, segment de somme maximale).

Dans la partie 4 nous décrivons notre travail sur le tri et les résultats expérimentaux obtenus pour trier en parallèle et pour les cas homogènes et hétérogènes. Les techniques utilisées sont *l'échantillonnage* (de pivots) [SS92] et le schéma de sur-partitionnement [LS94]. Nous présentons également nos résultats pour le cas du tri sur disque en milieu hétérogène.

Dans la partie 5 nous donnons nos pistes d'études actuelles. Cette partie est divisée de la façon suivante. Le chapitre 1 est une présentation générale des perspectives, le chapitre 2 évoque les évolutions technologiques en matière d'interconnexion de systèmes de disques, le chapitre 3 présente notre implication sur le thème du calcul global, le chapitre 4 évoque plus particulièrement les structures de données adaptées aux problèmes nécessitant des disques et le chapitre 5 adresse l'analyse de l'activité des machines connectées à une grille de calcul. Le chapitre 6 termine le rapport et synthétise nos apports algorithmiques, méthodologiques et met en avant les perspectives de travail.

Deuxième partie

Architecture des processeurs

CHAPITRE 1

Modélisation et bornes des performances des architectures multiflots

PLUSIEURS DIZAINES de millions de transistors peuvent être intégrés sur un unique substrat de silicium. Les fréquences d'horloges internes des microprocesseurs augmentent rapidement (Le Pentium 4 à 2Ghz est disponible en grand public depuis juin 2001). Le fossé entre horloge interne (sur le circuit intégré) et horloge système (sur la carte mère) ne cesse de s'accroître ce qui entraine que les temps d'accès à la mémoire principale (RAM) pénalisent sévèrement les systèmes.

Pour exploiter un grand nombre de transistors, plusieurs formes de parallélisme d'exécution sont mises en œuvre : duplication des CPUs ("multi-processeur on a chip"), duplication de certaines unités fonctionnelles comme les unités arithmétiques et flottantes, partage du pipeline super-scalaire (multiflot simultané)...

Nous nous intéressons au gain potentiel que peut apporter une architecture multiflot. La modélisation et le calcul du gain proprement dit permet d'apprécier à priori (sans avoir à construire la machine) les performances de la machine multiflot. La machine multiflot capable de gérer à un niveau matériel n processus est vue comme un automate, plus précisément comme un << produit >> de n fois l'automate modélisant une machine à 1 processus.

Les autres travaux de modélisation des performances des architectures multiflots que nous connaissons [SBCvE90, VVK96, NG95, SMH94, Fan93] utilisent exclusivement des techniques de files d'attentes qui nécessitent de connaitre certaines distributions. Les distributions sont supposées connues et fixées de manière arbitraire.

Pour notre part, nous caractérisons une borne supérieure et/ou une borne inférieure du gain que procure potentiellement une architecture à $n \geq 1$ thread(s). Le calcul des bornes est effectif et se réalise sur une expression algébrique (au sens des langages formels) du produit synchronisé. Le produit synchronisé est le produit d'automates intersecté avec des contraintes, dans notre cas, la contrainte qui interdit d'avoir plus de $k \geq 1$ flots d'exécution actifs simultanément sur une machine à $k \geq 1$ unités fonctionnelles de calcul par exemple.

Il s'agit là d'une approche complètement inédite de l'efficacité d'une architecture multiflot. De plus nous ne caractérisons pas une efficacité moyenne mais des bornes maximales et/ou minimales de l'efficacité et ceci à l'aide de techniques algébriques. Cette approche est nouvelle par rapport aux études précédentes [OH97,Aga89,SBC91, Squ94] d'utilisation de la théorie des files d'attente qui nécessitent de caler arbitrairement des lois de probabilité sur les évènements.

Dans notre modèle, nous faisons l'hypothèse que toutes les actions élémentaires sont unitaires (elles coûtent une seule unité de temps). Ainsi nous modélisons les systèmes

multiflots à grain fin c'est à dire ceux qui autorisent un changement de contexte à chaque cycle processeur comme par exemple la machine MTA de Tera[1] qui possède au niveau matériel 128 contextes de processus permettant de masquer jusqu'à 128 cycles processeurs.

1. Objectif de la modélisation d'un processeur multiflot

Le modèle à un seul flot de la machine est donnée à la Figure 1 (partie de gauche). L'unique unité fonctionnelle de calcul ne peut passer que dans quatre états qui sont : ne rien faire (état idle), calculer (état running), quitter un calcul (état leaving) et donc de changer de contexte et enfin être bloqué sur la latence de changement de contexte (état blocked).

Nous déterminons quand nous avons $n, (n \to \infty)$ threads actifs quelle est la meilleure et la pire exécution, qui correspondent respectivement au mot comprenant le plus de lettres "Running" et celle qui en comprend le moins. Cela définit deux courbes : la courbes des meilleures performances et celles des moins bonnes. L'aire comprise entre ces deux courbes représente intuitivement la qualité de la machine multiflot.

La question centrale est de dire si par exemple une machine à 128 threads et 4 unités d'exécutions est meilleure qu'une machine à 64 threads simultanés et 8 unités d'exécutions.

Les exécutions d'une machine multiflot à $n > 1$ threads sont modélisées par le *produit synchronisé d'automates*. Rappelons de manière intuitive que le produit synchronisé d'automates est un produit (cartésien) d'automates auquel on enlève des mots. Sur la Figure 1, partie de droite, nous avons un exemple de produit syncronisé de deux automates où l'on interdit que les deux threads soient en même temps dans l'état "running"... ce qui modélise un processeur multiflot à une seule unité d'éxécution.

Considérons un exemple complémentaire. De l'état "ready,running" (qui modélise que le premier thread est dans l'état ready et le second dans l'état running, lorsque l'on exécute la transition "e1,R" (le premier thread effectue une l'action nulle e1 et le second une transition R) on arrive alors à l'état "running,leaving".

Notons que la modélisation des systèmes multiflots à gros grain (basculement tous les $k > 1$ cycles processeurs ou au plus tous les k cycles) peut se réaliser en ajoutant k états dans l'automate. Pour modéliser les systèmes multiflots simultanés (il y a plusieurs unités de calcul qui peuvent être actives en même temps) il est juste nécessaire de modifier le produit synchronisé pour autoriser cette situation.

Il s'agit alors de calculer l'exécution où le CPU fait le plus de travail utile possible en étant dans l'état "Running" et de calculer l'exécution où le CPU exécute le plus d'instructions de gestion des flots. L'*efficacité* d'une exécution de la machine multiflot

1. Voir `http://www.cray.com` - TERA a été rachetée par Cray Systems

est définie comme le nombre de fois où l'on est dans l'état "Running" vis à vis de la somme de toutes les actions effectuées dans cette exécution. Cette définition est étendue à un langage (toutes les exécutions possibles de la machine multiflot) sans difficulté en prenant la borne inférieure (ou supérieure) de l'efficacité des différents mots.

Nous obtenons de la sorte une notion de *taux de parallélisme* d'un langage c'est à dire de l'ensemble des exécutions possibles de la machine multiflot.

Notre résultat principal [CP98] dit que ces métriques sont calculables et de manière modulaire (le calcul peut se faire en le décomposant en plus petits calculs à partir d'une expression régulière).

La force du résultat principal repose d'abord sur le fait que les langages obtenus par produit et intersection avec les contraintes sont reconnaissables. Ils peuvent donc être représentés par une expression régulière dont la forme générale est une union de produit de << sous-langages réguliers >> munis de l'opération d'étoile. Le calcul est modulaire dans le sens ou nous le faisons par une induction sur la hauteur d'étoile (le nombre d'expressions régulières << sous >> une étoile) de l'expression régulière.

Nous avons démontré que le taux de parallélisme de l'union de deux langages est le maximum (respectivement le minumum) des taux de parallélisme des deux langages. De manière plus surprenante, le taux de paralléliste du langage L^* est égal au taux de parallélisme de L. Enfin pour compléter la preuve de notre résultat principal nous avons montré un résultat concernant la concaténation de deux langages.

Ainsi, les trois principaux lemmes utilisés que nous avons démontrés dans le calcul du taux de parallélisme minimum sont les suivants :

LEMME : 1. *Soient* $L, L' \subseteq \Sigma^+$ *des langages non vide. Alors*

$$\mathfrak{E}_{min}(L \cup L') = \min\{\mathfrak{E}_{min}(L), \mathfrak{E}_{min}(L')\}$$

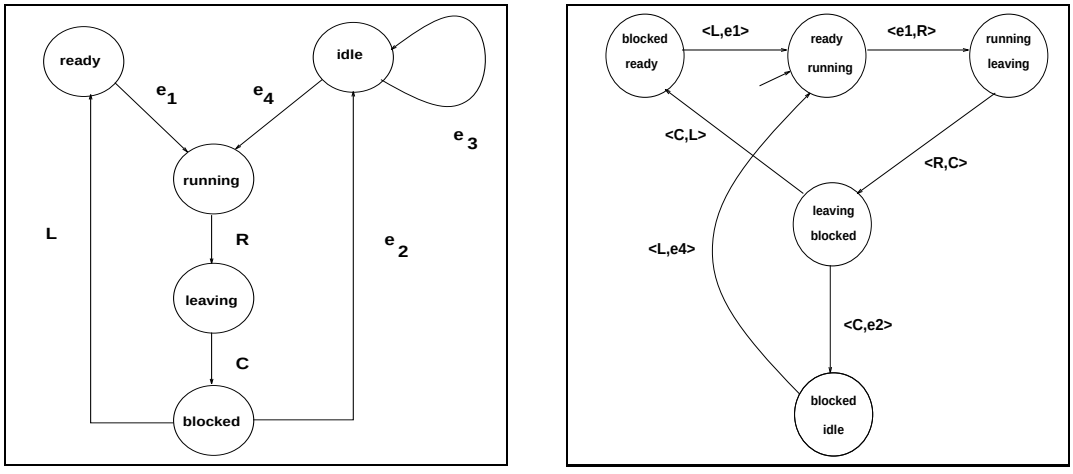

FIG. 1: *À gauche, modèle à un seul thread - À droite, produit synchronisé*

LEMME : 2. *Soit* $L \subseteq \Sigma^+$ *un langage non vide. Alors*

$$\mathfrak{E}_{min}(L^*) = \mathfrak{E}_{min}(L)$$

LEMME : 3. *Soient* $L_1, L_2, L_3 \subseteq \Sigma^*$ *des langages non vide. Alors*

$$\mathfrak{E}_{min}(L_1 L_2^* L_3) = \min\{\mathfrak{E}_{min}(L_1 L_3), \mathfrak{E}_{min}(L_2)\}$$

Exemple : considérons l'expression régulière et le langage R correspondant à l'automate à droite de la Figure 1 :

$$R = \Big(< e1, R >< R, C > . \Big(< C, e2 >< L, e4 > \bigcup < C, L >< L, e1 > \Big) \Big)^*$$

Le taux de parallélisme inférieur du langage est calculé de la manière suivante : $\mathfrak{E}_{min}(R) = \mathfrak{E}_{min}\left((< e1, R >< R, C > . \ (< C, e2 >< L, e4 > \bigcup < C, L >))\right)$ et à partir des lemmes intermédiaires on dérive que

$$\begin{aligned} \mathfrak{E}_{min}(R) &= \min\{\mathfrak{E}(< e1, R >< R, C > . < C, e2 >< L, e4 >), \\ &\quad \mathfrak{E}(< e1, R >< R, C > . < C, L >< L, e1 >)\} \\ &= \min\{\tfrac{2}{4}, \tfrac{2}{4}\} \\ &= \tfrac{1}{2} \end{aligned}$$

Ce résultat signifie que l'unité d'exécution est utilisée dans le pire cas à 50%. On ne peut pas faire moins bien.

2. Problèmes pratiques ouverts

Techniquement, comme il y a une explosion combinatoire du nombre d'états à traiter lorsque le nombre de threads croit, il serait judicieux d'utiliser des outils de codage comme les BDD (Binary Decision Diagram) [S.B78, R.E86, Sen96, MJH98]... car les 2^{32} états sont vite atteints dans le produit synchronisé. Les BDD permettent en particulier une représentation compacte en mémoire des automates. Ils ont été conçus initialement pour coder les fonctions booléennes et ont de nombreuses variantes permettant de factoriser au mieux les expressions booléennes conduisant au même état.

La question de savoir qu'elle est la famille de BDD à utiliser dans notre cas pratique n'est pas tranchée. Imaginons qu'elle le soit. Il s'agirait alors de répondre à la question suivante : soient deux processeurs multiflots caractérisés par leurs temps de latence sur un changement de contexte, leurs durées de cycle maximale avant un changement de contexte, leurs coûts de changement de contexte et leurs nombres de flots simultanés possibles, quel est le meilleur (celui qui utilise au mieux les unités fonctionnelles réalisant des calculs utiles) des deux processeurs ?

Notre travail se poursuivra dans ces directions. Il a d'abord consisté à donner un fondement théorique à l'évaluation des machines multiflots.

CHAPITRE 2

Hiérarchie mémoire : impact du cache et des disques

1. Introduction

La loi d'Amdahl [Amd67], imaginée dans le cadre du calcul parallèle en mémoire centrale nous a appris que c'était la portion de code séquentiel et pas la partie parallèle qui dominait les performances en temps d'une application. Par exemple, si 10% du programme est séquentiel, le gain procuré par un programme parallèle sera inférieur à 10 et ce quel que soit le nombre de processeurs.

La partie séquentielle d'un code doit donc aussi être examinée dans les détails si l'on veut tirer vers le haut les performances des processeurs. Les caches des processeurs jouent un rôle important dans les performances. Leurs architectures, leurs gestions ont un impact sur les performances. Comme le nombre de caches présents sur les processeurs a tendance à croître ; il semblerait que le passage de deux à trois caches se généralise ; c'est donc à une *hiérarchie* de cache à laquelle nous devons nous intéresser.

Un paramètre important est le nombre de fois où l'on a un *défaut de cache* qui entraine le chargement d'une nouvelle ligne dans le cache.

Nous étudions dans ce chapitre, à travers la problématique du tri séquentiel, les relations qui peuvent exister entre les problèmes de cache et les problèmes avec les disques. Nous pouvons en effet faire remarquer qu'après tout, le disque n'est qu'un cache de données comme un autre et donc que tout problème nécessitant les disques doit ou peut se traiter par des approches et des techniques de gestion des caches. La problématique générale est donc celle de la hiérarchie mémoire : les problèmes avec les disques auront de << bonnes solutions >> (algorithmiques, pratiques) lorsque l'on saura bien traiter le cache. Il n'y aurait pas à proprement parlé de problématique disque ?

En fait, nous nous sommes intéressés à l'impact du cache de premier niveau dans le cas du tri en mémoire et pour les défauts sur le cache de données. On suppose que si l'on arrive à réduire le nombre de défauts de cache sur le premier niveau, alors le nombre de défauts de cache dans les autres étages est aussi réduit.

De plus nous avons travaillé au niveau des compteurs de performace matériels disponibles sur tous les processeurs at pas à l'aide de simulateurs.

Le plan de la discussion qui suit est le suivant. Nous commençons par rappeler les liens, au niveau des modèles, des problèmes de gestion des caches et des problèmes de gestion efficace des disques. Nous donnons ensuite un exemple de modèle de hiérarchie mémoire et ses limites pour les clusters dont les nœuds sont constitués de plusieurs processeurs puis nous présentons trois techniques de gestion du cache dans le cadre du problème du tri qui, une nouvelle fois, nous sert de base de discussion. Nous

examinerons ensuite différents modèles de cache dont le modèle *oublieux du cache* (cache oblivious) introduit par Frigo, Leiserson, Prokop et Ramachandran [FLPR99].

Nous terminerons par donner les résultats que nous avons obtenus en matière d'impact des caches sur les performances du tri séquentiel d'entiers.

2. Modèle de disque et modèle de cache

Les problèmes de cache peuvent se capter par l'intermédiaire du modèle de disques PDM (Parallel Disk Model) [VS94a] que nous utilisons tout au long de la partie prospective. On interprète les paramètres du modèle PDM de la façon suivante:

N = taille du problème $\Leftrightarrow$ taille du problème,
M = taille de la mémoire interne (RAM) $\Leftrightarrow$ taille du cache,
B = taille d'un bloc transféré $\Leftrightarrow$ taille d'un bloc mémoire transféré dans le cache ou vers la mémoire,
D = nombre de disques indépendants $\Leftrightarrow$ nombre de bancs mémoire qui peuvent ête accédés simultanément,
P = nombre de processeurs de calcul. On prend cette valeur égale à 1.

Pour le modèle PDM, la borne au nombre d'opérations d'entrées sorties du problème du tri est connue... et donc aussi la borne au nombre de fois où l'on charge une donnée depuis la mémoire. Nous avons donc à notre disposition cette borne lorsque l'on considère l'interprétation précédente et pour la mémoire RAM vis à vis du cache. Cette borne nous serviva de point de comparaison afin de la comparer avec le nombre de défauts procurés par les algorithmes que nous proposons. Cette borne permet de fixer des objectifs quantitatifs.

Ainsi, le modèle PDM nous permet de capter les problèmes de performances liés aux caches et aussi ceux liés aux disques. Le modèle PDM est universel dans ce sens.

Nous nous intéressons à quelques modèles de hiérarchie mémoire, en particulier de cache afin de les mettre en perspective avec les modèles de disques. Nous prenons comme exemple, une nouvelle fois, le problème du tri séquentiel.

3. Le modèle HiHCoHP

Fanck Cappello et all. dans [CRE99, CRE00, CR99] s'intéressent à différents modèles de hiérarchie mémoire afin de déterminer si les clusters (de SMP) ont un intérêt. Selon leurs mesures effectuées sur des cartes bi-processeurs, ils montrent que dans le meilleur cas et par exemple sur les jeux d'essais NAS NPB 2.3 qu'un gain de $1,6$ s'obtient pour LU, FT et MG. Ils obtiennent même un gain de $1,9$ pour EP. Dans ces articles, la comparaison de deux modèles de hiérarchie fournit de précieuses indications. La méthodologie consiste d'abord à (re-)coder les codes en partie avec OPEN-MP et en

partie avec MPI. Il s'agit ensuite de mesurer les temps d'exécution correspondant à ces deux parties. La conclusion générale qui est tirée de ces exemples est qu'aucun des deux modèles étudié n'est supérieur à l'autre dans tous les cas. Par exemple le modèle hybride proposé convient bien aux tests BT et SP mais LU s'exécute mieux dans l'autre modèle de hiérarchie mémoire. Peut-on (doit-on) généraliser ses remarques pour le cas de cartes quadri ou octo-processeurs? Quant est il du problème du tri pour les deux modèles de hiérarchie mémoire proposée?

Concernant la hiérarchie mémoire et les modèles de communication à utiliser, nous pouvons noter les articles de Pierre Fraigniaux [Fra98] ainsi qu'un "extended abstract" intitulé << HiHCoHP - Toward a Realistic Communication Model for Hierarchical HyperClusters of Heterogeneous Processors >> écrit en collaboration avec F. Cappello, B. Mans et A. Rosenberg. Dans ce dernier article les auteurs proposent un modèle pour des hyperclusters (des clusters de clusters) de processeurs hétérogènes caractérisés par trois paramètres principaux qui rendent compte a) de l'hétérogénéité des processeurs b) de la hiérarchie des clusters d'hyperclusters c) de l'hétérogénéité des clusters dans chaque niveau de la hiérachie. Le seul exemple proposé est le cas de la *diffusion* d'un seul paquet depuis une source unique. C'est un point de départ. Il nous semble que le tri parallèle pourrait constituer un challenge intéressant, non trivial en matière de spécification dans le modèle. L'algorithme PSRS (Parallel Sorting by Regular Sampling) fait apparaître une diffusion des pivots vers chacun des processeurs ce qui est déjà traité dans le modèle HiHCoHP. Il reste principalement à spécifier le schéma de diffusion ou chaque processeur envoie une (plusieurs) valeur(s) vers un même processeur i.e. quand il s'agit de concentrer sur un processeur dédié les pivots sélectionnés. Cela ne semble pas être une tâche insurmontable. Par contre il nous semble assez difficile de pouvoir tester et valider le modèle à partir d'une << vraie machine >> c'est à dire d'écrire le code parallèle et d'obtenir pratiquement chacune des métriques du modèle et enfin de montrer que la prédiction est << conforme >> à l'expérimentation.

Pour l'instant, dans nos expérimentations nous avons supposé un modèle de programmation SPMD car les programmes dans ce modèle sont faciles à écrire, de notre avis. Nous pouvons aussi envisager que chaque nœud de calcul tourne une version différente de tri séquentiel par exemple, adaptée aux ressources du site : le processeur d'un assistant personnel n'a pas actuellement les mêmes caractéristique qu'un processeur de PC. Les questions qui se posent sont nombreuses : comment choisir la brique séquentielle la meilleure qui ne pénalisera pas l'exécution parallèle? Peut on développer un modèle de programmation avec des métriques permettant à priori d'estimer le temps d'éxécution, voire les ressources utilisées? Est ce que cela a un sens lorsque le support de communication est Internet? En effet on peut estimer à priori que ce sont les coûts de communication et non pas la hiérarchie mémoire interne à un processeur qui est le facteur déterminant pour des performances.

Cette dernière remarque est encore pertinente lorsque l'on considère la hiérachie disque. En effet, les temps d'accès aux disques sont d'un ordre de grandeur bien supérieur (actuellement) à ceux de la mémoire. À quoi bon s'intéresser à des algorithmes de tri « cache concious » comme [RKU00] et [ACVW01] puisque le temps d'éxécution est donné par le nombre d'opérations IO et pas par une gestion très éléborée des caches. Par contre, si Infiniband par exemple permet de faire en sorte que les temps d'accès aux disques soient comparables aux temps d'accès à la mémoire, alors ces techniques vont se développer, très certainement.

4. Rappels sur le cache mémoire

Dans le but d'accélérer l'accès à la mémoire centrale, de la mémoire rapide (avec un accès de l'ordre de quelques cycles processeur) et en petite quantité est placée entre le processeur et la mémoire centrale. C'est le cache. Puisque le cache est plus petit que la mémoire principale, il ne contient qu'une partie du contenu de celleci. Deux cas peuvent se présenter : ou bien la donnée à accéder réside dans le cache, alors elle est accédée directement.

On dit qu'un « cache hit » est survenu. Si la donnée à accéder n'est pas dans le cache alors cette donnée est recherchée dans la mémoire, le bloc qui la contient sera copié dans la mémoire cache puis la donnée sera accédée par le processeur. On dit qu'un défaut de cache ou « cache miss » est survenu.

Le « hit ratio » d'un programme est le rapport entre le nombre des « cache hits » et le nombre total des accès au cache. C'est la métrique qui mesure les performances du cache pour un programme donné. Pour la mémoire centrale, de manière analogue, on peut définir les « memory hit » ou « défaut de mémoire ».

On caractérise le cache mémoire par les paramètres suivants :

La capacité : c'est le nombre total d'octets que le cache peut contenir. Elle est notée Z dans la suite.

La taille du bloc : ou encore la longueur d'une ligne de cache, c'est le nombre d'octets qu'on peut lire ou écrire dans la mémoire centrale, à la fois. Elle est notée L dans la suite.

L'associativité : c'est le nombre de positions dans lesquelles un bloc particulier peut être chargé. Soit N ce nombre, le cache est alors appelé « Nway setassociative ». Les « caches à accès direct » (direct mapped caches) ont une associativité de 1 (un bloc ne peut être chargé que dans une seule position). Pour les « caches complètement associatifs » (fully associative cache), on peut charger un bloc dans n'importe quelle position du cache.

Stratégie de remplacement : c'est l'algorithme qui décide du bloc qui va être évincé lors d'un défaut de cache. Ce problème de choix ne se pose pas

pour les << directemapped caches >> puisque chaque bloc n'a qu'une seule position qui peut l'accueillir.

Dans les machines les plus modernes on peut trouver toute une hiérarchie de caches (généralement, deux ou trois niveaux) placés entre le processeur et la mémoire principale et ils sont ordonnés de manière à ce que le cache le plus proche du processeur soit le plus petit et le plus rapide en temps d'accès et le plus éloigné soit le plus grand en taille et le plus lent en temps d'accès. Ces caches respectent la propriété d'inclusion : les données contenues dans le cache i sont contenues dans le cache i+1 (i étant plus proche du processeur que i+1).

Il est admis que le seul comptage des instructions d'un programme ne suffit pas toujours pour obtenir une bonne prédiction de la durée d'exécution d'un programme. Pour analyser et prédire les performances d'un algorithme conformément à l'architecture réelle des calculateurs, il faut d'abord concevoir des modèles qui tiennent compte de ces composants architecturaux. Voici quelques exemples pertinents.

5. Motivations pour l'étude des algorithmes oublieux du cache

La notion a été introduite en 1999 par Frigo, Leiserson, Prokop et Ramachandran [FLPR99]. Il s'agit d'une des nombreuses propositions de modèle pour la hiérarchie mémoire mais ce n'est pas le seul modèle développé dans la littérature.

La notion sert à construire des algorithmes de sorte qu'aucune variable de l'algorithme comme par exemple la taille du cache et des lignes de cache n'a besoin d'être spécialisée (tuned) pour obtenir l'optimalité (dans un sens précisé plus loin dans le texte). Il a été prouvé qu'un algorithme oublieux du cache conçu pour deux niveaux de hiérarchie mémoire (par exemple, le CPU vis à vis du cache primaire) est aussi optimal pour un nombre quelconque de niveaux hiérarchiques de mémoire.

Les algorithmes *conscients du cache* (par opposition à oublieux du cache) sont des algorithmes qui contiennent des paramètres qui prennent leurs valeurs en fonction de la configuration de la mémoire cache. Ces paramètres peuvent être initialisés à la compilation ou à l'exécution : ils sont spécialisés (par des ajustements à la compilation ou à l'exécution) pour améliorer les performances. Pour mieux illustrer cette notion considérons le problème de la multiplication matricielle (calculer la matrice C qui est le produit des matrices A et B). Les trois matrices sont de taille $n \times n$. Nous supposons d'abord que n est assez grand : $n > L$ (L est la taille d'une ligne de cache (bloc)). Pour calculer C, on a recourt à un algorithme par blocs (blocked algorithm). L'idée est de voir chaque matrice comme une juxtaposition de $(n/s) \times (n/s)$ sous-matrices. Chaque sous-matrice est de taille $s \times s$. s, ici, est un paramètre à régler. L'algorithme du produit a alors cette forme générique :

```
Block_Mult(A, B, C,n)
Pour i de 1 à n/s faire
     Pour j de 1 à n/s faire
          Pour k de 1 à n/s faire
                Ord_Mult(Aik, Bkj, Cij, s).
```

où `Ord_Mult(A, B, C, s)` est une procédure qui calcule $C = C+AB$ sur des matrices $s \times s$ avec la complexité habituelle en $\mathcal{O}(s^3)$. Selon la taille de la mémoire cache de la machine sur laquelle on va tourner ce programme, on peut ajuster le paramètre s de manière à minimiser les défauts de cache pour chaque appel à la procédure `Ord_Mult`. Il s'agit de faire en sorte que les trois sous-matrices peuvent être contenues, ensembles, dans le cache.

Notre motivation initiale pour l'étude des algorithmes oublieux du cache vient du fait que nous étudions les algorithmes parallèles de tri en milieu hétérogène. Pour cela nous avons besoin d'utiliser des algorithmes séquentiels qui trient des sous problèmes du problème initial. En utilisant des algorithmes oublieux du cache dans la construction d'algorithmes parallèles, nous pouvons nous concentrer sur la problématique de la mise en parallèle d'actions en milieu hétérogène plutôt que sur les aspects de hiérarchie mémoire.

Enfin, si la hiérarchie mémoire est bien captée avec la notion d'algorithme oublieux du cache tant sur le point théorique que sur les implémentations qu'on peut faire avec les premiers niveaux de mémoire alors on envisagera d'examiner expérimentalement le comportement de ces mêmes algorithmes vis à vis des disques. Un algorithme oublieux du cache est conçu pour fonctionner aussi à ce niveau de hiérarchie mémoire. Pour l'exemple du tri externe, peut être que l'on pourrait montrer que pour obtenir des performances, il est suffisant de bien gérer les disques (par un algorithme oublieux du cache) plutôt que de chercher à développer des algorithmes dépendants de paramètres matériels (taille des blocs disque lus, par exemple).

Les caractéristiques précédentes font que de la notion est particulièrement intéressante. Dans ce mémoire, nous nous intéressons plus spécifiquement au cas du tri. Pour le cas du tri, d'autres modèles que le modèle oublieux du cache et tenant compte de la hiérarchie mémoire ont par ailleurs été proposés dans un passé récent, par exemple [AV88, ACFS94, S.C00] mais les performances du tri dans ces modèles n'ont pas été étudiés. Ainsi, nous ne pouvons pas exploiter ces modèles immédiatement.

5.1. Détails sur le modèle oublieux du cache. Les auteurs [FLPR99] introduisent la notion de *modèle idéal de cache* de paramètre (Z, L) qui consiste en une machine à deux niveaux de hiérarchie mémoire consistant en un cache de données de Z mots, chaque ligne du cache étant constituée de L mots, qui est une quantité, dans la pratique, plus grande que 1 ceci afin d'amortir les coûts de lecture écriture

dans le cache. On fait l'hypothèse que le cache est de petite taille: $Z = \Omega(L^2)$ ce qui est généralement vrai en pratique. Mais c'est aussi pour simplifier l'analyse des algorithmes et pour réutiliser un résultat de compléxité de [AV88], que les auteurs font cette hypothèse.

Le processeur ne peut que référencer le cache: si on trouve dans le cache (cache hit) alors le mot est délivré au processeurs, sinon un défaut de cache (cache miss) est généré et on doit rapatrier une ligne de cache depuis la mémoire. On fait aussi l'hypothèse que le cache est parfait dans le sens "fully associatif" : une ligne lue depuis la mémoire peut aller n'importe où dans le cache. Dans la pratique ceci n'est pas tout à fait vrai car il est très coûteux en terme de circuit d'implémenter un tel cache. Cependant les auteurs donnent des justifications théoriques de cette hypothèse. Nous ne les détaillons pas ici.

Enfin le cache doit avoir une politique de gestion quand il devient plein : une ligne doit être évincée. Le modèle oublieux du cache prend la politique optimale d'évincer la ligne dont le prochain accès est le plus loin dans le futur. Des justifications théoriques sont données pour montrer que cela est pertinent vis à vis d'une politique LRU (Least Recently Used).

La *complexité* en terme de cache $Q(n, Z, L)$ est le nombre de défauts de cache produit par l'algorithme. Notons qu'avec les définitions précédentes, telles qu'elles sont données dans [FLPR99], il n'est pas clair de savoir si une éviction du cache est comptée dans le dénombrement des opérations d'entrée sortie. Nous essayerons de répondre à cette question en présentant un algorithme de tri oublieux du cache qui figure dans [FLPR99].

Enfin, les auteurs définissent la notion *d'algorithme non oublieux* (cache aware) comme étant un algorithme qui contient des paramètres initialisés à la compilation ou à l'exécution qui peuvent être spécialisés (tuned) pour améliorer la complexité de cache.

Pour le cas du tri, Frigo, Leiserson, Prokop et Ramachandran [FLPR99] montrent, à partir d'un idée de Aggarwal et Vitter dans [AV88] que la complexité cache de n'importe quel algorithme de tri séquentiel est :

$$Q(n) = \Omega(1 + (n/L)(1 + \log_Z n)) \tag{1}$$

La preuve se fait à partir d'un résultat de [AV88] en spécialisant avec $Z = \Omega(L^2)$. Un des avantages des algorithmes oublieux du cache est intuitivement le suivant: une fois que les données d'un sous-problème sont contenues dans le cache, un sous-problème plus petit du premier, peut être résolu sans avoir à traiter des défauts de cache supplémentaires.

6. Le Modèle I/O de Aggarwal et Vitter

Il a été conçu [AV88] dans le cadre des disques comme éléments de la hiérarchie mémoire. Si on considère un programme qui traite des données situées dans une mémoire externe, on estime réaliste de calculer sa complexité en temps seulement en fonction des opérations d'entrée/sortie. En effet, il existe un rapport très grand entre les temps d'accès à la mémoire externe et le temps d'un cycle processeur, donc on peut négliger tous les calculs fait par le processeur devant le temps nécessaire à une opération d'entrée/sortie.

Le modèle initial a ensuite donné lieu à l'étude de nombreux algorithmes d'usage généraux (tri, fft...). Les paramètres du modèle sont les suivants et ils ont déjà été présentés à la Figure 1 page 100 et suivantes: N (nombre d'enregistrements), M (nombre d'enregistrement qui peut tenir en mémoire principale), B (le nombre d'enregistrements qui peuvent être transférés en une opération), P (nombre de blocs mémoire qui peuvent être transférés concurremment). On autorise ainsi du parallélisme puisque chaque opération de transfert (vers ou depuis la mémoire) concerne B enregistrements (supposés consécutifs sur le disque). De plus on autorise P transferts simultanés ce qui modélise partiellement certaines caractéristiques des disques comme le fait d'avoir plusieurs canaux d'entrées/sorties ou plusieurs têtes.

Ce modèle est tout particulièrement intéressant parce qu'il permet de montrer le résultat suivant à propos du tri séquentiel disque de N objets avec $D \geq 1$ disques:

THÉORÈME : 1 ([AV88], [NV95]). *The average and worst-case number of I/Os required for sorting* $N = nB$ *data items using* D *disks is:*

$$\mathrm{Sort}(N) = \Theta\left(\frac{N}{DB}\log_{M/B}\frac{N}{B}\right) = \Theta\left(\frac{n}{D}\log_m n\right) \tag{2}$$

Ce théorème est aussi présenté dans les discussions à partir de la page 100 et nous remarquons que dans la pratique le terme $\log_m n$ est une petite constante.

Leiserson dans [FLPR99] utilise une variante du théorème précédent qui est la suivante (voir [AV88]):

THÉORÈME : 2. *The average and worst-case number of I/Os required for sorting* N *records is:*

$$\mathrm{Sort}(N) = \Theta\left(\frac{N}{P.B}\frac{\log(1+N/B)}{\log(1+M/B)}\right) \tag{3}$$

For the lower bound, the comparison model is used, but only for the case when M and B are extremely small with respect to N, namely, when $B\log(1+M/B) = o(\log(1+N/B))$.

En fait, Leiserson fait l'hypothèse que la taille Z du cache est $\Omega(L^2)$ (L est la taille d'une ligne de cache) et ils utilisent ce dernier théorème pour dériver la borne de la complexité du tri dans le modèle << oublieux du cache >> .

Enfin, notons que le modèle PDM modélise aussi le premier niveau de cache grâce à l'analogie suivante : M représente maintenant la taille du cache, B la taille d'une ligne de cache. Nous devons aussi forcer $P = D = 1$. Les deux théorèmes précédents donnent alors les bornes sur le nombre de défauts. Nous privilégierons ce modèle pour nos études.

7. Le modèle de cache de LaMarca

En s'inspirant du modèle précédent plusieurs autres chercheurs (LaMarca et Ladner [LL99], Sen et Chatterjee [S.C00]) ont proposé un modèle de cache dans lequel le cache joue le rôle de la mémoire centrale, et la mémoire centrale joue le rôle du disque. Il faut, par ailleurs, tenir compte des quelques différences suivantes entre les deux modèles :

(1) Dans le modèle I/O le temps de calcul est négligeable par rapport au temps d'une E/S. Dans le modèle du cache, le temps de calcul est inférieur mais comparable au temps d'accès mémoire;
(2) Les défauts de cache n'augmentent pas le temps d'exécution s'il y a des instructions qui peuvent s'exécuter pendant le traitement du défaut de cache ;
(3) Les registres peuvent contenir les données les plus fréquemment utilisées ;
(4) La taille du TLB ("Translation Lookside Buffer" : qui assure la traduction des adresses virtuelles en des adresses physiques) limite le nombre de sections de la mémoire accessibles à la fois ;
(5) Le cache a une associativité de valeur faible ;
(6) Les systèmes mémoires sont optimisés pour des accès séquentiels.

Pour ce modèle, il n'y a pas à note connaissance de résultat de compléxité permettant de borner le nombre de défauts de cache pour le cas du tri. C'est pour cela que nous n'avons pas utilisé ce modèle.

8. Tris séquentiels tenant compte de la hiérarchie mémoire

Le travail d'expérimentation s'appuie en particulier sur un ensemble d'algorithmes classiques (QuickSort, MergeSort optimisés pour le cache) ainsi que sur des implémentations et des algorithmes originaux, jamais testés au niveau des compteurs de performances mais qui valent soit pour leur performance théorique (le tri rapide de Nilsson a une complexité théorique de $\mathcal{O}(n \log\log n)$) soit pour les structures de contrôle utilisées (FAME).

Par ailleurs nous avons développé un nouvel algorithme de tri séquentiel qui tient compte du cache (ZZZmerge) qui est un tri par fusion. Les résultats de performance de cet algorithme sont donnés dans le prochain paragraphe.

Dans les sous paragraphes qui suivent nous présentons les algorithmes testés.

8.1. Funnelsort (tri en entonnoir). C'est un algorithme présenté en [FLPR99] dont la complexité en terme d'instructions est asymptotiquement optimale ($\mathcal{O}(n \log n)$) ainsi qu'une complexité cache optimale en $\mathcal{O}(1 + (n/L)(1 + \log_Z n))$. Funnelsort est une variante particulière du tri fusion où la fusion est assurée par un mécanisme appelé *k-fusionneur*:

1: On partage le tableau en entrée en $n^{1/3}$ tableaux de taille $n^{2/3}$ que l'on tri de manière récursive.
2: On fusionne les $n^{1/3}$ tableaux triés avec un $n^{1/3}$-fusionneurs.

Un k-fusionneur reçoit k séquences triées et les fusionne. Il produit à chaque invocation k^3 éléments. Il fonctionne de manière récursive produisant à chaque fois des séquences plus longues. Un k-fusionneur a la spécificité aussi de suspendre le travail de fusion d'un sous problème dès que la séquence fusionnée à la sortie devient << suffisamment longue >>, puis il entame un autre sous-problème. Chaque k-fusionneur est construit de manière récursive avec les $k^{1/2}$-fusionneurs situés au premier étage du k-fusionneurs, noté $L_1, L_2, \cdots, L_{k^{1/2}}$. Les sorties de ces $k^{1/2}$-fusionneurs sont connectées à $k^{1/2}$ tampons, chacun de taille $2k^{3/2}$. Au deuxième étage du k-fusionneur on trouve un seul $k^{1/2}$—fusionneur, noté R, dont les entrées sont les $k^{1/2}$ sorties des $k^{1/2}$ tampons (voir la Figure 1). Le cas de base de la récursivité est pour $k = 2$; un 2-fusionneur produit alors 8 éléments lorsqu'il est invoqué. La Figure 1 donne un exemple de 16-fusionneur.

Pour produire une séquence de k^3 éléments, le k-fusionneur fait appel au $k^{1/2}$-fusionneur R $k^{3/2}$ fois. Avant chaque appel, le k-fusionneur doit alimenter les tampons qui sont remplis à moins de la moitié, ceci en faisant appel au $k^{1/2}$-fusionneur L_i correspondant, qui produit $k^{3/2}$ éléments, ainsi le buffer correspondant devient rempli à plus que la moitié.

On a les résultats suivants [FLPR99]: un k-fusionneur nécessite $\mathcal{O}(k^2)$ places de mémoire pour les tampons, en plus des places nécessaires pour les $k^{1/2}$-fusionneurs.

Pour assurer la borne de complexité, les tampons dans un k-fusionneur doivent être implémentés comme des files circulaires. Sachant que si on traite un défaut de cache pour une insertion (respectivement une enlèvement) d'un élément de la file alors les insertions (respectivement enlèvements) suivantes seront faites sans défaut de cache (L étant la taille d'une ligne de cache). On en déduit que l'exécution de r insertions

ou enlèvements d'une file circulaire cause $\mathcal{O}(1+r/L)$ défauts de caches tant qu'il y a deux lignes de caches qui sont disponibles dans le buffer.

Les auteurs ont prouvé qu'un k-fusionneur s'exécute avec au plus :

$$Q_M(k) = \mathcal{O}(1 + k + k^3/L + k^3 \log_Z(k/L))$$

défauts de cache, et par conséquent, pour trier n éléments funnelsort fait

$$\mathcal{O}(1 + (n/L)(1 + \log_Z n))$$

défauts de cache ce qui est optimal au sens oublieux du cache.

8.2. Distribution sort. Tout comme l'algorithme précédent, distribution sort a été présenté en [FLPR99] et il réalise les mêmes complexité : $\mathcal{O}(n \log n)$ en terme d'instructions, et $\mathcal{O}(1+(n/L)(1+\log_Z n))$ défauts de cache. En partant d'un tableau A de longueur n, rangé dans des positions contiguës, l'algorithme se déroule comme suit :

(1) On partitionne A en $n^{1/2}$ sous-tableaux contiguës, chacun de taille $\sqrt{n}$. On procède au tri, récursivement, de chaque sous tableau.
(2) On distribue les $n^{1/2}$ sous-tableaux triés en q paniers $B_1, B_2, \cdots, B_q$ de taille $n_1, n_2, \cdots, n_q$ tels que :
   - $\max\{x | x \in B_i\} \leq \min\{x | x \in B_{i+1}\}$ avec $i = 1, 2, \cdots, q$.

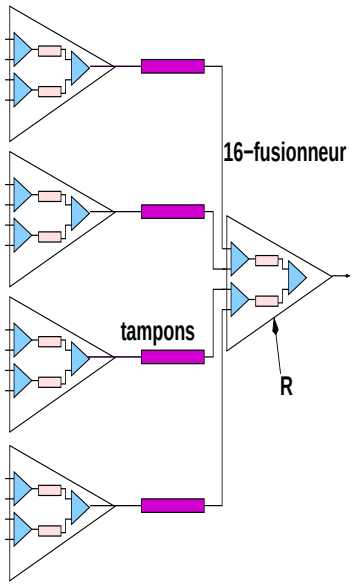

FIG. 1: *Exemple de 16-fusionneur*

– $n_i \leq 2n^{1/2}$ avec $i = 1, 2, \cdots, q$.

(3) Trier de manière récursive chaque panier.
(4) Copier les paniers triés dans A.

On définit alors l'état d'un sous tableau par son l'indexe `next` qui pointe vers l'élément du sous-tableau qui va être copié dans un panier ainsi que le numéro du panier `bnum` dans lequel l'élément pointé par `next` va être copié.

L'étape de distribution (étape 2) est assurée par la procédure récursive `DISTRIBUTE(i, j, m)` qui distribue les éléments de m sous-tableaux (du sous-tableau i au tableau $i + m - 1$) sur les paniers commençant par le panier B_j.

On montre que l'étape de distribution fait un travail en $\mathcal{O}(n)$ et des défauts de cache en nombre $\mathcal{O}(1 + n/L)$.

Pour trier n éléments, l'algorithmes *Distribution Sort* fait alors un travail en $\mathcal{O}(n \log n)$, et cause $\mathcal{O}(1 + (n/L)(1 + \log_Z n))$ défauts de cache ce qui est optimal.

8.3. L'approche FAME (Finite Automaton MErgesort). La technique utilisée dans [RKU00] pour trier en utilisant au mieux le premier niveau de hiérarchie mémoire (cache et registres processeur) consiste en un tournoi entre les différents éléments en tête des séquences triées. Cette technique est appelée FAME (Finite Automaton MErgesort)

Un automate d'état fini est construit pour décider de ce que l'on fait après un tournoi. Le principal problème réside dans le fait que si l'on a m séquences à fusionner, alors il y a 2^{m-1} états dans l'automate et au total, le nombre de lignes de code nécessaire pour code l'automate est $\mathcal{O}(m2^m)$. Dans la pratique, pour une machine donnée, il faut donc trouver le bon compris entre la valeur de m et la taille du cache de premier niveau pour faire en sorte que tout le code de l'automate tienne dans le premier niveau cache de la hiérarchie mémoire.

Les auteurs montrent que dans certains cas (entre 8K et 512K données environ), leur approche est meilleure que Quicksort pour des mesures effectuées sur UltraSparc, Alpha250. Par contre, avec un Pentium l'approche FAME ne fournit pas de bons résultats comparée à QuickSort : les registres ne sont pas en assez grand nombre.

8.4. Tri rapide de Nilsson. L'idée présentée en [AHNR95], implémentée en [Nil00] conduit à un algorithme de tri d'entiers en $\mathcal{O}(n \log\log n)$. En fait l'algorithme permet de trier, si les mots machine sont codés sur w bits, n entiers dans l'intervalle $0 \cdots 2^{w-1}$ en temps $\mathcal{O}(n \log\log n)$ et pour un w choisi arbitrairement et vérifiant $w \geq \log n$.

Pour réaliser cette borne, les auteurs utilisent les opérations de comparaison, d'addition, de soustraction, de manipulation de bit (AND, OR, décalage) uniquement.

Les deux techniques utilisées sont un compactage de plusieurs entiers dans un mot et la possibilité de réduire l'intervale des valeurs à un intervalle plus petit que celui de départ. Ces deux techniques sont connues respectivement sous le nom de "packet sorting" et "range reduction". Les auteurs réussissent à stocker $k = \log n \log\log n$ clés dans un mot machine et utilisent ensuite un tri fusion (un peu spécial) pour fusionner deux séquences triées, chacune contenant k clés.

8.5. ZZZmerge. Nous avons développé notre propre algorithme de tri séquentiel qui tient compte du cache. `ZZZmerge` capitalise sur la *localité temporelle* et la *localité spaciale* des données. On dit qu'un programme présente de la *localité temporelle* s'il y a de bonnes chances pour que la donnée à laquelle on accède le soit encore dans un futur proche. Un programme présente de la *localité spaciale* si on a de bonnes chances que les données auxquelles on va accéder soient dans un voisinage mémoire de la donnée accédée.

Le principe de `ZZZmerge` est simple. On commence par trier des segments de taille $z * z$ qui est le nombre d'entiers qui peuvent tenir dans une ligne de cache. Puis on organise un arbre virtuel qui va fusionner, niveau par niveau, deux sous listes triées. Comme tout algorithme de tri fusion, `ZZZmerge` requiert 2N place mémoire, N étant la taille de l'entrée.

Nous avons montré le résultat suivant :

THÉORÈME : 3. *Pour un vecteur en entrée de taille n et une taille de ligne de cache pouvant contenir $z * z$ entiers, le nombre de défauts de cache de ZZZmerge est exactement :*

$$n/(z * z) + \frac{n}{z * z} \log n/(z * z)$$

Le nombre de défauts de cache n'atteint pas la borne du nombre d'opérations d'entrées sorties pour le tri. Cependant, notre tri `ZZZmerge` bat en terme de temps d'exécution tous les candidats sélectionnés, en particulier une version optimisée de Quicksort qui procure les deuxièmes meilleurs résultats. Le facteur de gain entre `ZZZmerge` et cette version de Quicksort se situe entre 10 et 15% en faveur de `ZZZmerge`.

Expérimentalement, on a montré que plus le code était optimisé, on modifie pour cela les options d'optimisation à la compilation, meilleur était notre code. Cette remarque confère à notre code une bonne propriété.

9. Méthodologie d'expérimentation

À partir des idées précédentes, nous avons procédé comme suit. Premièrement, nous avons réalisé les implémentations des deux algorithmes oublieux du cache présentés dans [FLPR99] et nous avons repris les codes de FAME [RKU00], du tri rapide de

Nilsson appelé FastSort [AHNR95] et de 3-way Quicksort qui est une variante de Quicksort.

Comme les complexités de tous ces algorithmes connus s'expriment en terme de $\mathcal{O}$ notations, il faut s'intéresser aux constantes. En particulier, on ne sait pas si FastSort a de bonnes performances vis à vis du cache et s'il appartient (modulo quelques adaptations) à la classe des algorithmes oublieux du cache.

Si ces algorithmes ne se comportent pas très bien dans la pratique, peut être qu'il faudra revoir la définition de *oublieux du cache*. En effet, pour l'instant, la notion s'intéresse uniquement aux défauts de cache en lecture. Certes il est reconnu ([HP02] page 105) que les défauts en écriture pour les SpecInt92 représentent environ 7% du trafic mémoire alors que les défauts en lecture représentent environ 25% du trafic mémoire pour le même test. Ce n'est peut être pas le cas pour le tri ! Un raffinement de la notion qui ferait la distinction entre lecture et écriture dans et depuis le cache apporterait peut être plus de précisions. Les expérimentations envisagées devront clairement s'intéresser au trafic vers et depuis la mémoire.

9.1. Utilisation des compteurs de performances. Les compteurs de performances sont disponibles sur tous les microprocesseurs actuels. Ces compteurs sont implémentés dans des registres et ils mesurent des évènements liés à l'activité du processeur : le nombre de défauts de caches réalisé par un programme est accessible par exemple.

La boite à outils `Perfctr`[1] est l'un des outils très utilisés dans la communauté. Par example, des outils avec des interfaces graphiques comme PAPI [DLM+01] développé par l'équipe de Jack Dongarra, utilisent en sous main les fonctionnalités de `Perctr`. C'est un outil de bas niveau car pour l'utiliser il faut recompiler le noyau de son système d'exploitation.

Nous avons utilisé `Perfctr` pour mesurer le comportement du cache d'un AMD Athlon et pour sept programmes de tris séquentiels. Ces sept programmes sont à priori des bons candidats puisqu'ils ont été développé spécialement pour tenir compte du cache.

Aucun des programmes de tris connus, optimisés pour le cache ou pas optimisés, n'avaient été observé sous l'angle de leurs << vraies >> performances cache. Curieusement, les seules études connues à ce jour [LL99, Aga96, NBC+94, LPJN97, ACVW01, RKU00] étaient réalisées à partir de simulateur.

Mais dans toutes ces études, on mesure tout au mieux le temps d'execution et l'on dérive une approximation du nombre de défauts de cache. Dernièrement, N. Rahman [RR00] étudient radix sort et plus particulièrement l'importance de réduire les défauts

1. `http://user.it.uu.se/~mikpe/linux/perfctr/`

sur la TLB (Translation Look-aside Buffer). Là encore, aucune mesure n'est réalisée à vraie grandeur alors que des moyens matériels sont maintenant disponibles !

Nous avons comblé ce manque en proposant une étude exhaustive de tris séquentiels majeurs et optimisés pour le cache. Cette étude s'est effectuée à vraie grandeur et en mesurant effectivement le nombre de défauts.

Nous avons préféré travailler au niveau des compteurs de performance plutôt à l'aide de simulateurs pour les raisons suivantes :

- les observations sont fines ;
- les observations se font à coûts nuls ;
- le fait d'avoir plusieurs unités fonctionnelles, une execution spéculative, plusieurs niveaux de cache sont intégrées à la mesure ;
- les politiques dans le processus d'ordonnancement ainsi que la politique du système d'exploitation pour la mise en correspondance des adresses virtuelles avec les adresses physiques sont captées.

Les simulateurs, Atom [SE94] par exemple ne permettent pas de capter tous ces facteurs. Cependant, un simulateur à l'avantage de ne pas nécessiter la construction d'un circuit. L'évaluation du processeur se déroule à priori.

10. Résultats expérimentaux

Le principal résultat que nous avons obtenu pour le tri séquentiel est que `ZZZmerge` bat en temps tous ses concurrents. Cependant, `ZZZmerge` n'est pas le plus performant en terme de nombre d'instructions utilisées ni en terme de défauts de cache.

La grande lesson que nous tirons des expérimentations peut se résumer de la façon suivante : pour obtenir les meilleures performances possibles en terme de temps d'éxecution, il est nécessaire de développer un algorithme « bien équilibré » entre le nombre de défauts de cache de premier niveau et le nombre d'instructions exécutées.

La notion d'algorithme « bien équilibré » que nous proposons se résume par la conjecture suivante : « Soient X et Y deux programmes répondant à la même spécification. Si le programme X a une IPC (nombre d'Instructions Par Cycle) au moins deux fois plus grand que Y mais qu'il a un nombre de défauts de cache L1 et un nombre d'instructions exécutées deux fois plus important que Y, alors le programme X a quand même une exécution en temps meilleure que Y » .

Cette conjecture est très importante pour celui qui travaille en parallélisme. En effet, elle implique par exemple que le code le plus court n'est pas celui qui produira forcément le meilleur temps d'exécution mais qu'un soin tout particulier doit être

apporté à la construction de programmes avec << suffisamment d'instructions >> indépendantes qui pourront alimenter les différentes unités d'exécution des processeurs récents.

Les mesures sur l'Athlon montrent que ZZZmerge a une IPC de 1.4 (sous certaines conditions liées aux options de compilation). Comme l'Athlon est un "3-way superscalar x86 processor" on peut se demander s'il est possible de dériver un tri avec une IPC qui approche le plus près possible de 3? Il semblerait que les algorithmes connus depuis 40 ans (Quicksort...) ne soient pas capables de les atteindre parce qu'ils sont intrinsèquement trop optimisés pour le cas séquentiel!

Nous estimons qu'il y a encore des possibilités pour améliorer, sur les architectures de processeurs actuels (qui sont de plus en plus parallèles), les performances des algorithmes séquentiels, en particulier le tri. Nous pensons qu'un modèle capable de capter les principales caractérisques d'un cache en même temps qu'il capterait les principales caractéristiques d'unités fonctionnelles travaillant en parallèle apporterait beaucoup à un renouveau de l'algorithmique séquentielle et parallèle.

Notre objectif à moyen terme est de dégager des techniques générales pour utiliser au mieux le parallélisme au niveau du << circuit processeur >>.

Les résultats présentés dans ce chapitre ont été mené conjointement avec Mohamed Jemni , enseignant à Tunis et Hazem Fkaier, étudiant de DEA à tunis. Ils sont en cours de pré-publication. Ils peuvent éventuellement être consulté à l'adresse:

`http://www.laria.u-picardie.fr/~cerin/`

Troisième partie

PRAM et variantes

CHAPITRE 1

Modèles de calcul en mémoire commune

CALCULER ET PROGRAMMER en parallèle lorsque la mémoire est partagée par tous les processeurs à des avantages. En particulier le programmeur n'a pas à se soucier des communications afférentes aux accès à la mémoire. De nombreux systèmes parallèles à mémoire commune ont été construits par le passé. La limitation technique tient à ce que le dispositif permettant le partage de la mémoire (le bus) est un goulet d'étranglement avec les technologies actuelles. Le nombre de processeurs réellement exploitables par cette technique est pour l'instant limité à moins de 10 processeurs.

La mémoire commune a été très étudiée au niveau des modèles afin de classifier les problèmes. On admet maintenant qu'un algorithme parallèle en mémoire commune (dans le modèle PRAM - Parallel Random Access Memory) est performant si son temps d'exécution est logarithmique et qu'il utilise un nombre linéaire (vis à vis de la taille du problème) de processeurs.

Nous avons étudié deux modèles en mémoire commune : le modèle PRAM (Parallel Random Access Memory) et le modèle BSR (Broadcast with selective Reduction) qui peut être vu comme une PRAM avec des capacité de diffusion de données. Le chapitre est donc organisé autour de ces deux modèles et il précise nos apports dans chacun des modèles. Ces apports concernent l'obtention de bornes et d'algorithmes qui offrent des présentations favorables à des implémentations dans des langages à passage de messages ou encore des algorithmes qui s'écrivent avec moins d'instructions par rapport aux précédents algorithmes connus pour les problèmes traités.

1. Introduction au modèle PRAM (Parallel Random Access Machine)

Le modèle PRAM (Parallel Random Access Machine), pour lequel un grand nombre d'algorithmes ont été développés et synthétisés dans de nombreux ouvrages [MC93], [Jáj92], [Ble93], [Lei92], [Vis93], [Akl97] est un modèle théorique permettant de rendre compte de l'exécution concurrente de tâches dans un système composé de plusieurs processeurs (machine parallèle). Dans ce modèle, il est possible de classer les problèmes selon, par exemple, leurs performances en « temps partagé ». Des briques de base ont déjà été identifiées : somme préfixe, circuit eulérien, saut de pointeurs... qui, à l'usage, apparaissent dans beaucoup d'applications. Par exemple, il est bien connu que l'évaluation en parallèle d'une expression arithmétique peut s'implémenter à l'aide de la technique de la somme prefixe.

Le modèle PRAM est composé :

- d'un ensemble (à priori infini) de processeurs indicés, chacun connaissant son indice et possédant un compteur ordinal (pointeur d'instructions) ;

- d'une mémoire globale partagée infinie ;
- d'un contrôle global qui permet de gérer l'ensemble des processeurs, de leur envoyer les calculs à effectuer et de récupérer les résultats.

Dans le cas d'algorithmes utilisant la mémoire partagée, il est nécessaire de détailler la gestion des accès mémoires. Il faut, en effet, spécifier les types d'accès à la mémoire autorisés par le modèle. On distingue trois types de PRAM selon les accès concurrents permis sur la mémoire :

- EREW-PRAM : (Exclusive Read Exclusive Write) plusieurs processeurs ne peuvent au même instant ni lire, ni écrire dans une même cellule mémoire.
- CREW-PRAM : (Concurrent Read Exclusive Write) une cellule mémoire peut être lue à un instant donné par plusieurs processeurs ou, un seul processeur peut en modifier le contenu.
- CRCW PRAM : (Concurrent Read Concurrent Write) à un instant donné, une cellule mémoire peut être lue par plusieurs processeurs ou, plusieurs processeurs peuvent écrire dans la même cellule. Dans le cas d'écritures concurrentes, il existe plusieurs mode de gestion des conflits :
  - COMMUNE-CRCW : les écritures concurrentes ne sont valides que si tous les processeurs écrivent la même valeur, sinon le contenu de la cellule reste inchangé.
  - ARBITRAIRE CRCW : la valeur inscrite dans la cellule, est prise arbitrairement parmi les valeurs proposées par tous les processeurs candidats à l'écriture à cet instant.
  - PRIORITAIRE CRCW : on utilise les numéros d'indices des processeurs pour déterminer celui dont la valeur sera retenue. Une relation d'ordre (donnée par la relation minimum, maximum par exemple) est utilisée pour valider l'écriture d'un des processeurs.

Pratiquement les machines parallèles à mémoire commune sont limitées par les engorgements sur le bus partagé qui relie tous les processeurs à la mémoire. Il est même probable, puisque les temps de cycle d'accès à la mémoire décroissent moins vite que les temps de cycle des processeurs, que les seuls systèmes à mémoire commune viables dans le futur proche seront constitués d'au plus quatre processeurs (alors qu'on a construit par le passé des machines à 16, 32 nœuds).

1.1. La mise en correspondance de parenthèses sur PRAM. Dans le modèle PRAM, nous nous sommes intéressé à la mise en correspondance de parenthèses qui a pour application l'évaluation d'expressions arithmétiques par exemple : lorsqu'on isole dans une expression arithmétique quelle est la parenthèse fermante qui est en correspondance avec une parenthèse ouvrante, on peut demander l'évaluation de l'expression contenue entre ces deux parenthèses.

En séquentiel, la gestion d'une pile permet simplement en temps linéaire de résoudre le problème. Quand on rencontre une parenthèse ouvrante on l'empile, quand on rencontre une parenthèse fermante, on la met en correspondance avec celle du sommet de pile. Comme la solution optimale en séquentiel est linéaire et parait intrinséquement séquentielle, il y a travail non trivial à accomplir pour dériver une solution parallèle.

Le problème de mise en correspondance des parenthèses a été évoqué pour la première fois en algorithmique parallèle dans [BOV85]. Il s'agit donc de trouver dans une chaîne de parenthèses bien parenthèsée tous les couples de parenthèses présents.

De manière formelle, le problème du *parenthesis-matching* s'énonce comme suit : Soit $V[1 \cdots n]$ un vecteur contenant une chaîne de n parenthèses. Il s'agit de déterminer les couples de parenthèses présents dans la chaîne. On associe à chaque parenthèse ouvrante la position de la parenthèse fermante qui lui correspond et réciproquement.

Par exemple, considérons la chaîne $(_1)_2(_3(_4)_5(_6(_7)_8)_9)_{10}$ où chaque parenthèse est indicée par son rang. Il s'agit de retourner les informations suivantes :

$(_1$ est en correspondance avec $)_2$
$(_3$ est en correspondance avec $)_{10}$
$(_4$ est en correspondance avec $)_5$
$(_6$ est en correspondance avec $)_9$
$(_7$ est en correspondance avec $)_8$

Les premiers auteurs à notre connaissance à évoquer le problème de la mise en correspondance de parenthèses sont I. Bar-On and U. Vishkin [BOV85]. Le problème apparait également comme sous problème pour déterminer si deux arbres sont égaux dans [Sto96] et encore dans d'autres types d'applications.

Nous avons donné dans [BC97b], pour ce problème dans le modèle PRAM, un algorithme qui égale la complexité en temps du meilleur algorithme connu à ce jour, mais qui de part sa présentation où les calculs se déroulent en étant « pipelinés », est plus favorable, à notre avis, à une implémentation dans un langage par envois de messages. Autrement dit, la présentation proposée fait référence à un modèle de programmation de machine à mémoire distribuée. La principale difficulté technique a été de traiter des chaînes ayant un niveau d'imbrication maximun quelconques et pas en $\mathcal{O}(\log n)$.

La technique consiste à organiser un arbre virtuel de calcul où dans chaque nœud de l'arbre on met en correspondance des sous-chaînes. À un nœud de l'arbre correspond un processeur. Le problème n'est pas trivial car une partie des chaînes peut se mettre en correspondance avec des parties de chaînes situées chez un fils du nœud père et donc sur un autre processeur.

2. Le modèle BSP (Bulk Synchronous Parallel model)

2.1. Introduction. Nous présentons maintenant le modèle BSP (Bulk Synchronous Parallel model) introduit par valiant [Val90b] comme un modèle de calcul parallèle. Le modèle BSP à été implémenté et des librairies de programmation existent pour ce modèle.

Une grande partie des expérimentations que nous avons conduites ont été réalisées avec des librairies de programmation BSP (Bulk Synchronous Programing model). BSP est donc avant tout un modèle de calcul pour lequel des librairies existent. Il y a deux distributions : la première de l'université d'Oxford (disponible depuis 1996 environ) qui est la BSPLib et celle de l'université de Paderborn (disponible depuis 1999) et que l'on appelle PUB ou PUB7. On peut les retrouver à partir du lien `http://www.bsp-worldwide.org/bspwwact.htm`.

Un autre pointeur afin de récupérer des sources de tri en parallèle est notre page référencée `http://www.laria.u-picardie.fr/~cerin/=paladin`. Des implémentations différentes (en terme de primitives de communication) d'un même problème nous a permis d'évaluer les deux implémentations de BSP [CG01].

Les références [GS99] et [FS99] traitent du tri dans le cadre de BSP mais les auteurs ne fournissent aucune expérimentation bien que des librairies implémentant BSP existent.

De nos expérimentations ressort que PUB est plus efficace que BSPLib en particulier lorsuq'on utilise les primitives de lectures ou d'écriture distantes en mémoire (primitives `put, get`). Il y a une meilleure gestion des tampons servant à la communication.

Nous donnons maintenant un bref aperçu de BSP sur le plan du modèle de calcul.

2.2. BSP comme modèle de calcul parallèle. BSP (Bulk Synchronous Parallel model) est un modèle de calcul parallèle qui a été introduit par Valiant [Val90b] en 1990. C'est un modèle général du parallélisme qui intègre la prédiction des performances. Le modèle prédictif est capté via quatre paramètres : $\mathtt{p}$ le nombre de processeurs de la machine, $\mathtt{l}$ le temps de synchronisation des processeurs, $\mathtt{g}$ qui est la capacité du réseaux de communication sous des hypothèses de trafic continu et $\mathtt{g}$ qui est la vitesse de la machine. Nous utilisons les bibliothèques BSP afin de compter sur leurs simplicités (voire de leurs dépouillements en terme de primitives disponibles) dûe au modèle de programmation SPMD (Single Program, Multiple Data).

2.3. Fonctionnement de BSP. Un programme BSP est représentable par une structure à deux dimensions comme le montre la Figure 1 :

(1) La dimension horizontale illustre l'exécution concurrente des processus virtuels en nombre fixé. L'ensemble de ces processus n'est pas ordonné et peut

être projeté sur des processeurs de façon quelconque. La localité ne joue aucun rôle dans le placement des processus sur les processeurs.

(2) La dimension verticale montre l'évolution des calculs au cours du temps. Un programme BSP est une composition séquentielle S de super étapes $s(i)$ avec $0 \leq i \leq S$, lesquelles occupent conceptuellement la pleine largeur de l'architecture sur laquelle elles s'éxecutent. Chaque super étape est constituée de trois phases successives (voir la Figure 1):

   (a) une phase de calcul locale à chaque processus, utilisant les données stockées dans la mémoire associée au processeur exécutant le processus.

   (b) une phase de communication entre les processus, impliquant des mouvements de données entre les processeurs.

   (c) une barrière de synchronisation, annonçant la fin des communications et, rendant valide, dans la mémoire locale des processeurs destinations, les données en mouvement.

Ainsi, le modèle permet de prédire les performances (nous n'avons pas exploré, à travers l'exemple du tri, ni la pertinence ni l'exactitude de la prédiction) mais aussi de décrire de manière assez intuitive un programme parallèle en terme de super-étapes.

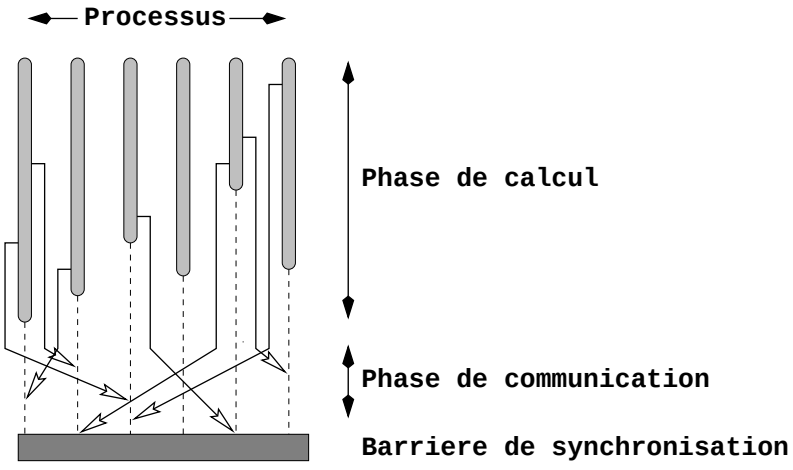

FIG. 1: *Exécution d'une bulle BSP.*

3. Le modèle BSR (Broadcasting with Selective Reduction)

3.1. Présentation du modèle de calcul. Le modèle BSR a été présenté dès son origine, par ses concepteurs, comme une « évolution naturelle » du modèle CRCW-PRAM [AG89]. Une propriété importante du PRAM tout comme BSR, est que l'accès à la mémoire se fait en temps constant.

Le modèle BSR est un modèle de calcul parallèle qui a été proposé par Selim Akl [AG89]. Il consiste en n processeurs partageant m locations mémoires. Les processeurs possèdent une petite mémoire locale et exécutent un jeu d'instructions simple. La mémoire partagée stocke les données et les résultats et, sert également comme support de communication entre les processeurs. Une unité d'interconnexion (IU) permet aux processeurs d'avoir accès à l'ensemble de l'espace de la mémoire partagée en lecture et en écriture.

D'un point de vue modèle, l'avantage de BSR est de permettre d'écrire des spécifications de manière très condensée. Notre objectif a été de nous approprier le modèle dans le but plus lointain d'arriver à caractériser les problèmes qui ont des solutions élégantes et « optimales » dans le modèle.

La puissance et l'élégance du BSR ont été démontrées par nombre d'algorithmes pour des problèmes tels que la somme préfixe [AG89], l'unicité d'un élément, les tris, le vecteur maximum, l'enveloppe convexe, le calcul de l'union d'intervalles [Akl89], le calcul du diagramme de Voronoi, l'intersection de polygone [SAG91, SA94], la mise en correspondance de parenthèses [Sto96], la génération de structure B-order d'arbre et la reconstruction d'arbres a partir de leur parcours, le décodage d'arbre binaire représenté en *bitstring-code*, la sous-séquence de somme maximale.

Tous ces algorithmes ont un temps d'exécution parallèle constant dans le modèle en utilisant n processeurs ! C'est ce qui fait l'originalité d'un grand nombre de problèmes traités dans le cadre BSR. Plus récemment Myoupo et Semé [MSct, MS99] ont appliqué le modèle BSR à la combinatoire des mots et aux problèmes de la plus grande sous suite commune à deux séquences, l'alignement de séquences, les répétitions qui sont autant de problèmes intervenant par exemple dans le décodage du génome.

Toutes les données reçues par une cellule mémoire sont sélectionnées selon un certain critère paramétrable, et sont ensuite réduites à une valeur calculée par un opérateur de réduction (typiquement, il s'agit d'un opérateur binaire associatif tel que $\max, \min, +, * \cdots$). C'est cette valeur calculée qui est finalement stockée dans la cellule mémoire concernée.

En plus de l'accès simultané des processeurs à une même cellule mémoire en lecture et écriture (CRCW PRAM), le modèle BSR autorise à chaque instant l'accès simultané en écriture dans toute la mémoire pour chaque processeur. Cette opération est appelée *diffusion* (*broadcast*). C'est l'élément central du modèle.

Une *instruction de diffusion* dans le modèle BSR est notée :

$$x_j := \underset{1 \leq i \leq n}{\mathfrak{R}} d_i | t_i \sigma l_j, \text{pour } 1 \leq j \leq n.$$

L'opération de diffusion est effectuée pour chaque j en parallèle. On interprète la notation comme suit : si le test σ portant sur t_i et l_j est vrai (t_i est le descripteur de diffusion du processeur p_i, et il est utilisé comme critère de sélection par $\sigma - l_j$ est la valeur limite du critère de selection σ sur les données de la diffusion de la cellule x_j).alors la donnée d_i est acceptée par la cellule mémoire x_j. L'ensemble des données acceptées par x_j est réduit à une simple valeur au moyen de l'opérateur $\mathfrak{R}$ et elle est stockée dans la cellule mémoire x_j. Si aucune donnée n'est acceptée par x_j alors l'ancienne valeur de x_j reste inchangée. Si une seule donnée est acceptée par x_j alors c'est sa valeur qui est stockée. Ces opérations sont effectuées en parallèle pour l'ensemble des j avec $1 \leq j \leq n$.

La vérification qu'une chaîne de parenthèses est bien parenthésée s'écrit en BSR de la manière suivante et en considérant que $\cup$, $\cap$, Σ symbolisent respectivement l'union, l'intersection et la somme :

$y_j := \Sigma b_j$
$x_j := \cup p_i$
si $y_j \neq 0$ ou $x_j < 1$ alors "chaîne non équilibrée"

Le problème de la mise en correspondance de parenthèses discuté précédemment s'écrit simplement comme suit en BSR :

si $l_j = '('$ alors $b_j := 1$ sinon $b_j := -1$
$p_j := \Sigma b_i | i \leq j$
si $b_j = -1$ alors $p_j := p_j + 1$
$p'_j := p_j - \frac{1}{j}$
$q_j := -1; t_j := 0; r_j := 0$
$q_j := \cap p'_i | p'_i < p'_j$
$t_j := \cap i | p'_i = q_j$
$r_j := \cup i | t_i = j$
si $l_j = ')'$ alors $m_j := t_j$ sinon $m_j := r_j$

Le problème du tri se résoud avec n processeurs, en temps constant avec trois opérations de diffusion de la façon suivante : la première diffusion donne le rand r_j de l'élément x_j dans le tableau d'entrée $x_1, \cdots, x_n$ c.à.d. le nombre d'éléments dans le tableau qui sont plus petit ou égal à x_j :

$$r_j = \Sigma 1 | x_i \leq x_j$$

3.2. Implémentation du circuit BSR. La première implémentation de BSR (en terme de circuiterie), dite naïve, a été proposée par Akl [AG89] où le réseau d'interconnexion (IU) a une taille de $\mathcal{O}(N, M)$, N étant le nombre de processeurs et M la taille de la mémoire disponible. Son avantage est sa simplicité qui permet d'attirer l'attention du lecteur sur la puissance de calcul du modèle plutôt que sur des détails fonctionnels. Le circuit proposé n'est pas performant mais il facilite une description simple et directe de l'opération de *diffusion*.

Une implémentation optimale en terme de circuiterie utilisée et de temps d'accès à la mémoire par les processeurs est proposée dans [AFL93]. Il s'avère que l'instruction de diffusion peut-être implémentée en $\mathcal{O}(\mathcal{T}(N, M))$ temps où $\mathcal{T}(N, M)$ est le temps requis pour un accès mémoire dans une PRAM ayant N processeurs et M emplacements mémoires. Si $\mathcal{T}(N, M) = \mathcal{O}(1)$, comme on le suppose pour la PRAM, alors les algorithmes BSR fonctionnent en temps constant.

Selim Akl et Lorraine Fava-Lindon [AFL93] montrent comment le BSR peut être implémenté en utilisant une unité d'interconnexion de taille $\mathcal{O}(M \log M)$, pour $N = \mathcal{O}(M)$. Ce résultat est optimal au vu de la taille minimale $\Omega(M \log M)$ d'une IU connectant M processeurs a M emplacements mémoires [Sha50]. Ces mêmes considérations s'appliquant à toutes les variantes de la PRAM, il en découle que le BSR ne requiert pas plus de ressources, de manière asymptotique, que le modèle PRAM le moins puissant, à savoir l'EREW-PRAM. Autrement dit, un circuit BSR n'est pas plus coûteux à construire qu'un circuit implémentant une PRAM de base.

4. Le problème du segment de somme maximale en BSR

Pour notre part, nous nous sommes intéressés au problème de la somme maximale à une et deux dimensions. Ces problèmes ont été introduits dans la littérature comme des problèmes de *recherche de motifs* dans des images. Ces problèmes peuvent aussi être vus comme des applications utilisant potentiellement la technique du calcul préfixe. En effet, en séquentiel le problème a comme idée sous-jacente que l'on passe d'un sous segment de taille k à un sous segment de taille $k+1$ en ajoutant l'élément visité (sous certaines conditions).

Soit $L = \{x_1, x_2, \cdots, x_n\}$ une séquence de n entiers. Soit maintenant la séquence $L' = \{x_i, x_{i+1}, \cdots, x_{i+(l-1)}\}$ avec $(1 \leq i \leq l \leq n)$ un sous segment de L de longueur l. L' est simplement un ensemble de l éléments consécutifs de L démarrant à la position i. Le problème du sous segment de somme maximale (Maximal Sum Subsegment Problem (MSSP)) consiste à trouver l'élément X de type L' d'une séquence L donnée

de somme maximale. En séquentiel, le problème se résout en temps $\mathcal{O}(n \log n)$ avec un algorithme qui tient sur quelques lignes :

```
   /* Le tableau b, de taille n, implante la séquence L */
1  s := 0;
2  h := 0;
4  c := 0;
4  tantque h != n faire
5       c := max(c+b[h],0);
6       s := max(s,c);
7       h := h+1;
8  fintantque
9  afficher(s);
```

La solution ci dessus est linéaire en temps et parait fortement séquentielle. Il n'est pas immédiat de dériver une solution parallèle.

Selim Akl [AG91] résoud le problème du sous segment de somme maximale d'une séquence stockée dans le vecteur `d[1..n]` en temps constant au moyen de deux étapes de réduction BSR plus une boucle parallèle. Quand nous examinons en profondeur l'algorithme précédent, nous constatons que celui-ci ne considère pas les propriétés et dérivations de l'algorithme séquentiel tel qu'il vient d'être présenté.

Nous avons proposé dans [BC97a] une autre approche du problème qui réalise la même complexité en temps constant mais qui s'écrit en deux lignes BSR au prix d'un nouvel opérateur BSR qui s'implémente sans surcoût de circuiterie par rapport à l'implémentation classique du circuit BSR. En fait, nous avons implémenté ou plus exactement simulé en BSR les lignes 5 et 6 de l'algorithme séquentiel.

Comme autre exemple de spécification et de dérivation d'algorithmes BSR, nous avons traité dans [BC97a] le problème du sous rectangle de somme maximum (c'est à dire le cas précédent mais à deux dimensions maintenant). L'algorithme utilise seulement 3 étapes BSR et une étape parallèle de calcul. La précédente solution faisait appel à 4 instructions BSR et 2 étapes parallèles. Quant à la place mémoire utilisée par notre algorithme nous utilisons 3 zones de mémoire de taille $\mathcal{O}(m^2 n)$, avec m, n définissant la taille du rectangle en entrée.

Tous ces exemples nous ont permis d'acquérir une expérience en matière de spécification de problèmes dans le modèle BSR à partir de problèmes intrinsèquement séquentiel. Nous avons apporté des améliorations aux précédents résultas connus en terme d'opérations de diffusion nécessaires pour résoudre le problème.

Ces études menées conjointement avec Laurent Bergogne que j'ai encadré pendant sa thèse, nous ont permis de mieux saisir le modèle et d'en comprendre les ressorts et les limites. En fait le BSR a une généralisation qui consiste à autoriser dans l'opération

de diffusion plusieurs critères de choix. Quel que soit le modèle BSR retenu, il serait intéressant d'aborder dans le futur la problématique générale qui consiste à caractériser l'ensemble des problemès qui ont des solutions en temps constant dans le monde BSR. L'algorithmique géométrique a été bien étudiée par Myoupo, Semé et Stojmenovic [JFMS02]. Cependant, à ce jour on ne sait pas s'il y a un algorithme BSR en temps constant pour réaliser le produit matriciel et qui utiliserait un nombre polynomial, en fonction de la taille du problème, de processeurs. Par contre nous avons vu que le tri est un problème << simple >> en BSR.

Quatrième partie

Le problème du tri

CHAPITRE 1

Introduction et motivations

BEAUCOUP DE TRAVAIL a été accompli dans le passé au sujet du tri en séquentiel afin d'optimiser les performances pour les processeurs RISC [Aga96], [NBC+94], [LPJN97] ou pour les processeurs disposant de peu de registres et de peu de cache [ACVW01], [RKU00] comme les processeurs embarqués (téléphones portables ou organiseurs personnels).

Pour notre part, nous avons montré dans un chapitre précédent comment tirer partie des différents organes architecturaux (cache, unités fonctionnelles de traitement) pour exploiter finement les processeurs actuels.

Trier en parallèle des enregistrements dont les clés proviennent d'un << ensemble ordonné >> est un problème étudié depuis de nombreuses années [Akl85]. Il est aussi souvent dit [Akl85] que de 25 à 50 pourcent de tout le travail effectué par les ordinateurs consiste à réaliser du tri.

Une des raisons est qu'il est souvent plus facile de gérer des données triées que non triées. Par exemple, le problème de la recherche d'un élément est traité beaucoup plus rapidement si la donnée de départ est triée.

Les études sur les algorithmes de tri en parallèle sont aussi guidées par des propriétés requises de l'entrée et de la sortie de l'algorithme afin de bien capter les nouveaux paradigmes architecturaux sous une large palette de situations. Par exemple, on veut s'intéresser au cas du *tri stable* (l'ordre des clés égales est préservé dans le vecteur de sortie) ou au cas où l'entrée est sous une distribution particulière (et non pas obtenue aléatoirement), par exemple une distribution cyclique.

De plus, le tri est particulièrement intéressant (et difficile) à cause des accès à la mémoire et aux motifs de communication irréguliers. A ce titre nous pensons que le tri parallèle est un bon problème pour évaluer les machines parallèles au niveau utilisateur. D'ailleurs il est présent dans plusieurs jeux de tests que l'on peut trouver sur le site de la NASA [1]. On discutera du cas des NAS parallèle pour le tri parallèle un peu plus tard.

Le projet Nowsort [2] à Berkeley est sans aucun doute le premier projet (1997) concernant le tri sur clusters. Il s'intéresse à un cluster homogène de processeurs SPARC et aux performances en temps uniquement. Dans ces travaux, on ne s'intéresse pas à la problématique de l'équilibrage des charges. Le projet `NowSort` rebondit actuellement autour du projet `Millenium` [3] qui se propose de développer un cluster de clusters

1. Voir : `http://www.nas.nasa.gov/`
2. Voir : `http://now.cs.berkeley.edu/NowSort/index.html/`
3. Voir `http://www.millennium.berkeley.edu/`

sur le campus de Berkeley afin de suporter des applications de calcul scientifique, de simulation et de modélisation.

On trouve aussi des industriels spécialistes du tri. Par exemple, la société `Ordinal`[4] distribue Nsort pour différentes plate-formes, généralement des machines multiprocesseurs avec beaucoup de mémoire vive. À titre d'exemple, en 1997, une Origin2000 avec 14 R10000 à 195Mhz, 7Go de RAM et 49 disques était capable de trier 5.3Go de données en 58s. Le but est de trier le plus possible de données en une minute.

Cependant, l'algorithme de tri a peu d'intérêt d'un point de vue scientifique : il s'agit de lire les disques, de trier en mémoire et d'écrire les résultats sur disque. C'est brutal, mais efficace ! On notera que la taille totale de la mémoire centrale est toujours supérieure à la taille du problème !

En 2000 l'algorithme Nsort de la société Ordinal a trié 12Go en 1 minute sur une SGI ORIGIN IRIX à 32 processeurs[5]. De même on a trié un tera-octet d'enregistrements de 100 octets en 17 minutes et 37 secondes[6] en utilisant la solution matérielle suivante : "SPsort was run on an RS/6000 SP with 488 nodes. Each node contains 4 332MHz 604 processors, 1.5GB of RAM, and a 9GB SCSI disk. The nodes communicate with one another through the high-speed SP switch with a bi-directional link bandwidth to each node of 150 megabytes/sec. Global storage of 6 TB of disk storage in the form of 336 RAID arrays is attached to 56 of the nodes. Besides the 56 disk servers, 400 of the SP nodes actually ran the sort program". On est loin du PC à 1000USD !

Les records 2001 sont pour le test Minute Sort (qui consiste à trier le plus possible en une minute) : 12GB en 60s, la méthode utiliséee étant Ordinal Nsort, sur une SGI 32 cpu, Origin IRIX. Pour le test Datamation (introduit en 1985) où il s'agit de trier un million d'enregistrements de 100 octets, le record est de 0.48s sur $32 * (2 * 550\text{Mhz})$ Pentium P3 équipés de 896MB de RAM, de $5 * 9\text{GB}$ de disques SCSI IBM et d'un réseau Gigabit Ethernet. En 1986 il fallait environ 26s sur un Cray. Les résultats complets sont disponibles sur la page :

`http://research.microsoft.com/barc/SortBenchmark/`

En 2002, le record pour Minute Sort est toujours obtenu avec la solution matérielle précédente.

4. Voir : `http://www.ordinal.com/`
5. Voir l'annonce sur `http://research.microsoft.com/barc/SortBenchmark/`
6. Voir `http://www.almaden.ibm.com/cs/gpfs-spsort.html`

1. Résultats obtenus

Pour notre part, l'axe principal de notre projet est l'exploitation du parallélisme sur clusters. Dans cette partie, l'application du tri est étudiée en particulier pour des clusters hétérogènes ce qui constitue l'originalité de notre travail. L'objectif à moyen terme est d'intégrer notre travail sur les tris disques dans des applications de bases de données (l'opération de jointure peut s'implémenter avec un tri), de fouille de données comme par exemple l'exploitation des états des machines connectées à un système de jachère. Dans ces systèmes, un utilisateur peut mettre en commun ces ressources (CPU, disques...) afin de constituer un super ordinateur (de calcul, de stockage...)

Nous fournissons une étude exhaustive du tri parallème en mémoire et sur disques pour les cas de clusters homogènes et de clusters hétérogènes. Nous couvrons un spectre très large de systèmes

Rappelons que les *clusters hétérogènes* sont définis comme les clusters habituels (réseau de communication unique, même système d'exploitation installé, même processeur, même montant de place mémoire et de place disque) mais les processeurs peuvent être à différentes vitesses. Il s'agit donc d'un premier niveau d'hétérogénéité.

La Figure 1 classifie les principaux résultats obtenus et fait référence aux articles de la bibliographie. D'autres résultats sont en pré-publications. Sur la Figure 1, la classification des résultats se fait selon deux axes orthogonaux : les résultats obtenus dans le cas homogène vis à vis du cas non homogène pour le premier axe et les résultats obtenus pour le cas en mémoire ("in-core") vis à vis du cas sur disque ("out-of-core").

Nous avons apporté les contributions suivantes (voir la Figure 1) : la publication [Cér02] a été réalisé pour la conférence IPDPS'2002 et elle présente notre première approche pour trier sur disque en milieu hétérogène. Il s'agit de la mise en œuvre d'un algorithme façon PSRS (Parallel Sorting by Regular Sampling). Pour le tri hétérogène en mémoire nous avons développé deux techniques distinctes [CG00e, CG00f]. Elles ont été publiées respectivement pour Cluster'2000 et HiPC'2000. La publication [CG02] est un article de synthèse sur le tri en mémoire et en hétérogène. Pour les tris homogènes avec des distributions de données particulières, par exemple avec une distribution avec << beaucoup >> de doublons nous avons obtenu les résultats publiés dans [CG99b]. Enfin, les références [CG01, CG00d] concernent l'évaluation d'une grappe de PC au moyen du tri.

Nous avons réalisé des implémentations qui valident nos approches et qui montrent clairement le bien fondé des techniques développées, en particulier pour l'équilibrage des charges de chaque processeur. Dans les implémentations pour clusters hétérogènes, un vecteur contient les vitesses relatives des processeurs les uns vis à vis des autres.

Notez que les algorithmes à droite du trait vertical sur la Figure 1 peuvent se transformer en algorithmes << homogènes >> en faisant en sorte que le vecteur de performance (dans les implantations) ne contienne que des valeurs 1.

Par rapport aux préoccupations des industriels nous garantissons des propriétés sur les temps d'exécution et sur l'équilibrage des charges. Nous apportons donc des bornes théoriques sur l'équilibrage des charges des processeurs. Des expérimentations confirment les qualités de nos codes. Les implémentations sont réalisées avec des bibliothèques MPI (Message Passing Interface) ou BSP (Bulk synchronous Parallel model) standards. Elles sont disponibles en ligne.

Il est aussi très important de noter que contrairement aux implémentations << record du monde >> proposées qui utilisent toute la mémoire RAM disponible (on sous entend alors que le cluster est dédié), nous avons cherché à limiter et à borner au mieux l'usage des ressources du système. Ainsi, on verra que les tris disques hétérogènes que nous proposons sont de << vrais >> tris disques puisqu'ils utilisent de l'ordre de 32K octets de mémoire RAM et aussi un petit nombre de fichiers intermédiaires.

À moyen terme, comme nous l'avons signalé un peu plus haut, nous pensons intégrer nos tris dans un système de jachère de calcul. Un tel système permet d'exécuter des applications sur des machines que l'on détecte inactives, par exemple des PC

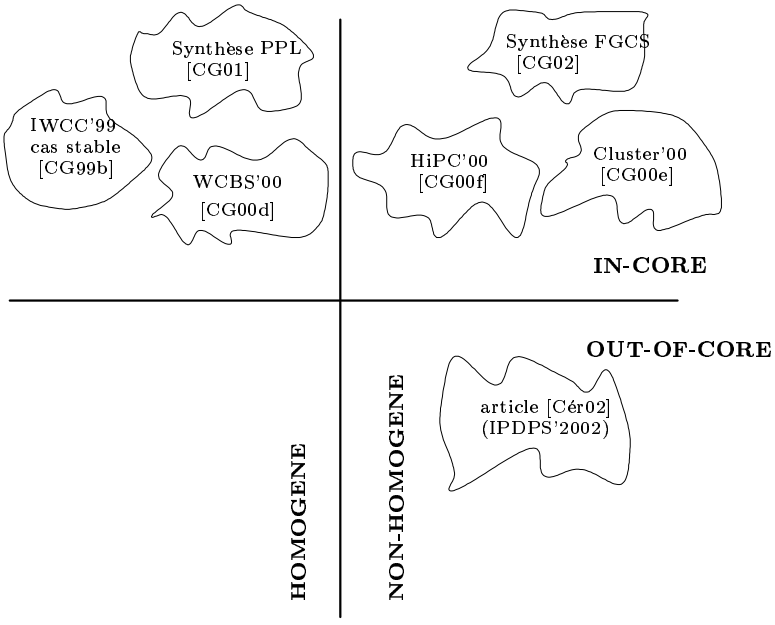

FIG. 1: *Une vue schématique des résultats obtenus pour le tri.*

chez un particulier. Il sera alors impératif de n'utiliser que modérément la RAM d'un utilisateur qui participe au système de jachère, celui ci ne souhaite pas que sa participation dégrade les performances des applications qu'il fait tourner sur sa machine.

Par ailleurs, à court terme nous allons utiliser notre tavail pour valider la bibliothèque `Read`^{2} [CRU02] développée à Amiens et qui vise en particulier à effectuer des transferts rapides de disques à disques dans des clusters de PC.

2. Équilibrage de charge par échantillonnage

Nous décrivons maintenant l'ensemble des notions dont nous allons avoir besoin dans cette partie.

À propos de métriques de performances étudiées, le temps parallèle d'exécution n'est pas le seul centre d'intérêt en parallélisme. Blelloch [BLM91] définit la *métrique d'expansion* (sublist expansion metric) comme étant le ratio entre la taille de la plus grande sous liste traitée par un processeur à un instant de l'algorithme et la taille moyenne attendue. Autrement dit, cette métrique rend compte de l'équilibrage des charges : idéalement, on attend une valeur 1. Dans ce cas l'équilibrage est parfait.

C'est cette métrique d'équilibrage que nous utilisons. En complément nous étudions également les temps d'exécution et nous donnons aussi, assez souvent l'écart type sur les temps d'exécution.

On rappelle également qu'un tri est *stable* quand l'ordre des clés égales est préservé dans le vecteur de sortie.

Le critère de stabilité est important dans les bases de données. Si l'on fait un tri sur le nom d'une personne et que l'on a dans la base avant le tri, trois personnes de même nom P_1, P_2, P_3 rangées selon l'ordre des indices, alors on souhaite retrouver après le tri que les personnes soient rangées dans le même ordre (P_1 avant P_2 avant P_3) car la base était peut être initialement donnée à partir d'un double classement, sur le nom puis sur le prénom, par exemple. On doit donc préserver cette propriété.

Nous avons développé une interface qui permet de générer à la fois des entrées aléatoires ou des entrées sous des distributions particulières, par exemple une *distribution cyclique* (le vecteur en entrée est $1, 2, 3, 1, 2, 3 \cdots$). Ainsi nous avons tous les moyens pour vérifier que notre travail à une portée générale : les algorithmes proposés se comportent bien indépendamment de la distribution initiale des données. C'est une propriété remarquable.

Les algorithmes proposés font de *l'échantillonnage* de *pivots* [RV87] : il s'agit d'isoler dans l'entrée des échantillons qui vont servir à partitionner le vecteur en entrée en sous-vecteurs. Les valeurs des échantillons s'appelent des *pivots*.

Le principe général des tris est alors le suivant : après le partitionnement, on redistribue les partitions aux processeurs (selon un certain ordre) et au final les processeurs trient ou fusionnent selon le cas les différentes partitions reçues.

Pour assurer de bonnes qualités d'équilibrage il faut faire en sorte que les partitions créées soient de taille approximativement égales. On n'est pas obligé d'avoir exactement des partitions de tailles égales, tout dépend de la tolérance que l'on s'autorise par rapport au critère d'équilibrage.

On dit que l'échantillonnage est *régulier* lorsque les pivots sont pris à des intervalles réguliers dans l'entrée.

On dit qu'un algorithme fait du *sur-échantillonnage* quand on choisi dans l'entrée plus de pivots qu'une certaine borne donnée. Cette technique va permettre de mieux assurer la métrique d'équilibrage mais elle à un coût que nous maîtrisons.

CHAPITRE 2

Stratégies d'échantillonnage pour trier en parallèle

LA STRATÉGIE d'échantillonnage (de pivots) est étudiée dans ce chapitre, principalement pour le cas homogène. Intuitivement il s'agit d'isoler dans le vecteur d'entrée des pivots qui le partitionnent en segments de tailles égales. Cette phase permet de sélectionner des pivots de telle sorte qu'entre deux pivots consécutifs, il y ait le même nombre d'objets. Après une phase de redistribution des données selon les valeurs des pivots, il ne reste plus à trier localement les valeurs. Nous expliquons en quoi cette stratégie est intéressante à priori pour les architectures de clusters et nous synthétisons les principaux résultats mathématiques connus de l'échantillonnage de pivots. Nous expliquons ensuite pourquoi la technique PSRS (Parallel Sorting by Regular Sampling) [RV87, RV83, SS92, LLS+93, HJB96] est pertinente pour évaluer les performances des clusters au niveau de « l'application ». À partir de là nous expliquons quelles sont les techniques candidates pour trier dans un contexte non homogène.

1. Les familles de tri

On reconnaît maintenant qu'il y a deux grandes approches génériques pour trier en parallèle. Ces deux approches se sont révélées pertinentes en pratique : les algorithmes implémentés selon ces techniques sont efficaces sur une large gamme d'architectures.

Les algorithmes par fusion (MERGE-BASED) : ils suivent le schéma suivant : (1) chaque processeur contient initialement une portion de la liste à trier (2) on trie les portions et on les échange entre tous les processeurs (3) on fusionne les portions en une ou plusieurs étapes.

Les algorithmes « diviser pour régner » (QUICKSORT-BASED) : ils suivent les étapes suivantes : (1) les listes initiales non triées sont partitionnées progressivement en plus petites sous listes définies par un choix de valeurs pivots (2) on trie les sous listes dont les processeurs sont responsables. Il n'y a pas besoin de fusionner.

Nous nous sommes intéressés uniquement à l'approche par fusion et pour cette approche uniquement aux algorithmes avec une seule étape de fusion. Cela se justifie principalement parce qu'une seule étape de fusion engendre très peu de communications et donc peu de pénalités dues par exemple aux latences de communications sur des réseaux locaux de type Fast-Ethernet. Une telle technique est donc bien adaptée à ce qui est requis actuellement en programmation par des envois de messages : un nombre limité de messages longs. De plus, quand les éléments sont déplacés d'un processeur à un autre ils vont nécessairement à leur destination finale.

Les références [Qui89, LLS+93, SS92, LS94] appartiennent à la catégorie des *algorithmes de tri parallèle à une seule phase de fusion* (one step merge based algorithms); les références [Bat68, BR93, BH82, Col88, Lei84, NS82, Pla89, TB93] à celles des *algorithmes à plusieurs phases de fusion* (multi steps merge-based algorithms).

La technique la plus intéressante à notre avis est *l'échantillonnage régulier* tel que nous allons le voir maintenant pour le cas du tri stable. Elle surpasse sur beaucoup de points tous les algorithmes précédemment cités.

2. Résultats obtenus pour le tri stable

Dans le domaine public, on trouve une implémentation[1] BSP qui suit la technique du tri selon un échantillonnage de pivots.

On peut faire remarquer que l'implémentation ne donne pas un tri *stable* car un tri rapide est utilisé en sous main et un tri rapide n'est pas stable. Les résultats que nous avons obtenus dans [CG99b] considèrent le problème du tri stable selon une stratégie d'échantillonnage.

Nous avons montré [CG99b] que différentes approches étaient possibles pour préserver l'ordre des clés égales (utilisation d'un codage comme celui fait en [Sto96], utilisation d'un tri ShellSort à k partitions). Quand on utilise un tri stable (ShellSort) comme brique de base pour trier localement il faut encore faire très attention à ce que les permutations des données entre processeurs préservent l'ordre relatif des éléments entre eux.

Pour rendre des clés entières uniques, le codage que nous avons implanté est celui présenté dans [Sto96]. Il est basé sur la propriété suivante : soient deux entiers r_i et r_j rangés aux positions i, j respectivement. On a alors :

$$r_i = r_j \wedge i > j \Rightarrow r_i - \frac{1}{i} > r_j - \frac{1}{j}$$

La conséquence est qu'au lieu de trier sur les entiers proprement dit on trie sur les valeurs $r_i - \frac{1}{i}$ qui sont maintenant uniques. Cependant, le codage augmente par un facteur 2, au moins, l'espace nécessaire au traitement et l'échange se déroule sur deux fois plus d'octets. En pratique, c'est une limitation sérieuse. Le seul cas où nous avons pu observer que le codage était meilleur que l'utilisation d'un ShellSort (qui est stable) était le cas où une des valeurs étaient dupliquées plus de n/p.

Jusqu'à présent, la seule référence qui traitait de la stabilité est, à notre connaissance, les papiers de JàJà [HJ99, HJ97]. Les auteurs affirment que "their algorithms can be easily implemented as a stable sort"... mais ils ne le font pas !

1. Le code est dû à de Ronald Sujithan (voir le lien : `http://www.comlab.ox.ac.uk/oucl/users/ronald.sujithan/`)

3. Efficacité de l'échantillonnage régulier

Soit n la taille du problème et p le nombre de processeurs. Les quatre étapes canoniques de l'algorithme du tri par échantillonnage PSRS (Parallel Sorting by Regular Sampling) sont :

Étape 1: chaque processeur commence par trier localement ses n/p données, puis ils sélectionnent p pivots qui sont rassemblés sur le processeur 0 (on passe les détails concernant le choix proprement dit qui se fait à intervalles réguliers d'où le nom) ;

Étape 2: tri local des p^2 pivots ; on en garde $p-1$ (ils sont sélectionnés tous les $ip+p/2, (1 \leq i \leq (p-1)$ intervalles) ; Tous les pivots sont diffusés à tous les autres processeurs.

Étape 3: chaque processeur produit p partitions en fonction de ses $p-1$ pivots et envoie la partition i (marquée par les pivots k_i et k_{i+1}) au processeur i ;

Étape 4: les processeurs ont reçu des partitions triées ; ils les fusionnent.

La Figure 1, que l'on trouve à la page 80, donne un exemple de déroulement d'un tel algorithme. Il s'agit de l'algorithme de Shi [SS92]. Sur cette Figure, on remarque le tri initial (Phase 1), local sur chaque processeur, puis le choix des pivots se fait à intervalles réguliers (Phase 2) ; la centralisation des pivots et leur tri sont représantés à la Phase 3 et enfin on a la redistribution des entiers selon les valeurs des pivots (Phase 4).

Ce qui fait la force de ce type algorithme c'est qu'en échantillonnant les portions triées sur tous les processeurs, nous pouvons considérer que l'ensemble des données est << représenté >> dans le vecteur des pivots et que l'information sur l'ordre des données est capturée.

Il a été montré par Shi [SS92] que le coût de calcul de PSRS atteint la borne d'optimalité en $\mathcal{O}(n/p \log n)$. Pour obtenir ce résultat, il faut s'assurer de l'unicité des données. Dans ce cas, on peut même montrer que, sous l'hypothèse que $n > p^3$, PSRS garantit un équilibrage de charge qui diffère de l'optimal par une constante 2 , ce qui est très bon ! C'est la meilleure borne connue. Cela signifie qu'à l'étape 4 de l'algorithme présenté plus haut, aucun des processeurs ne trie plus de deux fois ce qu'il avait au départ. En pratique, la constante est même assez proche de l'optimal. Nous présenterons quelques résultats expérimentaux plus loin dans le texte. Ils valident l'approche.

Par ailleurs, les implémentations de la techniques que nous avons réalisées pour le cas stable (ainsi que le cas hétérogène qui sera vu plus loin ainsi que le cas disque) confortent les résultats pratiques précédemment connus. L'échantillonnage de pivots offre un *cadre générique* pour trier de manière efficace. Nous l'avons utilisé et généralisé pour trier en milieu hétérogène.

3.1. Trier avec des duplicats. Les auteurs de la référence [LLS+93] s'intéressent au problème des doublons pour PSRS et ils montrent que la présence de doublons augmente la borne sur l'équilibrage de manière *linéaire*: si d représente la clé avec le plus grand nombre de doublons et que la borne supérieure soit $U = 2.n/p$, alors la borne supérieure avec d doublons devient $U + d$. En pratique, le problème est de trouver la valeur de d telle que le terme $2.n/p$ ne domine pas le terme d. De manière asymptotique il faut $d = \mathcal{O}(n/p)$ doublons.

Nos codes sont conçus pour générer des duplicats ce qui nous a permis de vérifier expérimentalement les résultats théoriques précédents. Les implémentations connues des tris parallèles autres que PSRS ne permettent pas de les évaluer lorsqu'il y a beaucoup de duplicats. La seule distribution de données considérée est généralement une distribution aléatoire (par une méthode congruentielle). Nous offrons, dans nos code, la possibilités de les évaluer à partir de 8 distributions initiales différentes, et en particulier avec des doublons.

4. Choix d'implantations

Ce paragraphe précise les choix d'implémentation effectués en matière de sélection des pivots pour PSRS (Parallel Sorting by Regular Sampling). Nous décrivons également les alternatives implémentées qui visent à ne pas réaliser de tri initial. Nous donnons des résultats expérimentaux qui permettent de sélectionner la meilleure technique (en terme de temps de calcul).

Jusqu'à très récemment, les processeurs qui triaient selon la technique PSRS commençaient par un tri séquentiel, typiquement QuickSort. Cette phase, comme nous l'avons déjà dit, permet de sélectionner des pivots de telle sorte qu'entre deux pivots consécutifs, il y ait le même nombre d'objets. C'est ce qui permet d'assurer la borne sur l'équilibrage. Même Shi et Schaeffer, les inventeurs de la technique disent, dans le papier originel [SS92] que "It appears to be a difficult problem to find pivots that partition the data to be sorted into ordered subsets of equal size without sorting the data first". Il se trouve qu'une optimisation peut être faite en pratique si l'on considère la notion de *quantile* telle qu'elle est présentée, dans le contexte du tri dans [FS99]. Rappelons que les quantiles sont l'ensemble des « points de coupure » qui divisent un échantillon en groupes contenant le même nombre d'observations. Les *p-quantiles* sont l'ensemble ordonné A de taille n qui sont les $p-1$ éléments de rang $n/p, 2n/p, \cdots, (p-1)n/p$, qui coupent A en p parties d'égale taille.

Le *problème de la recherche des p-quantiles* consiste à déterminer les « éléments de coupure » de A. Remarquons que le problème est lié à la *recherche du meilleur* (le maximum, par exemple) comme suit : le médian - le $\lceil n/2 \rceil$ème meilleur - est donné par un appel à la procédure `find(t,0,n-1,n/2)` ($n = 2^k$) où `find` est la procédure

qui trouve l'indice du ième plus grand élément du vecteur t entre les bornes 0 et $n-1$.

Nous avons implanté un algorithme probabiliste qui donne le meilleur en temps séquentiel $\mathcal{O}(n)$ en moyenne. En pratique, comme le nombre de pivots à trouver n'est pas du même ordre de grandeur que les tailles des problèmes à traiter (ce nombre est beaucoup plus petit), on peut espérer implémenter la recherche des pivots dans un temps « quasi-linéaire » au lieu de $\mathcal{O}(n \log n)$ pour un QuickSort. Remarquons que les auteurs de [DNS91] font référence à la notion de quantiles (pour développer un tri sur disque) mais à aucun moment ils capitalisent sur la remarque que l'on vient de faire.

Finalement, nous avons implémenté les étapes 1 et 4 de l'algorithme PSRS, à partir des deux choix possibles suivants selon que l'on veut être :

conforme à l'algorithme original: Dans ce cas on implante l'étape 1 avec un QuickSort et l'étape 4 avec un merge. C'est ce qui est fait dans l'implantation de Sujithan pour la BSPLib ;

non conforme à l'algorithme original: mais conforme à l'esprit du papier originel [SS92]. Dans ce cas, l'étape 1 est implantée avec une recherche de quantile (qui n'offrent pas en sortie un vecteur trié) puis un QuickSort pour l'étape 4. L'idée est qu'on espère que le coût d'une recherche de quantiles plus un quicksort soit inférieur dans la pratique au coût d'un quicksort plus une fusion.

Examinons nos résultats à la Table 1. Ils donnent les temps d'exécution de notre implantation qui respecte l'idée originale ainsi que les variances et les écart types des temps d'exécution.

La Table 2 est notre implantation avec une recherche de quantiles plus un QuickSort. Nos expériences sont faites sur un cluster de quatre processeurs Alpha (EV56 21164, 500Mhz) interconnectés par fast-ethernet.

TAB. 1: Configuration: 4 Alpha 21164, 500Mhz, 256Mb of RAM, 4Mo cache fast Ethernet

#PROCs	INPUT SIZE	MEAN (s)	VARIANCE	ST. DEVIATION
4	131072	0.13695	0.00233	0.04834
4	262144	0.53295	0.00650	0.08068
4	524288	1.06904	0.02577	0.16055
4	1048576	1.64023	0.07724	0.27792
4	2097152	2.69613	0.03101	0.17610
4	4194304	5.44492	0.09799	0.31304

TAB. 1: Configuration: 4 Alpha 21164, 500Mhz, 256Mb of RAM, 4Mo cache fast Ethernet

4	8388608	11.23528	0.28563	0.53445

Execution time metrics for benchmark 0
Algorithm used: original PSRS
– code: `psrs_pub_merge_bench.c` –

Nous concluons que dans la pratique il n'y a pas vraiment de gagnant (en terme de gain en temps) sauf que l'implémentation par quantile, qui ne nécessite pas de merge est moins gourmande en mémoire, de l'ordre de n, que l'implantation avec un merge. Par ailleurs, ces résultats expérimentaux sont confirmés lorsqu'on exécute les codes sur un cluster de huit processeurs Celeron.

TAB. 2: Configuration: 4 Alpha 21164, 500Mhz, 256Mb of RAM, 4Mo cache, fast Ethernet

#PROCs	INPUT SIZE	MEAN (s)	VARIANCE	ST. DEVIATION
4	131072	0.13828583	0.00189696	0.04355412
4	262144	0.51180058	0.01442428	0.12010115
4	524288	1.01821333	0.03200934	0.17891154
4	1048576	1.61490733	0.06455982	0.25408625
4	2097152	2.55807533	0.14855640	0.38543015
4	4194304	5.42543410	0.12039131	0.34697451
4	8388608	11.35150307	0.29101192	0.53945521

Execution time metrics for benchmark 0
Algorithm used: modified PSRS with quantile search
– code: `psrs_pub_quantile_bench.c` –

5. Techniques de sur-échantillonnage

L'échantillonnage régulier de pivots n'est pas la seule technique utilisée, dans le cas homogène, pour assurer des propriétés d'équilibrage de charge des processeurs. Par exemple, l'algorithme de Li et Sevcik [LS94] est un sérieux candidat. L'idée est d'éviter le tri initial des données et de sélectionner, de manière aléatoire « suffisamment de pivots » (d'où le terme de sur-échantillonnage) de telle sorte que ces pivots partitionnent l'entrée en morceaux de tailles (presque) égales. L'algorithme présenté en [LS94] se présente alors sous la forme canonique suivante :

ALGORITHME 1 (Texte tel qu'on peut le trouver dans [LS94]).

Step 1: *initially, processor* i *has* l_i*, a portion of size* n/p *of the unsorted list* l*;*

Step 2 (selecting pivots): *a sample of* $p.k.s$ *candidates are randomly picked from the list, where* s *is the oversampling ratio and* k *the over partitioning ratio. Each processor picks* $s.k$ *candidates and passe them to a designated processor. These candidates are sorted and then* $p.k-1$ *pivots are selected by taking (in a 'regular way')* $s^{th}, 2.s^{th}, \cdots, (pk-1)^{th}$ *candidates from the sample. The selected pivots* $d_1, d_2, \cdots, d_{pk-1}$ *are made available to all the processors;*

Step 3 (partitioning): *since the pivots have been sorted, each processor performs binary partitioning on its local portion. Processor* i *decomposes* l_i *according to the pivots. It produces* pk *sublists per processor denoted* l_{ij} *where* i, j *stands for two consecutive pivots (except for the initial an final case). A sublist* S_j *is the union of* l_{ij} *with* i *ranging over all processors. There is* pk *sublists.*

Step 4 (building a task queue and sorting sublists): *Let* $T(S_j)$ *denotes the task of sorting* S_j*. The size of each sublist can be computed:*

$$|S_j| = \sum_{i=1}^{p} | l_{ij} |$$

Also the starting position of sublist S_j *in the final sorted array can be calculated:*

$$\sigma_j = 1 + \sum_{h=1}^{j-1} | S_h |$$

A task queue is built with the tasks ordered from the largest sublists size to smallest. Each processor repeatedly takes one task $T(S_j)$ *at a time from the queue. It processes the task by (a) copying the* p *parts of the sublist into the final array at position* σ_j *to* $\sigma_j + | S_j | -1$*, and (b) applying a sequential sort to the elements in that range. The process continues until the task queue is empty.*

Les auteurs de l'article [LS94] reconnaissent que la métrique d'expansion décroît lorsque s croit ce qui est conforme à l'intuition. Cependant ils observent une métrique d'expansion encore de $1,3$ pour $p > 64, s = 128$. Ceci signifie que certains processeurs reçoivent 25% de travail en plus ou en moins par rapport à la moyenne attendue. Comparativement, PSRS (Parallel Sorting by Regular Sampling) procure une métrique d'expansion très proche de 1 en pratique.

Li et Sevcik donnent dans leur article des informations très pratiques (voire empiriques) pour calibrer au mieux les paramètres de leur modèle. Par exemple, ils proposent de fixer le paramètre k avec la valeur $\log p$ (p est le nombre de processeurs) et de fixer la valeur de s à 4. Expérimentalement les résultats sont meilleurs avec ce couple de valeurs mais ils n'ont aucun moyen de montrer formellement que << ça marche >> quelque soit p.

Enfin on doit aussi dire que le problème du tri avec des doublons n'est pas exploré dans le travail de Li et Sevcik [LS94]. C'est un inconvénient vis à vis du tri PSRS (voir plus haut).

6. Choix aléatoire des pivots

Revenons un instant sur les résultats connus en matière de choix de pivots de manière aléatoire. Il ne s'agit pas de résultats établis pour des clusters homogènes ou hétérogènes ni pour des cas sur disques ou en mémoire. Curieusement, c'est un article paru dans une conférence relative aux bases de données. Il s'agit du travail de S. Seshadri et J.F. Naughton [SN92, LNSS93]. Ce travail est centré sur l'échantillonnage parce qu'il sert dans l'obtention de bonnes propriétés d'équilibrage, qu'il sert aussi pour une bonne utilisation des opérations d'entrée/sortie et pour l'estimation des quantiles. Dans les articles cités les auteurs montrent que la technique où l'on forme un échantillon (de pivots) à partir de k échantillons de taille s (i.e. on sélectionne s pivots sur chacun des k processeurs - cette technique est connue sous le nom d'échantillonnage stratifié) ne réduit pas la pertinence de l'échantillon comparativement à la technique qui consiste à échantillonner $k.s$ valeurs en considérant un seul échantillon (on dit alors qu'il s'agit d'un échantillon simple).

Un premier résultat présenté dans le papier [SN92] est une adaptation d'un résultat de Gonnet [Gon81] :

LEMME : 4. *Considérons un échantillon simple de taille n pris à partir de k sites, si $n/k = \mathcal{O}(1)$ alors, de manière asymptotique, certains sites fourniront*

$$\frac{\ln k}{\ln \ln k}$$

valeurs d'échantillons.

On peut par exemple calculer le cas $n = 1000, k = 10$ sites et on trouve que le nombre maximum attendu de valeurs qui proviennent d'un même site est 120 et que le nombre minimum attendu de valeurs qui proviennent d'un même site est de 80. Il y a donc un grand écart entre ces valeurs ce qui se traduit par : certain site sont sur-représentés, d'autres sont sous représentés. Il convient donc d'améliorer la borne (l'asymptote). On ne peut pas raisonablement développer un code de sur-partionnement à partir de cette idée car elle conduirait à de très mauvaises performances vis à vis de l'équilibrage.

Un autre résultat de l'article de Gonnet est particulièrement intéressant lorsqu'on travaille avec les disques parce qu'on lit une page (un bloc) et non pas une seule valeur :

THÉORÈME : 4. *Soit* V_{page} *la variance de l'estimation obtenue par échantillonnage simple de* n *pages (on considère que l'échantillon est formé de toute les valeurs lues par les instructions d'entrées sorties) et soit* V_{tuple} *la variance de l'estimation obtenue à partir d'un échantillonnage simple de* n *tuples (on considère ici que l'échantillon est formé de* n *valeurs chacune prise dans une page (lue par une opération d'I/O)) alors* $V_{page} \leq V_{tuple}$.

Ainsi, à chaque fois que l'on fait une opération d'entrées sorties, on ramène au niveau du processeur une page avec $k > 1$ valeurs. Le résultat précédent dit qu'il est possible de garder toutes les valeurs contenues dans la page (dans le bloc lu) plutôt que de garder uniquement celle qui était pointée. Bien sûr le théorème dit que cela se passe comme cela si on lance beaucoup d'expériences mais il ne dit pas que c'est toujours le cas. Nous sommes en cours d'expérimentation d'une technique de sur-partionnement pour réaliser du tri disque. Le résultat précédent sera envisagé afin de recueillir les pivots.

Le résultat du Théorème 4 reste vrai lorsqu'on fait de l'échantillonnage stratifié.

Enfin, le paragraphe 7 de [SN92] est spécifiquement dédié à l'estimation des quantiles qui, nous l'avons vu, ont un lien avec le tri parallèle. Les auteurs s'intéressent explicitement à savoir si une des partitions (donnée par les quantiles) aura une taille $\alpha n/p$ pour un $\alpha > 1$. Le terme α peut être vu comme la métrique d'équilibrage de charge. Les résultats présentés sont meilleurs que ceux de Blelloch [Ble93] qui servent en particulier dans le tri parallèle de [DNS91]. Rappelons la borne obtenue par Blelloch. Dans [Ble93] les auteurs choisissent les pivots comme suit : premièrement chacun des p sites partitionne ses n/p données en s groupes de taille $n/(ps)$ et sélectionne alors aléatoirement une seule clé (pivot) dans chaque groupe. Ils montrent alors que pour $\alpha > 1$, la probabilité qu'une des partitions soit plus grande que $\alpha n/p$ est

$$n \exp \frac{-(1 - 1/\alpha)^2 \alpha s}{2}$$

Ce résultat est à comparer par exemple avec le théorème 13 de [SN92] qui affirme que la probabilité qu'une des partitions reçoive plus de $\alpha n/p$ données est inférieure ou égale à :

$$k\alpha^s \left(\frac{k - \alpha}{k - 1}\right)^{k.s-s}$$

Les résultats que nous avons obtenus en milieu hétérogène et pour la technique du sur-échantillonnage considèrent en fait une généralisation de la méthode de sélection de Li et Sevcik [LS94].

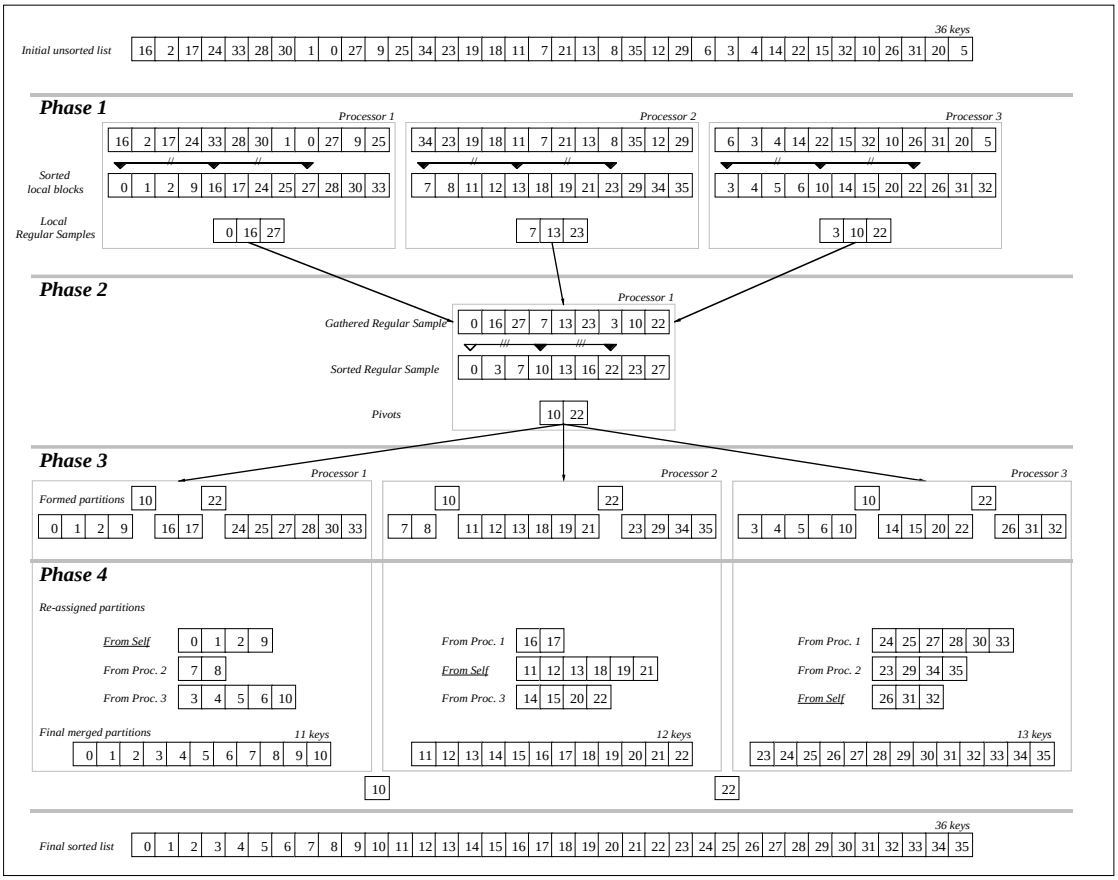

FIG. 1: *Déroulement de l'algorithme PSRS [SS92]*

CHAPITRE 3

Exemples de résultats expérimentaux obtenus

POUR L'IMPLANTATION DES ALGORITHMES DE TRI en parallèle, le cas de PSRS est particulièrement attractif puisque sous l'hypothèse que $n > p^3$, on peut borner par $2.n/p$ la quantité de données qu'un processeur reçoit dans la dernière phase de l'algorithme. La constante 2 est excellente pour la pratique. En conséquence, un petit nombre d'expériences est nécessaire à l'obtention d'une bonne variance sur les temps d'exécution. De ce point de vue, PSRS est donc un bon programme pour mesurer les performances d'un système.

Il s'agit maintenant d'expliquer comment nous avons construit nos tests et nos plans d'expérimentation. Le tri est en effet présent dans de nombreux tests mais les implémentations proposées ne sont pas toutes convainquantes en particulier vis à vis de l'équilibrage des charges garanti et du nombre de distributions de données qui peuvent être instrumentalisées.

Toutes nos expérimentations se font avec 30 exécutions pour une même taille de problème. Une liste prospective de tests pour mesurer les performances des clusters peut être trouvée sur une page de l'université de Turin (à l'adresse :

`http://www.di.unito.it/~mino/cluster/benchmarks/` .

1. Les NAS parallèles

Parmi les tests mentionnés à partir de ce lien, on peut citer les "NAS parallel benchmark 2.3" et trouver une implémentation d'un tri. Il s'agit du test IS (Integer Sorting) qui relève des algorithmes d'échantillonnage qui s'abstiennent de trier localement les données dans la première phase de l'algorithme générique.

Nous n'avons pas choisi de travailler avec IS parce qu'il repose en grande partie sur un niveau empirique qui ne nous permet pas, par exemple, d'estimer à priori l'équilibrage de charge des processeurs. Par exemple dans cette phrase tirée du code source des NAS on ne sait pas comment calibrer automatiquement la taille de certains vecteurs :

```
/*****************************************/
/* On larger number of processors, since */
/* the keys are (roughly) distributed,   */
/* the first and last processor sort keys*/
/* in a large interval, requiring array  */
/* sizes to be larger.                   */
/*****************************************/
```

Le nombre de pivots nécessaires à de bonnes performances est fixé à partir de l'expérience des programmeurs par des directives comme suit :

```
#if CLASS == 'S'
#define  TOTAL_KEYS_LOG_2    16
#define  MAX_KEY_LOG_2       11
#define  NUM_BUCKETS_LOG_2   9
#endif
```

Autrement dit, pour certaines tailles de problèmes on va garantir plus ou moins un bon équilibrage mais pour d'autres tailles (en gros au delà de tailles de 8M) on ne peut plus rien dire ! Le code IS des NAS parallèle 2.3 n'est pas totalement satisfaisant pour la plus grande généralité du programme et pour tester de manière très large des clusters. Enfin, la seule utilisation possible du test se fait avec une distribution aléatoire des données. On n'a pas le loisir de « jouer » sur la distribution pour tester la robustesse de l'algorithme dans différentes situations, contrairement à nos codes qui offrent 8 distributions différentes.

2. Palette de tests

Enfin, comme nous l'annoncions précédemment, chacun de nos codes peut s'utiliser selon 8 distributions différentes des données. Nous avons réutilisé partiellement le travail de Bader, Helman et JàJà[1] que nous avons fusionné avec le notre. Les huit tests présents sont :

benchmark 0: : il y a un appel à la procédure `fill_bench_i_N()` i.e. remplissage du vecteur en entrée grâce à un générateur pseudo-aléatoire uniforme qui retourne un nombre entre $(0, 1)$ en utilisant une méthode congruentielle du type $x_{k+1} = a.x_k (\text{mod } 2^{46})$

benchmark 1: : appel à la procédure `fill_bench_i_C()` i.e. remplissage du vecteur d'entrée avec les valeurs $k, 2k, 3k, \cdots, pk, k+1, 2k+1, \cdots, pk+1, k+2, 2k+2, \cdots pk+2, k+3, 2k+3, \cdots$;

benchmark 2: : remplissage avec la valeur 1 ;

benchmark 3: : appel à la procédure `fill_bench_i_same()` i.e. remplissage avec les valeurs $0, 1, \cdots, k, 0, 1, \cdots k, 0, 1, \cdots k, \cdots$ cycliquement ;

benchmark 4: : appel à la procédure `fill_bench_i_block()` i.e remplissage avec les valeurs $0, 1, \cdots, k-1, k, k+1, \cdots, 2(k-1), 2(k-1)+1, \cdots = 0, 1, 2, 3, ..., n-1$

benchmark 5: : remplissage de l'élément à la position `i` après un seul appel à la procédure `random()` ;

benchmark 6: : remplissage de l'élément à la position `i` après 5 appels à la procédure `random()` ;

1. Voir : `http://www.umiacs.umd.edu/research/EXPAR`

benchmark 7: : remplissage avec des valeurs comprises entre 1 et `dist`. Ce test permet de générer des doublons.

3. Résultats d'expérimentation

Examinons maintenant quelques résultats que nous pouvons obtenir à partir de nos codes. Ces exemples permettent d'apprécier les types de métriques mesurées pour évaluer les performances de différents clusters dans différentes situations. Dans tous les tableaux qui suivent il faut remarquer les temps d'exécution (nous donnons également les variances et écarts types) ainsi que la méthode de communication qui est soit par des accès directs à la mémoire distante (put/get) soit avec du send/receive. Cela nous permet de comparer des clusters en jouant aussi sur le type de primitives de communication utilisées.

Sur l'ensemble des tables qui vont suivre jusqu'à la fin du chapitre, nous listons les temps d'exécution dans la colonne MEAN ainsi que les variances et écart types des temps d'exécutions (colonnes VARIANCE et ST. DEVIATION). Sur ces tables, les colonnes $S(max), S(min)$ donnent la métrique d'expansion lorsqu'on considère respectivement le rapport entre la plus grande (respectivement la plus petite) valeur observée sur la valeur optimale. La valeur optimale est donnée dans la colonne MEAN/PROC. La colonne SIZE donne le nombre d'entiers en entrée, la colonne #PROC donne le nombre de processeurs utilisés.

TAB. 1: Configuration: 8 Celerons 466Mhz, 96Mb of RAM, Ethernet

#PROCs	INPUT SIZE	MEAN (s)	VARIANCE	ST. DEVIATION
8	131072	1.0242445	0.00012044	0.06011237
8	262144	1.1007769	0.00012975	0.06239024
8	524288	2.0009198	0.00021114	0.07958831
8	1048576	3.8748183	0.01260608	0.61496560
8	2097152	8.0455527	0.00156754	0.21685538
8	4194304	19.88127310	0.00374890	0.33536103
8	8388608	42.42296468	0.04470105	1.44946527

Execution time metrics for benchmark 0
– Implementation with DRMA routines –

Nous donnons la métrique d'expansion ce qui permet d'évaluer la qualité de l'équilibrage des charges et de vérifier la qualité des implémentations vis à vis de la métrique.

L'idée principale est d'offrir à une personne qui veut évaluer un cluster une large palette de situations à partir desquelles il pourra à son tour décider d'un choix de machine et d'un choix de primitives de communication pour son application.

Rappelons que tous les tests sont effectués en environnement dédié.

3.1. Implémentation standard de PSRS avec des put/get. Le test correspondant aux résultats d'exécution de la Table 1 est notre implantation BSP de PSRS (standard) avec uniquement des primitives de communication de type put/get distants ("one sided communication" ou DRMA). On note que la variance sur les temps d'exécution est bonne (pour chaque taille de problème nous lançons 30 expériences). Toujours sur la Table 1, pour les tailles supérieures à 1Mo, quand on double la taille le temps d'exécution est multiplié par un peu plus de deux. Ce résultat relativement moyen peut s'expliquer par l'utilisation d'Ethernet 10Mbits/s. De plus, 1Mo sur 8 processeurs revient à ce que chaque processeur reçoive 128K de données ce qui correspond à la taille du cache du Celeron. Donc, pour des problèmes plus grands que 1M, le cache est saturé et les performances décroissent.

TAB. 2: Configuration: 8 Celerons 466Mhz, 96Mb of RAM, Ethernet

#PROCs	SIZE	Mean/Proc	MAX	MIN	S(max)	S (min)
8	131072	16384	17215	15799	1.0507	.9642
8	262144	32768	34699	31138	1.0589	.9502
8	524288	65536	67346	64083	1.0276	.9778
8	1048576	131072	133505	128796	1.0185	.9826
8	2097152	262144	265849	259543	1.0141	.9900
8	4194304	524288	530224	518793	1.0113	.9895
8	8388608	1048576	1055074	1043317	1.0061	.9949

Sublist expansion metrics for benchmark 0
– Implementation with DRMA routines –

Toujours avec le même code, on note sur la Table 2 que la métrique d'expansion basée soit sur le minimum ou le maximum de données traitées est bonne voire presque optimale.

3.2. Implémentation standard de PSRS avec primitives send/receive. Les Tables 3 et 4 concernent également l'implantation de PSRS mais avec des primitives de type send/received au lieu des lecture / écriture distantes. De manière assez surprenante nous ne notons pas de différence sur les temps d'exécution par rapport à ceux de la Table 1 mis à part le cas 8M où l'amélioration est de 8%. On peut alors conclure que l'implantation en PUB (notre version utilisée de BSP) des primitives de type lecture/écriture distantes sont, au niveau utilisateur, plutôt satisfaisantes.

Sur la Table 5 nous avons changé de réseau d'interconnexion pour un Fast Ethernet. Par rapport à la Table 3 nous observons un gain sur les temps d'exécution de l'ordre

de 6 approximativement alors que le réseau est théoriquement 10 fois plus rapide. Sachant que notre application n'est pas gourmande en nombre de communications, nous pouvons estimer que le facteur 6 s'obtient à un coût de huit cartes Fast Ethernet, ce qui est modique.

Tous les résultats précédents sont obtenus pour le cas où les processeurs sont tous les mêmes. Nous tenons à la disposition du lecteur un descriptif exhaustif de tous les tests que nous avons pu mener sur différents clusters : Pentium Pro interconnectés par Myrinet, Alpha interconnectés par Fast Ethernet, Celeron interconnectés par Fast Ethernet et pour différents algorithmes de la famille des algorithmes d'échantillonnage avec une seule étape de fusion. Ce comparatif s'intéresse par ailleurs aux deux implémentations de BSP (BSPLib et PUB), le tri permettant de mettre à jour les performances relatives des implémentations. On en retire principalement que les implémentations des primitives put/get offerts dans PUB7 sont de meilleures qualités que celles offertes sous BSPLib.

Globalement, PUB7 nous semble supérieur à BSPLib. Les tests permettent d'apprécier le gain d'un cluster sous Myrinet par rapport à un Fast-Ethernet. Par exemple, au vu du prix des cartes Myrinet vis à vis des cartes Fast-Ethernet et au vu du gain relativement faible que l'on peut obtenir sur quelques processeurs interconnectés par les cartes Myrinet, on peut préférer une solution Fast-Ethernet pour les applications parallèles qui communiquent peu comme nos tris.

Bien sûr les tests sont effectués pour un matériel donné. Peut être que les performances avec Myrinet pourront être bien meilleures avec d'autres pilotes. Peut être aussi que la gestion des communications globales faite par PUB7 se prête mieux à un support Fast-Ethernet plutôt qu'à un support de communication de type Myrinet.

TAB. 3: Configuration: 8 Celerons 466Mhz, 96Mb of RAM, Ethernet

#P	SIZE	MEAN (s)	VARIANCE	ST. DEVIATION
8	131072	1.19026	0.01629	0.12763
8	262144	1.40763	0.01162	0.10782
8	524288	2.51928	0.01265	0.11250
8	1048576	4.60117	0.00763	0.08737
8	2097152	8.87850	0.01805	0.13435
8	4194304	18.83505	0.10263	0.32036
8	8388608	39.59852	1.13039	1.06319

Execution time metrics for benchmark 0
– Implementation with send/receive routines –

TAB. 4: Configuration: 8 Celerons 466Mhz, 96Mb of RAM, Ethernet

#PROCs	SIZE	Mean/Proc	MAX	MIN	S (max)	S(min)
8	131072	16384	17306	15615	1.05627	.9530
8	262144	32768	34047	31846	1.03903	.9718
8	524288	65536	66902	63871	1.02084	.9745
8	1048576	131072	133160	129366	1.01593	.9869
8	2097152	262144	265315	258972	1.01209	.9878
8	4194304	524288	529395	519623	1.00974	.9911
8	8388608	1048576	1055441	1040798	1.00654	.9925

Sublist expansion metrics for benchmark 0
–Implementation with send/receive routines–

TAB. 5: Configuration: 8 Celerons 466Mhz, 96Mb of RAM, Fast Ethernet

#PROCs	INPUT SIZE	MEAN (s)	VARIANCE	ST. DEVIATION
8	131072	0.28221903	0.00010097	0.01004839
8	262144	0.31610003	0.00008969	0.00947080
8	524288	0.45473823	0.00022831	0.01511008
8	1048576	0.77683036	0.00163511	0.04043654
8	2097152	1.55850710	0.01689717	0.12998914
8	4194304	3.25585880	0.06654323	0.25795975
8	8388608	6.58935290	0.68831433	0.82964711

Execution time metrics for benchmark 0
– Implementation with send/receive routines –

CHAPITRE 4

Tri en mémoire et en milieu hétérogène

DANS L'INTRODUCTION de cette partie sur le tri nous avons déjà dit combien il était important de controler la charge de travail qu'effectue chaque processeur pour celui qui doit composer avec plusieurs générations d'un même processeur au sein de son cluster. Il faut éviter que les processeurs les moins rapides recoivent au fil de l'exécution plus de données à traiter que ceux qui sont les plus rapides.

Le problème que nous traitons maintenant concerne le tri en mémoire lorsque les processeurs ne vont pas tous à la même vitesse. Le problème est le suivant : n données sont physiquement distribuées sur p processeurs et doivent être triées. Ici les processeurs sont caractérisés par leurs vitesses que nous dénotons par $s_i, (1 \leq i \leq p)$. Les taux de transferts avec les disques ainsi que la bande passante du réseau ne sont pas captés dans le modèle qui suit. De plus nous nous intéressons au << cas parfait >> c'est à dire au cas où la taille du problème peut se décomposer en p sommes d'entiers définies selon la vitesse de chacun des processeurs. La notion de *plus petit commun multiple* est utile ici afin de préciser les choses de manière mathématique. En d'autres termes, nous demandons à ce que les tailles n des problèmes s'expriment sous la forme :

$$n = k * perf[0] * lcm(perf, p) + ... + k * perf[p-1] * lcm(perf, p) \quad (4)$$

où

- k est une constante de $\mathbb{N}$;
- $perf$ est un tableau d'entiers de taille p qui contient les performances relatives des p processeurs du cluster ;
- $lcm(perf, p)$ est le plus petit multiple commun (least common multiple) des p entiers du tableau $perf$.

Par exemple, avec $k = 1, perf = \{8, 5, 3, 1\}$ (nous avons un processeur qui tourne 8 fois plus vite que le plus lent, un processeur tournant 5 fois plus vite que le plus lent et un processeur tournant 3 fois plus vite que le plus lent) nous obtenons que $lcm(\{8, 5, 3, 1\}, 4) = 120$ et donc $n = 120+3*120+5*120+8*120 = 2040$. Ainsi, avec des tailles de problèmes de la forme de l'équation 4, il est facile d'assigner à chaque processeur une quantité de données *inversement proportionnelle à sa vitesse*. C'est l'idée intuitive de départ et elle caractérise la précondition du problème. La propriété peut aussi s'exprimer par le fait que l'on demande à ce que la taille du problème soit divisible par la somme des valeurs du vecteur `perf`.

Si n ne peut pas s'exprimer comme à l'équation 4, des techniques autres comme celles présentées en [ARM95] peuvent être utilisées afin d'assurer l'équilibrage. Notre code

est écrit de sorte que seule la description du vecteur perf est à changer si l'on passe à un autre cluster !

1. Le ressort du partitionnement

Le point clé pour obtenir de bonnes performances d'équilibrage de charge est à nouveau le choix des pivots. En effet ce sont eux qui permettent de partitionner l'entrée en portion de taille approximativement identiques dans le cas homogène. Ici, pour le cas hétérogène il faut s'arranger pour que le partionnement constitue des portions de l'entrée de tailles proportionnelles à la vitesse de chaque processeur.

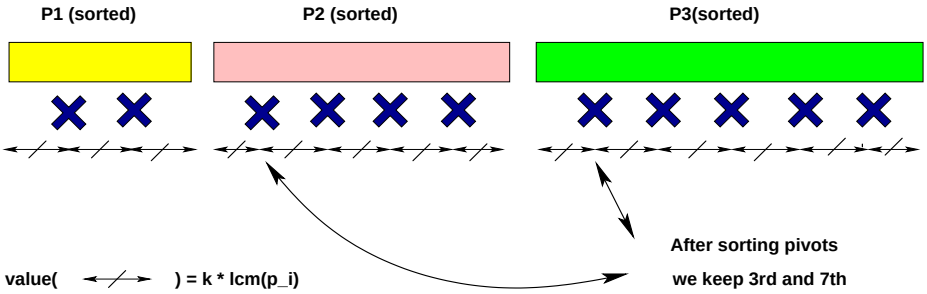

FIG. 1: *Sélection des pivots en hétérogène*

Considérons la Figure 1 pour expliquer intuitivement quels sont les ressorts profonds du partitionnement pour le cas hétérogène. La Figure représente l'état du programme au moment de la sélection (en parallèle sur chacun des processeurs) des pivots. Sur la Figure 1 nous avons trois processeurs qui ont trois vitesses différentes représentées par trois rectangles de dimension différentes. Les pivots sont pris à des intervalles réguliers ici au sens de PSRS et qui sont matérialisés par les symboles $\longleftrightarrow$. Sur la Figure 1, la notation $\mathtt{value}(\longleftrightarrow) = k \times \mathrm{lcm}(p_i)$ signifie que le nombre d'éléments entre deux pivots est pris de manière proportionnelle au plus petit commun multiple des vitesses des processeurs. Les pivots sur la Figure 1 proviennent des trois processeurs et ils sont triés à l'étape suivante de l'algorithme. Supposons que les pivots soient triés et examinons comment se fait le choix final.

Nous gardons ici, en particulier le troisième pivot (en partant de la gauche). La justification intuitive est la suivante : on opérant de la sorte on remarque que le nombre de données à gauche de ce premier pivot est nécessairement plus petit que deux fois la taille du segment de données contenues initialement sur le processeur de gauche (le moins rapide). En maintenant « cet invariant » quant au choix des autres pivots, on arrive à un résultat du type de celui de PSRS : aucun des processeurs ne traite plus de deux fois ce qu'il avait au départ. L'algorithme PSRS (Parallel Sorting by Regular Sampling) pour le cas homogène repose sur les mêmes observations.

De manière formelle, on peut exprimer la preuve de notre théorème qui affirme qu'à la dernière étape de l'algorithme aucun des processeurs hétérogène n'a en sa possession plus de deux fois ce qu'il avait au départ de la manière suivante. On note w_i la quantité initiale de donnée sur le processeur i et s la somme des performances. Avec les définitions précédente on a $w_i = k * lcm(p_i)$.

Considérons que les processeurs sont numérotés de 1 à p et qu'ils sont ordonnés selon leurs vitesses : le processeur avec la plus petite vitesse est numéroté 1, celui avec la plus grande vitesse est numéroté p. C'est le cas pour l'exemple de la Figure 1. Le nombre de pivots pris sur le processeur p_i est égal à $(perf[i] * p) - 1$. On a aussi que le nombre de données initialement sur le processeur i est $perf[i] * w_1, \forall i : (1 \leq i \leq p)$.

La preuve se fait par induction sur le nombre de processeurs. Considérons le processeur de numéro 1. Le premier pivot gardé par constrution et parmi les pivots candidats est le pivot numéro $perf[1] * p$. Toutes les données à fusionner par le processeur 1 doivent être inférieures ou égales à y_1, le premier pivot. Puisqu'il y a $s-(perf[1]*p-1)$ pivots qui sont supérieurs à y_i, il y a au moins $(s-(perf[1]*p-1)) * \frac{w_1}{perf[1]*p}$ eléments dans l'entrée qui sont supérieurs à y_1. En d'autres termes, il y au plus :

$$\begin{aligned} n - (s - (perf[1] * p - 1)) * \frac{w_1}{perf[1]*p} &= n - s * \frac{w_1}{perf[1]*p} + \frac{w_1 * perf[1]*p}{perf[1]*p} + \frac{w_1}{perf[1]*p} \\ &= n - n + \frac{w_1}{perf[1]*p} * perf[1] * p + \frac{w_1}{perf[1]*p} \\ &= w_1 + \frac{w_1}{perf[1]*p} < 2 * w_1 \end{aligned}$$

eléments de l'entrée qui sont inférieurs ou égaux à y_1 comme attendu.

Par une induction sur le nombre de processeurs et en spécialisant attentivement les pivots sélectionnés, on arrive au résultat. □

2. Introduction aux algorithmes sélectionnés

Nous proposons maintenant une présentation des algorithmes de tri que nous avons développés dans le cas hétérogène et en mémoire principale. Il s'agit d'un algorithme << à la PSRS<< et d'un algorithme << façon sur échantillonnage >> . C'est la méthode de sélection des pivots qui les différencie.

Les deux sous-sections qui suivent sont organisées en deux parties :

(1) présentation de l'algorithmique et justifications du choix des pivots ;

(2) plan d'expérimentation et performances obtenues.

3. Un algorithme façon « sur-échantillonnage »

On ne trie pas initialement les données. En procédant de la sorte on évite une étape coûteuse en temps. A priori, les résultats sur les temps d'exécutions seront meilleurs que pour PSRS (Parallel Sorting by Regular Sampling).

Ici, l'idée intuitive est de prendre « beaucoup » de pivots car plus on a de pivots, meilleur sera l'équilibrage car il est déterminé à partir de l'information codée dans la valeur des pivots. Comme nous ne pouvons tirer aucune information utile de données non triées (on ne connait pas l'ordre relatif des données contrairement à PSRS), le choix d'un nombre important de pivots est une nécessité afin d'arriver à constituer des partitions de tailles proches de l'optimum selon une définition adaptée au cas hétérogène.

De nouveau, comme pour le problème traité au paragraphe précédent, l'état initial de l'algorithme est le suivant : chaque processeur à une quantité de données en mémoire proportionnelle à sa vitesse. On veut qu'à l'état final de l'algorithme chaque processeur contienne à peu près la même quantité de données qu'initialement mais les données sont triées.

3.1. Présentation de l'algorithme. Rappelons d'abord le principal résultat de Li et Sevcik pour le cas homogène (voir [LS94] pour la preuve):

THÉORÈME : 5 (Li & Sevcik). *Pour une liste non triée de taille* n, $(p.k-1)$ *pivots (avec* $k \geq 2$*) partitionnent la liste en* $p.k$ *sous listes telle que la taille de la plus grande sous liste est plus petite ou égale à* n/p *avec la probabilité au moins égale à*

$$1 - 2p\left(1 - \frac{1}{2p}\right)^{p.k}$$

Faisons l'hypothèse que nous ayons $p > 1$ vitesses de processeurs différentes s_i. Dans le cas non homogène, il se peut qu'on arrive à la situation où le processeur p_i doit trier (dans la dernière étape de calcul) c_i données pour lesquelles on a la relation :

$$c_1 \times \frac{1}{s_1} = c_2 \times \frac{1}{s_2} = \cdots = c_p \times \frac{1}{s_p} = \text{constante}$$

Pour des raisons de simplicité, considérons maintenant que n, la taille du problème soit un multiple de chacun des s_i. Par exemple considérons le cas suivant : $s_0 = s_1 = 1, s_2 = s_3 = 2$ (deux processeurs vont deux fois plus vite que les deux autres dans un cluster de quatre processeurs). Prenons $n = 48$. D'après la relation précédente, deux processeurs seront chargés initialement avec 8 données et les deux autres avec 16 données chacun.

3.2. Choix des pivots. Le problème principal concerne l'équilibrage, puisque les processeurs n'ont pas la même quantité de données initialement. Deux alternatives sont possibles :

- soit on prend le même nombre de pivots par processeurs ;
- soit on prend un nombre de pivots par processeur de manière proportionnelle à chaque vitesse ;

Nous optons pour la seconde alternative. Pour observer pourquoi le choix est correct, nous pouvons écrire, à partir de nos hypothèses que $n = C \times n_1$ (n_1 est supposé être la quantité de données initialement à la disposition de P_1). L'idée est << d'égaliser le nombre de pivots >> en fonction de la vitesse du processeur le moins rapide et ceci en considérant le plus petit commun multiple des vitesses. Par exemple, si on prend $p = 4, n = 60$ et deux processeurs tournent deux fois plus vite que les deux autres alors il est équivalent de dire que $n = 6 \times 10$ c'est à dire que nous travaillons virtuellement avec un système à 6 processeurs au lieu de 4. Donc dans ce cas $C = 6$. Il n'est alors pas très difficile d'observer que le Théorème 5 est correct en prenant $p = 6$.

Sur la Figure 2 nous pouvons observer l'influence du paramètre k sur la probabilité d'obtenir des partitions proches de l'optimum. Sur l'axe des x nous avons le nombre de processeurs et sur l'axe des y la probabilité.

Noter que le cas homogène s'obtient quand $C = 1, n_1 = n$. Nous avons dessiné la courbe pour $k = 2\log p, 3\log p, 4\log p$. En redémarant notre exemple précédent (quatre processeurs physiques mais six virtuels) et pour le cas $k = 3\log p$, nous obtenons une probabilité de 80% que la taille maximale d'une sous liste soit plus petite ou égale à $60/6$. La valeur $k = 3\log p$ peut être considérée comme fournissant une probalitité suffisamment élevée.

Nous modifions l'algorithme pour l'ordonnanceur (étape 4 de l'algorithme 1) afin d'effectuer le travail de redistribution de manière proportionnelle à la vitesse. C'est la seule modification à apporter au schéma de Li et Sevcik [LS94] appellé PSOP (Parallel Sorting by Over-Partionning). Nous obtenons de de la sorte un algorithme << façon PSOP >> mais pour le cas hétérogène.

3.3. Résulats expérimentaux. Sur l'ensemble des tables qui vont suivre jusqu'à la fin du chapitre, nous listons les temps d'exécution dans la colonne MEAN ainsi que les variances et écart types des temps d'exécutions (colonnes VARIANCE et ST. DEVIATION). Sur ces tables, les colonnes $\mathtt{Sublist}(\mathtt{fast}), \mathtt{Sublist}(\mathtt{slow})$ donnent respectivement la métrique d'expansion pour les processeurs rapides et pour les processeurs. La valeur optimale de la métrique d'expansion est éventuellement donnée dans la colonne intitulée MEAN/PROC. La colonne SIZE donne le nombre d'entiers en entrée, la colonne #PROC donne le nombre de processeurs utilisés. Nous listons

également dans les colonnes Opt. Slow et Opt. Fast les tailles des partitions optimales attendues pour le cas des processeurs lents et pour le cas des processeurs rapides, respectivement. Le ratio des vitesses entre les processeurs lents et rapides (speed ratio) est pris égal à 2 par exemple.

Des résultats d'expérimentation pour la métrique d'expansion et notre algorithme façon PSOP (Paralle Sorting by Over-Partionning) sont présentés dans la Table 1. Le nombre de pivots pris sur un processeur dans l'étape 1 de l'algorithme 1 est 7 par processeur lent et 16 par processeur rapide soit un total de 46 pivots. Nous observons ici des meilleures métriques d'expansion par rapport à ceux de la Table 5 (voir l'algorithme qui suit dans la prochaine section). C'est attendu car nous avons plus de pivots.

TAB. 1: Configuration: 4 Celerons 466Mhz, 96Mb of RAM, Ethernet

#P	SIZE	Opt. Slow	Sublist (slow)	Opt. Fast	Sublist (fast)
4	131070	21845	0.9611	43690	1.1192
4	262140	43690	0.8740	87380	1.1239
4	524280	87380	1.2205	174760	0.8917
4	1048572	174762	1.2489	349524	0.7898

4 décembre 2002

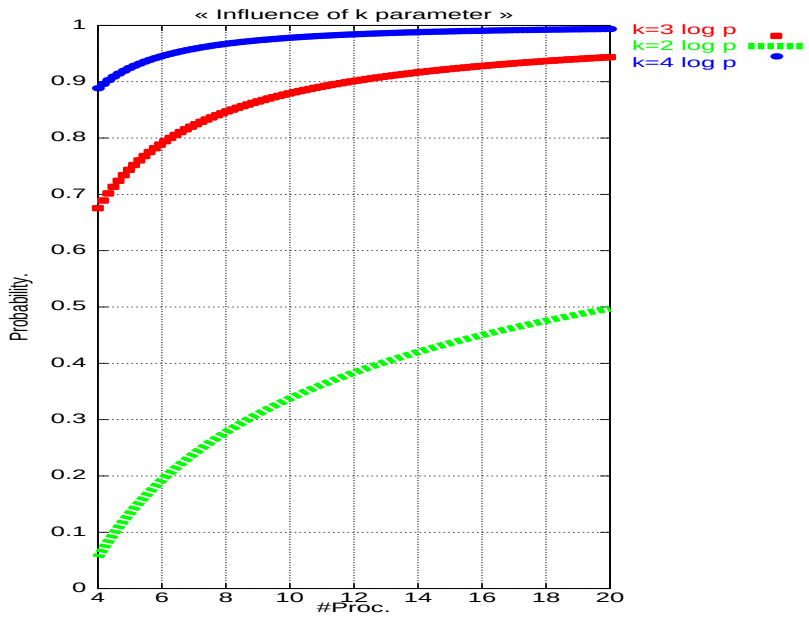

FIG. 2: *Bornes inférieures de la taille des sous listes*

TAB. 1: Configuration: 4 Celerons 466Mhz, 96Mb of RAM, Ethernet

4	2097150	349525	1.2350	699050	0.8043
4	4194300	699050	1.2502	1398100	1.1987
4	8388606	1398101	1.2446	2796202	1.1515

Sublist expansion metrics for benchmark 0
Algorithm used: PSOP augmented
– Speed ratio: 2, number of pivots: 46 –

La Table 2 présente les temps d'exécution de notre algorithme PSOP conçu pour le cas non homogène avec un tableau de performance initialisé avec des 1 c'est à dire que nous avons configuré l'algorithme pour le cas homogène. Cette table doit être comparée avec la Table 5. Nous remarquons une dégradation du temps d'exécution de 50% ce qui est beaucoup ! En fait les valeurs du couple de paramètres s, k ne sont pas suffisamment grandes pour assurer un bon équilibrage (qui n'est pas montré dans les tables).

Sur les Tables 3 et 4 nous avons encore des résultats expérimentaux avec notre algorithme PSOP non homogène configuré pour le cas homogène avec les initialisations suivantes $s = 2, k = 12$. Ici il n'y a pas d'améliorations sensibles pour les temps d'exécution comparés à la Table 2 mais il y a une amélioration concernant la métrique d'expansion qui est maintenant comprise entre 1.45 et 1.81. L'explication est conforme à ce que l'on attendait : avec plus de pivots on obtient un meilleur équilibrage mais aussi un surcôut dans la gestion des pivots.

Pour cette configuration de cluster (8 Celerons interconnectés par fast Ethernet) nous n'avons pas trouvé de couple s, k permettant d'arriver à une métrique d'expansion proche de 1.3 comme dans [LS94]. Notons que nous n'avons pas le même générateur de nombres aléatoires que celui utilisé en [LS94] mais il reste commun à toutes nos expérimentations.

Une façon d'améliorer les performances en temps serait de faire chevaucher les calculs et les communications, par exemple en commençant à trier (dernière phase) alors que des communications continuent à arriver. Le support des threads dans BSP PUB7 existe. Nous n'avons pas encore expérimenté avec ce support.

TAB. 2: Configuration: 8 Celerons 466Mhz, 96Mb of RAM, Fast Ethernet

#PROCs	INPUT SIZE	MEAN (s)	VARIANCE	ST. DEVIATION
8	131072	0.61519600	0.01807593	0.13444677
8	262144	0.69753413	0.01522933	0.12340720
8	524288	0.86364826	0.00764911	0.08745920

TAB. 2: Configuration: 8 Celerons 466Mhz, 96Mb of RAM, Fast Ethernet

8	1048576	1.42352820	0.02070231	0.14388299
8	2097152	2.48709573	0.00827337	0.09095808
8	4194304	4.79235303	0.00512561	0.07159339
8	8388608	9.83081696	0.00469754	0.06853863

Execution time metrics for benchmark 0
– PSOP Implementation with send/receive routines –
– Homogeneous case with $s = 2, k = 3(\log 8)$ –

TAB. 3: Configuration: 8 Celerons 466Mhz, 96Mb of RAM, Fast Ethernet

#PROCs	INPUT SIZE	MEAN (s)	VARIANCE	ST. DEVIATION
8	131072	1.26604006	0.00734029	0.08567549
8	262144	1.42320176	0.00640213	0.08001333
8	524288	1.54578996	0.00607784	0.07796054
8	1048576	1.90209646	0.00416259	0.06451817
8	2097152	2.86067366	0.00424281	0.06513686
8	4194304	5.00813026	0.00225436	0.04748017
8	8388608	9.71697716	0.00166868	0.04084953

Execution time metrics for benchmark 0
– PSOP Implementation with send/receive routines –
– Homogeneous case with $s = 2, k = 12(4 * \log 8)$ –

TAB. 4: Configuration: 8 Celerons 466Mhz, 96Mb of RAM, Fast Ethernet

#P	SIZE	Opt. size/proc	Sublist (max)	Sublist (min)
8	131072	16384	1.45886230	0.62561035
8	262144	32768	1.57421875	0.55142211
8	524288	65536	1.60064697	0.45127868
8	1048576	131072	1.55633544	0.54898834
8	2097152	262144	1.68928909	0.57336044
8	4194304	524288	1.61158561	0.51007080
8	8388608	1048576	1.81142711	0.48182010

Sublist expansion metrics for benchmark 0
Algorithm used: our PSOP framework
– Homogeneous case with $s = 2, k = 12(4 * \log 8)$ –

4. Un algorithme façon Parallel Sorting by Regular Sampling

Dans ce paragraphe nous décrivons un algorithme de tri en mémoire en milieu hétérogène qui est basé sur le schéma PSRS (Parallel Sorting by Overpartioning). On commence donc par trier, on isole ensuite des pivots qui servent à partitionner l'entrée, on redistribue et on termine par fusionner les séquences triées.

Le point clé en hétérogène est la manière de choisir les pivots ainsi que leur nombre. Nous allons présenter un algorithme lorsque le cluster est constitué de processeurs avec au plus deux vitesses différentes.

Nous allons montrer une nouvelle fois comment maîtriser l'équilibrage puis nous donnerons des résultats expérimentaux.

4.1. Présentation de l'algorithme. Nous réutilisons ici les notations de [SS92]. Nous adaptons principalement la phase de sélection du pivot de l'algorithme de tri par échantillonnage régulier [SS92], les autres phases restent identiques. Nous prenons p processeurs avec $p = p_1 + p_2$ et p_1 est le nombre de processeurs avec la plus grande vitesse s_1 et p_2 est le nombre de processeurs avec la plus petite vitesse s_2. Soit $k = \max\{\frac{s_1}{s_2}, \frac{s_2}{s_1}\}$ et faisons l'hypothèse que k est entier. Initialement, chaque p_1 processeurs a $n_1 = \frac{k.n}{(k+1)p_1}$ données et chaque p_2 a $n_2 = \frac{n}{(k+1)p_2}$ données qui, comme dans le reste du document, sont des entiers. Chaque processeur commence par trier séquentiellement sa portion par un algorithme approprié. Puisque chaque processeur trie un nombre de données proportionnel à sa vitesse, on peut s'assurer que tous les processeurs finissent en même temps.

Sur chacun des p processeurs, $p-1$ échantillons sont choisis, également espacés : les p_1 processeurs gardent les valeurs à des indices multiples de $\lfloor \frac{n_1}{p*k} \rfloor$, les p_2 processeurs gardent dans leurs sous-listes triées, les valeurs aux indices multiples de $\lfloor \frac{n_2}{p*k} \rfloor$. Les $p(p-1)$ pivots sont rassemblés sur un processeur dédiés qui les trie. Maintenant, supposons que $p-1$ valeurs sont choisies (d'une certaine manière) dans le vecteur des pivots triés. Nous pouvons alors appliquer le résultat de [SS92] pour affirmer que dans la phase 4 de l'algorithme, chaque processeur trie pas plus $2.n_1$ valeurs. Il s'agit d'une application directe du théorème de [SS92]. Notre résultat précise un peu les choses :

THÉORÈME : 6. *Dans la phase 4 de l'algorithme, sous les hypothèses que* $k \leq 2\frac{p_1}{p_2} \wedge p_2 = p_1 \wedge n > p^3$, *chacun des* p_1 *processeurs fusionne au plus* $2.n_1$ *valeurs et chacun des* p_2 *processeurs fusionne au plus* $2n_2$ *valeurs.*

Preuve informelle : [CG00d] le lecteur est invité à reprendre la figure 1 qui aborde des idées intuitives. Elle repose sur le choix des pivots sélectionnés parmi les pivots triés. En effet, dans le vecteur des pivots triés, si l'on considère deux valeurs

consécutives qui proviennent d'un processeur rapide par exemple, le nombre de données entre ces deux pivots est nécessairement plus grand que le nombre de valeurs situées entre deux pivots consécutifs mais pris depuis des processeurs lents. □

Remarquons que la notion de *plus petit commun multiple* peut être aussi utilisée ici dans un algorithme "PSRS like" mais elle conduit généralement à gérer au niveau de la phase de sélection des pivots plus que les $p(p-1)$ pivots de l'algorithme précédent, ce qui peut induire un surcout de traitement. Nous allons voir maintenant les performances de cette stratégie mais aussi de la stratégie par sur-échantillonnage.

4.2. Résultats expérimentaux. Les algorithmes de tri parallèle en milieu hétérogène exposées dans le paragraphe précédent (tris façon sur-partionnement et façon PSRS) ont été validées et testées sur différentes plates-formes (Pentium Pro interconnecté avec Myrinet, processeurs Alpha avec fast Ethernet, processeurs Celeron avec Ethernet). Les résultats introduits maintenant ont été obtenus sur un réseaux de Celeron et sur un réseau d'Alpha (alpha 21164 - EV56, Red Hat Linux 5.2, Kernel 2.2.11, 4Mo de cache secondaire) situés à Amiens[1], France, et dont les réseaux d'interconnexion sont fast Ethernet. Nous intéressons ici à la *métrique d'expansion* afin de vérifier la qualité de l'équilibre de charge. Nous nous intéressons au cas de clusters à quatre processeurs dont deux vont deux fois plus vite que les deux autres.

Rappelons que les pré-conditions des expérimentations sont que chaque processeur reçoit au départ une quantité d'entiers à trier inversement proportionnelle à la vitesse du processeur.

Sur la table 5, les colonnes $\mathsf{Sublist}(\mathsf{fast}), \mathsf{Sublist}(\mathsf{slow})$ donnent respectivement la métrique d'expansion pour les processeurs rapides et pour les processeurs. La colonne SIZE donne le nombre d'entiers en entrée, la colonne #PROC donne le nombre de processeurs utilisés. Nous listons également dans les colonnes Opt. Slow et Opt. Fast les tailles des partitions optimales attendues pour le cas des processeurs lents et pour le cas des processeurs rapides, respectivement. Le ratio des vitesses entre les processeurs lents et rapides (speed ratio) est pris égal à 2, le nombe de pivots finalement sélectionné est 3.

La Table 5 présente quelques résultats pour la technique PSRS modifiée telle qu'elle a été présentée au paragraphe précédent, pour le test 0 (choix aléatoire des données et 30 expérimentations pour chaque taille de problème). Nous vérifions qu'effectivement les ratios donnés dans les colonnes "Sublist" sont plus petits que 2. Cependant nous notons ici que les valeurs ne sont pas toujours voisines de l'optimal qui est 1 (équilibrage parfait). On remarque que les résultats expérimentaux sont situés entre l'optimal 1 et la borne supérieure 2. Ainsi, le nombre de pivots choisis est suffisant pour assurer la borne mais pas suffisant pour assurer un excellent résultat pratique.

1. http://www.laria.u-picardie.fr/PALADIN

TAB. 5: Configuration: 4 Celerons 466Mhz, 96Mb of RAM, Ethernet

#PROCs	SIZE	Opt. Slow	Sublist (slow)	Opt. Fast	Sublist (fast)
4	131070	21845	0.7561	43690	1.1219
4	262140	43690	0.7540	87380	1.1229
4	524280	87380	1.4005	174760	0.7997
4	1048572	174762	1.6248	349524	0.6875
4	2097150	349525	1.5750	699050	0.7124
4	4194300	699050	1.5002	1398100	1.4995
4	8388606	1398101	1.5000	2796202	1.5001

Sublist expansion metrics for benchmark 0
Algorithm used: PSRS with a merge
–Speed ratio: 2, number of pivots: 3–

5. Conclusion sur le tri hétérogène en mémoire

Nous avons présentés deux algorithmes de tri en mémoire pour clusters hétérogènes. Ces deux algorithmes sont des généralisations des techniques PSRS (Parallel Sorting by Regular Sampling) et PSOP (Parallel Sorting by Overpartitioning). Ils permettent maintenant de capter aussi bien le cas homogène que le cas ou les processeurs ne vont pas tous à la même vitesse.

Expérimentalement, notre algorithme « façon PSRS » fournit des résultats en terme d'équilibrage, supérieurs à notre algorithme « façon PSOP » . C'est attendu car on trie l'entrée ce qui permet de tirer une information sur l'ordre global des éléments. Cependant, une seule méthode de sur-échantillonnage a été comparée. Nous projetons d'étudier d'autres façons de sélectionner les pivots de manière aléatoire afin d'obtenir un panel d'algorithmes plus grand. Les méthodes potentielles de choix des pivots par échantillonnage ont été présentées.

D'un point de vue méthodologie expérimentale, les temps d'éxécution et les métriques d'expansions permettent de comparer entre eux les deux algorithmes. Nous avons ajouté également une comparaison, pour chaque algorithme, entre le cas où il est configuré en homogène (alors que le cluster est physiquement hétérogène) et le cas où il est configuré de manière hétérogène.

Cela permet de valider séparément les deux algorithmes sans avoir recours à un algorithme référence dont la détermination est d'ailleurs un problème en soit, en général. En effet, en algorithmique parallèle on a l'habitude de comparer les exécutions parallèles avec le « meilleur algorithme séquentiel connu pour le problème » ce qui défini le gain. Dans le cas d'un cluster hétérogène de processeurs, que veut dire le meilleur algorithme séquentiel? Notre méthode de comparaison résoud ce problème.

Nous allons voir maintenant comment les principes exposés dans ce chapitre s'appliquent également pour le tri sur disques.

Notre motivation pour nous intéresser au tri séquentiel vient aussi du fait que nous étudions par ailleurs les algorithmes parallèles de tri en milieu hétérogène. Pour cela nous avons besoin d'utiliser des algorithmes séquentiels qui trient des sous-problèmes du problème initial.

Enfin, si la hiérarchie mémoire est bien captée avec la notion d'algorithme oublieux du cache tant sur le point théorique que sur les implémentations qu'on peut faire avec les premiers niveaux de mémoire alors on pourra envisager d'examiner expérimentalement le comportement des algorithmes vis à vis des disques. Un algorithme oublieux du cache est conçu pour fonctionner aussi à ce niveau de hiérarchie mémoire. Pour l'exemple du tri externe, peut être que l'on pourrait montrer que pour obtenir des performances, il est suffisant de bien gérer les caches (par un algorithme oublieux du cache) plutôt que de chercher à développer des algorithmes dépendants de paramètres matériels (taille des blocs disque lus...).

CHAPITRE 5

Tri parallèle sur disque en milieu hétérogène

NOUS ALLONS EXAMINER dans ce chapitre comment les techniques d'échantillonage précédentes peuvent jouer un rôle majeur parmi les algorithmes parallèles de tri sur disque. Notre objectif consiste à réutiliser au maximum les approches retenues dans les chapitres précédents pour le choix des pivots. Par exemple, PSRS consiste dans sa première étape de calcul, en un tri séquentiel. Une stratégie consiste à le remplacer par le << meilleur >> algorithme séquentiel de tri disque et de capitaliser sur le travail du chapitre précédent sur le choix des pivots.

Comme autre statégie de développement, nous avons eu dès le départ l'idée de développer un code qui sépare clairement le travail séquentiel sur les disques et le travail de copie de disques à disques. L'approche PSRS offre cela. Olivier Cozette qui est au LaRIA développe (dans un projet plus général de système de fichiers) une interface pour faire de la copie distante de disques SCSI à disques SCSI. En dissociant clairement dans le code les accès aux disques locaux et ceux effectués sur les disques distants, nous allons pouvoir mesurer sans difficultés techniques importantes, au niveau d'une application largement répandue, quel est le gain procurée par l'interface.

Nous ne connaissons pas de travail sur le tri disque en milieu hétérogène. Ici les processeurs ne sont pas tous à la même vitesse, les données sont sur les disques initialement et des disques ne sont pas hétérogènes, ni le réseaux de communication. À chaque fois dans les études sur le tri disque il est sous entendu que la machine est homogène (CPU, disques, réseaux). Notre travail constitue donc une première en la matière. Il consiste précisément à effectuer un *vrai* tri de disques à disques : les données sont initialement sur les disques en un nombre qui est proportionnel à la vitesse des processeurs et les résultats sont écrits sur les disques.

1. Un aperçu de l'existant et points de terminologie

Les objectifs généraux poursuivis en séquentiel comme en parallèle sont soit la minimisation du nombre de déplacement des têtes de lecture soit le nombre de fois ou l'on accède au disque. En effet, les coûts d'accès aux disques (en particulier sur une architecture de type cluster) sont importants avec les technologies actuelles en partie mécaniques. Par exemple, le taux moyen de transfert depuis le disque du PC est de l'ordre de 15M octets/seconde avec une interface Ultra ATA/66 alors qu'il est possible d'échanger avec un réseau Myrinet [1] à des taux de 120M octets/seconde. On voit donc apparaître un facteur à peu près égal à 10 !

1. Voir http://www.myri.com

D'un point de vue modèle de disques, les papiers de Vitter [VS94a, VS94b, AV88] font référence ainsi que la synthèse [AV99]. Dans ces articles, on capture les propriétés des systèmes de disques dans un modèle que l'on appelle PDM pour *parallel disk model* qui fait référence aux paramètres suivants[2] :

$$\begin{aligned} \mathsf{N} &= \text{taille du problème ;} \\ \mathsf{M} &= \text{taille de la mémoire interne (RAM) ;} \\ \mathsf{B} &= \text{taille d'un bloc transféré ;} \\ \mathsf{D} &= \text{nombre de disques indépendants ;} \\ \mathsf{P} &= \text{nombre de processeurs de calcul.} \end{aligned}$$

avec $\mathsf{M} < \mathsf{N}$, et $1 \leq \mathsf{DB} \leq \mathsf{M}/2$ pour des raisons pratiques afin de correspondre au mieux aux << systèmes réels >> . La Figure 1 décrit le modèle PDM. Sur la partie gauche, un seul processeur accède à la mémoire partagée et les trois disques peuvent fonctionner en parallèle (<< à la RAID >>) mais les papiers pré-cités ne le précisent pas. Sur la Figure 1, partie de droite, les processeurs accèdent à leurs disques locaux et ne peuvent échanger leurs données contenues sur les disques que via le réseaux

2. Note : 150 membres de la "Storage Networking Industry Association" ont mis en ligne un dictionnaire sur la terminologie technique en matière de stockage - `http://www.snia.org` - Autres pointeurs HTML particulièrement intéressants : `http://www.cs.duke.edu/~large` (Lars Arge : I/O-Efficient Algorithms) et `http://www.buyya.com/superstorage` (liste de cours)

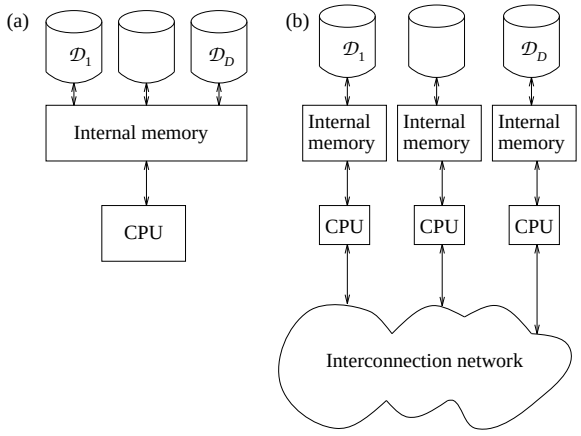

FIG. 1: *La Figure provient de [VS94a]. Modèles de disques : (a)* $\mathsf{P} = 1$*, pour lequel les* D *disques sont connectés par un processeur commun (b)* $\mathsf{P} = \mathsf{D}$*, pour lequel les* D *disques sont connectés à un processeur distinct. Cette << organisation >> correspond à la << notion >> de cluster.*

d'interconnexion. Cette partie de la figure décrit une architecture de type cluster. C'est encore une fois pour cette architecture que nous proposons des algorithmes.

La métrique de performance est définie en considérant le nombre de communications d'entrées sorties (I/O) [May00] entre la mémoire interne et le disque externe. On peut aussi s'intéresser à l'espace disque utilisé par l'algorithme ainsi qu'au temps CPU. Dans une opération d'I/O, chacun des D disques peut simultanément transférer un bloc de B données consécutives sur le disque. Il est utile de considérer les raccourcis suivants :

$$n = \frac{N}{B}, \qquad m = \frac{M}{B}$$

La valeur n représente donc le nombre d'entrées/sorties nécessaires pour lire dans son entier l'entrée du problème alors que m représente le nombre de données qui tiennent en mémoire principale. On peut faire l'hypothèse que les N données $x_1, \cdots, x_N$ sont initialement distribuées (on dit "striped") sur les D disques comme cela est montré sur la Table 1 ou $D = 4, B = 2$. D'autres distributions sont possibles mais le point important est qu'un fichier réparti de N données peut être lu ou écrit en $\mathcal{O}(N/DB) = \mathcal{O}(N/D)$ opérations d'I/O ce qui est optimal.

La technique connue sous le nom de *disk striping* [SGM86], [Kim86] autorise les opérations d'I/O uniquement sur des bandes ("stripes") entières, une bande à la fois. L'idée est de répartir le premier bloc d'un fichier sur le premier disque, le deuxième bloc sur le deuxième disque etc. Par exemple, sur la Table 1, les données x_{18}, x_{22} peuvent être accédées en une seule opération d'I/O parce qu'elles sont situées sur la même bande. Ainsi, un modèle à D disques << stripés >> peut être considéré comme un seul disque logique mais avec un taille de blocs de DB.

TAB. 1: Initial data layout a $N = 32, D = 4, B = 2$ disk model.

	D_0	D_1	D_2	D_3
stripe 0	x_1, x_2	x_3, x_4	x_5, x_6	x_7, x_8
stripe 1	x_9, x_{10}	x_{11}, x_{12}	x_{13}, x_{14}	x_{15}, x_{16}
stripe 2	x_{17}, x_{18}	x_{19}, x_{20}	x_{21}, x_{22}	x_{23}, x_{24}
stripe 3	x_{25}, x_{26}	x_{27}, x_{28}	x_{29}, x_{30}	x_{31}, x_{32}

De plus et de manière idéale, les algorithmes utilisant les disques devraient utiliser un espace linéaire de stockage i.e. $\mathcal{O}(N/B) = \mathcal{O}(n)$ blocs de disques. Il peut être montré que la borne au nombre d'opérations d'I/O pour trier $N = n$ données avec $D \geq 1$ disques est donnée par :

$$\mathrm{Sort}(N) = \Theta\left(\frac{N}{DB}\log_{M/B}\frac{N}{B}\right) = \Theta\left(\frac{n}{D}\log_m n\right)$$

Noter que dans la pratique le terme $\log_m N$ est une petite constante. Le théorème principal de Aggarwal et Nordine [AV88], [NV95] et concernant le tri dans le modèle PDM est alors le suivant :

THÉORÈME : 7. *The average and worst-case number of I/Os required for sorting* $N = nB$ *data items using* D *disks is:*

$$\mathrm{Sort}(N) = \Theta\left(\frac{n}{D}\log_m n\right) \tag{5}$$

Dans l'équation 5, c'est le terme $\frac{n}{D} = \frac{N}{B.D}$ qui a de l'importance pour la pratique. On a donc intérêt, pour diminuer les nombres d'opérations d'entrées sorties, a avoir B le plus grand possible.

Une présentation des techniques classiques pour accélérer les performances d'accès à des disques par un seul CPU est faite dans le chapitre qui expose nos problèmatiques de stockage dans une plate-forme de calcul globale. Parmi les technoloqies connues et employées, la technique du stockage par bande (*disk striping*) peut être utilisée pour convertir de manière automatique un algorithme conçu pour utiliser un seul disque de taille de blocs DB en un algorithme qui utilise D disques chacun de taille B. Il est connu que la technique du stockage par bande n'est pas pertinente dans le cas du tri disque.

Pour l'observer, comme cela est fait dans [AV99], il faut considérer le nombre optimal d'opérations d'entrées sorties lorsqu'on utilise un seul disque avec des tailles de blocs égales à B qui est :

$$\Theta(n\log_m n) = \Theta\left(n\frac{\log n}{\log m}\right) = \Theta\left(\frac{N}{B}\frac{\log(n/D)}{\log(M/B)}\right) \tag{6}$$

Avec la technique du stockage par bande, le nombre d'opérations d'entrées est le même que si nous utilisions des blocs de taille $D.B$ dans l'algorithme optimal précédent. En remplaçant B par DB dans l'équation 6 on obtient la borne :

$$\Theta\left(\frac{N}{DB}\frac{\log(n/DB)}{\log(M/DB)}\right) = \Theta\left(\frac{n}{D}\frac{\log(n/D)}{\log(M/D)}\right) \tag{7}$$

Par ailleurs, la borne du tri pour trier avec D disques est :

$$\Theta(\frac{n}{D}\log_m n) = \Theta\left(\frac{n}{D}\frac{\log n}{\log m}\right) \tag{8}$$

On remarque alors que la borne donnée par l'équation 7 est plus grande que la borne donnée par l'équation 8 par un facteur :

$$\frac{\log(n/D)}{\log n}\frac{\log m}{\log(m/D)} \cong \frac{\log m}{\log m/D}$$

Quand D est de l'ordre de m le terme $\log(m/D)$ du dénominateur est petit et le surcout est de l'ordre de $\log m$ ce qui peut être significatif dans la pratique.

Nous avons inventés des techniques spéciales autres que la technique du stockage par bandes pour trier sur les disques.

2. Méthodes par échantillonnage

Parmi les articles les plus récents sur le tri disque en parallèle qui comportent des études expérimentales nous pouvons citer les références [Raj98], [CH97], [Pea99]. Nous allons nous concentrer sur quelques algorithmes qui se servent de la technique d'échantillonnage.

Pour obtenir la borne de l'équation 5 les techniques connues sous le nom de *distribution* ou *par fusion* peuvent être examinées là aussi. Avec ces techniques, on accède aux D disques de manière indépendante pendant des opérations de lectures en parallèle, mais de manière "stripée" pendant les phases d'écriture.

Examinons par exemple le cas du *distribution sort* de Knuth [Knu98] qui est dans l'esprit des algorithmes par échantillonnage que nous avons examinés dans les paragraphes 3 et 4.

Le tri par distribution (au sens de Knuth) est un algorithme récursif pour lequel les entrées sont successivement partitionnées par $S-1$ pivots pour former S morceaux (buckets). Les morceaux ainsi formés sont à leur tour triés de manière récursive. Il y a $\log_S(n)$ niveaux de récursion dans l'algorithme et la taille des morceaux décroit par un facteur $\Theta(S)$ d'un niveau de récursion à un autre. Si chaque niveau de récursion utilise $\Theta\left(\frac{N}{DB}\right) = \Theta\left(\frac{n}{D}\right)$ opérations d'I/Os alors le tri par distribution assure une complexité en nombre d'I/Os de $\mathcal{O}\left(\frac{n}{D}\log_m n\right)$ ce qui est optimal.

Évidemment la clef du succès dépend des pivots qui doivent scinder les morceaux en tailles à peu près égales. Comme cela est noté une nouvelle fois dans [VS94a], "It seems difficult to find $S = \Theta(m)$ splitters (pivots) using $\Theta(n/D)$ I/Os (the formation

of the buckets must be done akin this bound to guarantee an optimal algorithm) and guarantee that the bucket sizes are within a constant factor of one another".

Les auteurs de l'article [SK97] proposent un paradigme pour mixer les notions de métriques à la BSP et le calcul parallèle sur disque. Ils introduisent une courte discussion sur le cas du tri disque ("column sort" [Lei84] et "merge sort " [Col88]) mais ne donnent pas de résultats expérimentaux et n'envisagent pas non plus le cas hétérogène. L'apport est de considérer le travail total effectué par l'algorithme et pas seulement le nombre d'opérations d'I/Os. En fait, le paradigme est construit principalement à partir des paramètres suivants : N la taille d'entrée du problème, M la taille mémoire (RAM) et B la taille d'un bloc.

L'algorithme parallèle qui dans l'esprit est le plus prêt des algorithmes par échantillonnage tels que PSRS ou PSOP est, de notre avis, pour le modèle à D disques, le papier de DeWitt et al. [DNS91] qui est présenté comme un algorithme aléatoire à deux phases. L'algorithme construit des petites séquences triées sur disque qu'il s'agit de fusionner dans la deuxième phase.

3. Nos apports algorithmiques

3.1. Une approche PSOP. Nous avons fusionné à la fois les idées de [DNS91], [LS94] avec notre version modifiée de l'algorithme de Li et Sevcik que nous avons proposé au chapitre précédent pour capter le cas des clusters hétérogènes. Nous proposons alors le schéma suivant :

ALGORITHME 2 (Overpartitioning scheme for ext. sorting on heterogenous clusters).

Preconditions and initial configuration: *we deal with a cluster of* $p = 2^q$ *heterogeneous CPUs. The heterogeneous notion is coded in the array 'perf' of size p of integers that denote the relative performances of the* p *processors in the machine. The input size is* n *and* n *verifies Equation 4. Initially, processor* i *has* l_i*, a portion of size* $(n/(\Sigma_{i=1}^{p} perf[i])) * perf[i]$ *of the unsorted list* l*; The size is relative to the performance of the processor and it is an integer since it verifies 4.*

Step 1 (selecting pivots): *a sample of candidates are randomly picked from the list. Each processor* i *picks* $perf[i] * s * \log_2(p)$ *(see the overpartitioning method for the definition of terms) candidates and passe them to a designated processor. These candidates are sorted and then* $p.k - 1$ *pivots are selected by taking (in a 'regular way')* $s^{th}, 2.s^{th}, \cdots, (pk-1)^{th}$ *candidates from the sample. The selected pivots* $d_1, d_2, \cdots, d_{pk-1}$ *are made available to all the processors; All the computations can be done in-core since the number of pivots is very small in practice (it is not an order of the internal memory size) except for the read of candidates that require a total of* $\Sigma_{i=1}^{p} perf[i] * s * \log_2(p)$

I/Os. In practice, this amount is small (few hundreds or less) and should be negliged.

Step 2 (partitioning): *since the pivots have been sorted (and fit in the main memory), each processor performs binary partitioning on its local portion. Processor* i *decomposes* l_i *according to the pivots. It produces* pk *sublists per processor denoted* l_{ij} *where* i, j *stands for two consecutive pivots (except for the initial an final case). A sublist* S_j *is the union of* l_{ij} *with* i *ranging over all processors. There is* pk *sublists. Since there are* $Q = (n/(\Sigma_{i=1}^{p} perf[i])) * perf[i]$ *data on processor* i*, there is no more than* $2 * Q/B$ *IOs per processor to accomplish the work in this step (we count read and write of data).*

Step 3 (redistribution of the sublists): *we form messages in order that the sizes can fit in the local and distant memory. The size is also a multiple of the block size* B*. If we have an hardware which is able to transfer data from disk to disk, it will be more efficient. The number of IOs is no more than* $2\|l_i\|/B$ *(we also count read (sender side) and write (receiver side) of data).*

Step 4 (final sort): *each processor has in its local disk the final portion that can be sorted with an external sorted for mono-processor system. For instance, we use the "classical" sorting of [Knu98] that matches the bound on external sorting with* $D = 1$ *disk.*

Nous avons décomposé le problème de telle façon qu'il ne requière que des briques de base du modèle PDM (Parallel Disk Model) avec $D = 1$ et les résultats sur le nombre de pivots nécessaires à un bon équilibrage pour des machines hétérogènes.

3.2. Une approche façon PSRS. Mais nous pouvons aussi suivre les quatre phases de l'algorithme PSRS de la façon suivante dans le cadre du tri externe. Puisque la première étape de PSRS est un tri séquentiel nous l'implémentons ici, par exemple par un "Polyphase Merge Sort" [Knu98]. Ce type de tri bien connu utilise $2m$ fichiers intermédiaires afin d'obtenir un "$2m - 1$ way merge" sans avoir besoin d'une redistribution séparée des données après chaque passe. En fait il a les avantages d'un tri non équilibré sans le désavantage de la redistribution. Nous pouvons alors donner la description suivante :

ALGORITHME 3 (A PSRS scheme for external sorting on heterogeneous clusters).

Preconditions and initial configuration:: *we deal with a cluster of* $p = 2^q$ *heterogeneous nodes. The heterogeneous notion is coded in the array 'perf' of size p of integers that denote the relative performances of the* p *nodes in the machine. The input size is* n *and* n *verifies Equation 4. Initially, disk (or processor)* i *has* l_i*, a portion of size* $(n/(\Sigma_{i=1}^{p} perf[i])) * perf[i]$ *of the unsorted list* l*.*

Step 1: (sequential sorting): *in using polyphase merge sort we get* $2.l_i(1+\lceil \log_m l_i \rceil)$ *IO operations on each processor (since we have one disk attached per processor).*

Step 2: (selecting pivots): *a sample of candidates are randomly picked from the list. Each processor* i *picks* $L = (p-1) * perf[i]$ *candidates and passe them to a designated processor. This step requires* L *IO operations that is very inferior, in practice, to the IO operations of spet 1. These candidates are sorted and then* $p-1$ *pivots are selected by taking (in a 'regular way')* $s^{th}, 2.s^{th}, \cdots, (p-1)^{th}$ *candidates from the sample. Equiped with equation 4, it can be checked that* $s = k * \left\lfloor \frac{n * perf[i] * lcm(perf,p)}{p} \right\rfloor$. *The selected pivots* $d_1, d_2, \cdots, d_{p-1}$ *are made available to all the processors; All these computations can be done in-core since the number of pivots is very small in practice (it is not an order of the internal memory size);*

Step 3: (partitioning): *since the pivots have been sorted (and fit in the main memory), each processor performs binary partitioning on its local portion. Processor* i *decomposes* l_i *according to the pivots. It produces* p *sublists (files) per processor. Since there are* $Q = (n/(\Sigma_{i=1}^{p} perf[i])) * perf[i]$ *data on processor* i*, there is no more than* $2 * Q/B$ *IOs per processor to accomplish the work in this step (we count read and write of data).*

Step 4: (redistribution of the sublists):: *we form messages in order that the message sizes can fit in the local and distant memory. The size is also a multiple of the block size* B*. If we have an hardware which is able to transfer data from disk to disk, it will be more efficient. The number of IOs is no more than* $2 * l_i/B$ *(we also count read (sender side) and write (receiver side) of data).*

Step 5: (final merge): *each processor has in its local disk the* p *final portions (files) that can be merged with an external merged algorithm for monoprocessor system. We re-use the* `mergeSort()` *procedure of the polymerge sort algorithm used in step 1. The number of IO operations is lower than* $2.l_i(1+\lceil \log_m l_i \rceil)$*. Note that the constant 2 here stands for the bound of data given by the 'PSRS Theorem' that guarantee that in the last step of the algorithm, no processor has to deal with more than two times the initial amount of data. This theorem is still true for the heterogeneous case (see [CG00f]) and we apply it.*

Il est clair qu'à nouveau cette description ne fait appel qu'au modèle PDM dans le cas $D = 1$ plus le résultat obtenu dans le cas << en mémoire >> afin d'assurer un bon équilibrage dans le cas hétérogène [LS94, CG00f]. Concernant le processeur i qui au départ a l_i données, la borne optimale de $\Theta\left(\frac{l_i}{B} \log_m \frac{l_i}{B}\right)$ opérations parallèles d'I/Os est atteinte avec cet algorithme.

Donnons maintenant quelques indications sur la preuve de la borne 2. Notre stratégie pour obtenir la même borne que celle de PSRS est de se placer sous les conditions du théorème PSRS. Le point technique le plus délicat concerne le nombre et la façon de choisir les pivots après le tri initial. Puisque l'entrée vérifie l'Equation 4 nous pouvons prendre les pivots de manière régulière (au sens de PSRS) et de façon proportionnelle au plus petit commun multiple des performances. D'un point de vue programmation, le code que chaque processeur exécute pour sélectionner les pivots est celui-ci (la variable `blocksize` est le nombre d'entiers localement à un processeur - noter que la valeur de `i` est la même sur tous les processeurs ; ceci est dû à l'Equation 4):

```
i = (int) (blocksize / (performance[mypid] * nprocs)) - 1;
off = i + 1; k = 0;
while (i <= (blocksize - off - 1)) {
    fseek(MYfpIN, (long) (i * sizeof(MPI_INT)), SEEK_SET);
    fread(&pivot[k], sizeof(MPI_INT), 1, MYfpIN);
    k++;
    i += off;
}
```

La conséquence est que l'on prend un nombre de pivots proportionnel aux valeurs du vecteur de performance mais on s'assure qu'entre deux pivots consécutifs choisis il y a le même nombre d'éléments (triés). Ceci est la propriété essentielle de PSRS. Notre schéma est donc une généralisation de PSRS.

4. Expérimentations

Concernant l'épineuse question de savoir comment on choisit de remplir le vecteur `perf` qui donne les performances relatives de chaque nœud du cluster, nous avons choisi de procéder comme suit :

- pour une taille de problème N et sur p processeurs, nous exécutons des tests de mesure du temps d'exécution au moyen de l'implémentation du tri externe séquentiel (sur N/p données) qui s'exécute dans la version parallèle et externe. Les rapports au plus petit temps d'exécution moyen obtenu donnent les valeurs du vecteur 'perf'.

Ainsi nous captons à la fois la puissance de calcul de chaque processeur mais aussi les performances des entrées / sorties. Nous ne prétendons pas que les valeurs mesurées rendent compte des performances des nœuds dans l'absolu mais uniquement pour le cas du tri. Cependant, on peut dire que le travail effectué par nos algorithmes consiste à dérouler des versions séquentielles du tri, mis à part les phases de sélection des pivots, du tri en mémoire des pivots et de l'échange des partitions.

Les résultats qui suivent concernent les moyennes de 30 expérimentations (nous ne donnons pas les écarts types mais ils sont bons). Le cluster est physiquement homogène et nous allons le charger par des processus, typiquement des tris externes, afin de simuler un cluster hétérogène. La charge de chaque processeur reste identique pendant toutes les expérimentations. La plate-forme était dédiée à nos expérimentations.

Dans la Table 2 nous avons une vue synthétique de l'architecture du petit cluster avec lequel nous allons expérimenter l'algorithme de tri externe « façon PSRS » . La partition `work` sert pour tous les fichiers créés et il s'agit d'un disque SCSI.

TAB. 2: Configuration: 4 Alpha 21164 EV 56, 533Mhz - Fast Ethernet

Nœud	Cache L3/L2/L1	Disque	Noyau Linux	Taille `/work`
helmvige	4Mo/96Ko/8Ko	4Go SCSI	2.2.13-0.9	1Go
grimgerde	4Mo/96Ko/8Ko	4Go SCSI	2.2.13-0.9	4Go
siegrune	4Mo/96Ko/8Ko	4Go SCSI	2.2.5-16	4Go
rossweisse	2Mo/94Ko/8Ko	8Go SCSI	2.2.5-16	4Go

Sur la Table 3 nous avons les résultats d'expérimentation du tri séquentiel externe qui utilise l'algorithme *polyphase merge sort* qui est aussi la brique séquentielle utilisée dans les algorithmes parallèles. Remarquons qu'une taille de problème de 33554432 entiers correspond à $33554432 * 4 = 134217728$ octets, soit 134Mo. Remarquons également que le débit maximum pour les répertoires `/work` (mesuré) est de 20Mo/s pour siegrune, grimgerde et helmvige et de 40Mo/s pour rossweisse. Notre conclusion est que helmvige et grimgerde sont 4 fois plus performantes que siegrune et rossweisse pour trier ! À priori, la lecture des caractéristiques constructeur données à la Table 2 contredit cela. En fait, l'explication technique tient à ce que, au moment des expérimentations, rossweisse et siegrune sont chargés quatre fois plus que helmvige et grimgerde. Nous configurons ainsi le vecteur de performance de l'algorithme de tri parallèle d'inspiration PSRS avec les valeurs $\{1, 1, 4, 4\}$ et nous garantissons jusqu'à la fin de nos mesures que les machines restent dans les mêmes conditions de fonctionnement que pour les expérimentations en séquentiel.

Enfin, nous remarquons qu'en extrapolant nos résultats nous pouvons trier 268435456 octets en 886s que l'on peut comparer au 4.5Go triés en 886s sur un Pentium III 500Mhz sous Windows et avec et un disque de 30Mo/s de taux de transfert [3] (soit 16 fois plus que nous), mesure effectuée en 2000 dont on parlait en amont dans ce document. Cependant, on ne trie pas la même chose: pour nous ce sont des entiers, pour minute sort des enregistrements de 100 octects.

3. Voir les deux références `http://research.microsoft.com/barc/SortBenchmark/` et `http://www.stenograph.com/sorting/`

Avec notre tri "polyphase merge sort" nous pouvons aussi trier des enregistrements de 100 octets pour un coût, sur Alpha 500Mhz, de 18.5s pour 16Mo soit un équivalent de 766Mo sur 886s. Il semblerait même que nous pourions trier les 16Mo (d'enregistrement de 100 octets) plus rapidement en utilisant 50 fichiers intermédiaires[4].

TAB. 3: Tri externe séquentiel sur cluster (cf. Table 2)

Taille	Temps Exe. Moyen (s)	Écart type
	Helmvige	
2097152	22.92146	0.45283
4194304	51.17832	1.99283
8388608	111.40898	1.48268
16777216	235.74163	2.67709
33554432	492.02380	9.74561
	Siegrune	
2097152	88.94593	1.85451
4194304	188.71978	3.41997
8388608	409.09711	36.13593
16777216	909.34783	81.67
33554432	1910.8261	160.60827
	Rossweisse	
2097152	95.40269	1.09854
4194304	204.66360	4.59815
8388608	428.42470	3.35943
16777216	951.22738	77.4042
33554432	1998.72261	152.8972
	Grimgerde	
2097152	24.88658	10.20334
4194304	44.55758	0.86754
8388608	96.29102	1.46595
16777216	212.82059	2.54191
33554432	443.86681	10.12

Execution time metrics for benchmark 0
Algorithm: polyphase merge sort

Notons aussi que notre exécutable pour Alpha a été produit avec `gcc` qui n'est pas, et de loin, le meilleur compilateur possible : `ccc` (Compaq C Compiler) peut produire des codes plus rapides, dans certains cas. Cependant, nous n'avons pas relevé d'écart

4. Les expérimentations précédentes se sont faites avec 5 fichiers intermédiaires

majeur entre une compilation `gcc` et `ccc` pour le tri "polyphase merge sort". Techniquement, on peut dire aussi dire qu'en installant un Linux de base, il n'est pas toujours courant d'installer les pilotes disques les plus adaptés à son disque si bien que les performances disques sous Linux sous parfois moins bonnes que sous Windows car les pilotes sont capables de faire du DMA, ce qui est souvent assez performant par rapport aux autres modes de programmation des interfaces.

Passons maintenant au cas parallèle et discutons des performances de l'algorithme façon PSRS. Avec le cluster tel qu'il est décrit à la Table 2 et avec un vecteur de performance ne contenant que des valeurs 1, nous pouvons obtenir, avec Fast-Ethernet, des résultats moins bons que le cas séquentiel (la taille des paquets de données est de 8 entiers) : nous obtenons un temps d'exécution de $133,6$ secondes.

Par contre, toujours avec Fast-Ethernet mais avec des paquets de données de 8K nous obtenons un tri de 2097152 entiers en 32.6 secondes.

Examinons maintenant à la Table 4 les résultats d'expérimentations pour trier sur disque $2^{24} = 16777216$ entiers sur 4 processeurs en prenant soit un vecteur de performance ne contenant que des valeurs 1 soit rempli avec les valeurs $\{1, 1, 4, 4\}$. Plaçons nous d'abord dans un cas pas très favorable où le réseaux d'interconnexion est Fast-Ethernet (et pas Myrinet).

TAB. 4: Tri externe parallèle sur cluster (cf. Table 2), taille des messages: 32Ko, 15 fichiers intermédiaires, 30 expériences

Taille	TEM (s)	Écart type	Mean	Max	S(max)
Performance : $\{1,1,1,1\}$; Fast-Ethernet					
16777216	303.94	9.173	4193043.8	4204494	1.00273
Performance : $\{1,1,4,4\}$; Fast-Ethernet					
16777220	155.41	3.645	6816502.4	7342910	1.094
Performance : $\{1,1,4,4\}$; Myrinet					
16777220	155.43	3.465	6293368.5	7341545	1.093

Execution time metrics for benchmark 0
Algorithm: external PSRS

Les six colonnes de la Table 4 correspondent à (de gauche à droite) : taille du problème en nombre d'entiers, temps moyen d'exécution exprimé en secondes, écart type du temps d'exécution, moyenne de la taille de toutes les partitions finales (l'optimal est pour le cas où le vecteur de performance ne contient que des valeurs 1 est : 4194304), la taille de la partition maximale, la "sublist expansion metric" qui est le ratio entre

les valeurs de la colonne 5 et 4194304 (pour la cas où le vecteur de performance ne contient que des valeurs 1).

Nous remarquons que la "sublist expansion metric" est en pratique très proche de l'optimal 1. Ainsi la charge est complétement maitrisée. Cependant nous trouvons que le temps d'exécution est moins bon que le temps séquentiel (235s) pour trier la même quantité d'information sur Helmvige. Il est cependant meilleur que le temps séquentiel d'éxécution sur Siegrune (909s). Dans ce cas le gain avec 4 processeurs est de 3.

Le plus petit commun multiple de $\{1, 1, 4, 4\}$ étant 4, nous pouvons choisir la taille 16777220 comme taille de problème pour les deux dernières lignes de la Table 4. Ces deux lignes concernent une expérimentation sous Fast-Ethernet et un expérimentation sous Myrinet. L'optimal pour les deux processeurs les plus lents (Siegrune et Rossweisse) est de 1677722 alors que l'optimal pour les deux processeurs les plus rapides (Helmvige et Grimgerde) est 6710888 entiers. Sur la Table 4, la colonne Mean des cas non-homogènes donne la moyenne des données traitées par les deux processeurs les plus rapides, la colonne Max donne la plus grande taille des données traitées sur les deux processeurs les plus rapides et la colonne S(Max) donne la "sublist expansion metric" pour les deux processeurs les plus rapides. Là encore nous notons une valeur proche de la valeur optimale qui est 1.

Par rapport au temps d'exécution séquentiel le plus favorable (212s) nous obtenons un gain de 1.37 ; par rapport au temps d'exécution séquentiel le plus dévavorable (951s) nous obtenons un gain de 6.13 en utilisant 4 processeurs. Noter qu'un gain plus grand que le nombre de processeurs est possible ! Il n'y a rien de bien mystérieux à cela ! Nous obtenons un gain de presque 2 (1.96) par rapport à la configuration homogène (le vecteur de performance ne contient que des 1)...ce qui valide notre approche.

Enfin, nous notons que l'exécution avec Myrinet comme support de communication n'améliore pas les performances du temps d'exécution. L'explication tient dans le fait que l'application communique peut (mais bien). De plus, le temps d'exécution ne prend en compte ni la phase de distribution ni la phase de collecte des données. Les temps des calculs locaux et les accès aux disques sont prépondérants ici. Nous pouvions nous y attendre. Cette conclusion conforte l'idée que l'algorithme et les implémentations que nous proposons sont des canditats sérieux lorsque le réseau de communication n'est pas dès plus rapide !

5. Autres approches du tri externe hétérogène

Nous avons aussi étudié deux autres approches, toujours parmi les algorithmes par échantillonnage. Nous donnons simplement quelques explications élémentaires. Le tableau 5 de la page 113 permet de comparer les approches tant sur le point de

l'espace mémoire utiliséee, que du facteur d'équilibrage mesuré ou encore du nombre de fichiers intermédiaires utilisés.

Les deux autres approches étudiées ne réalisent pas de tri initial sur les portions détenues par chacun des processeurs et elle se distinguent par le choix des pivots. Ces approches s'apparentent à l'approche de :

Huang [HC83] : on sur-échantillonne le nombre de pivots c'est à dire que l'on en sélectionne plus que pour le cas PSRS ;

Li et Sevcik [LS94] : pour une liste initiale non triée de taille n, $(pk-1)$ pivots (avec $k \geq 2$) partitionnent la liste en $p * k$ sous listes telles que la taille de la plus grande sous liste est plus petite ou égale à n/p avec une probabilité d'au moins $1-2p(1-(1/(2p)))^{pk}$. Rappelons que p est le nombre de processeurs.

Pour les deux approches on sélectionne un nombre de pivots qui est fonction non pas du nombre de processeurs p directement mais de la somme des valeurs contenues dans le vecteur de performance. L'idée intuitive est la suivante : plus la machine est hétérogène et donc plus les valeurs rangées dans le vecteur de performance sont différentes, plus il y aura de pivots sélectionnés et meilleur sera l'équilibrage.

6. Conclusion

Nos travaux sur le tri en milieu hétérogène et sur disque nous ont permis de mettre en évidence quelques points étonnants :

- Premièrement, nous offrons une famile de trois algorithmes de tri pour clusters hétérogènes. Un comparatif est proposé plus loin dans le texte et à la Figure 5. C'est la première fois à notre connaissance que le problème est abordé. Les algorithmes sont validés expérimentalement.
- Deuxièmement, en contrôlant la charge, en bornant la taille des messages échangés et en utilisant notre technique de tri pour clusters hétérogènes nous obtenons (avec Fast-Ethernet) un gain de 1.37 pour 4 processeurs. De plus, tous les résultats sont obtenus avec un logiciel standard, un système d'exploitation standard (sans ajout pour gérer les entrées / sorties) et des configurations matérielles non optimisées pour permettre au niveau matériel un chevauchement entre calculs et opérations d'entrées / sorties.
- Troisièmement, nous pouvons aussi souligner que le schéma général proposé pour trier dans un contexte hétérogène qui est fondé sur l'échantillonnage de pivots fonctionne dans la pratique aussi bien dans le cas « en mémoire » que dans le cas « disque » vis à vis des métriques de temps et d'équilibrage. C'est le seul schéma à notre connaissance de tri en milieu hétérogène.

Criteria	H-PSS	H-PSRS	H-PSOP
Number of candidates	$6 * p' * \log_2(p')$	$p' * (p-1)$	$4 * p' * p' * \log_2(p')$
Number of pivots	$P-1$	$P-1$	$4 * p' * \log_2(p') - 1$
Initial sort	No	Yes	No
Load balance (theory)	Algorithm manages partitions of size n/p' with a probability which is a function of the number of candidates.	No processor has more that two times its initial load	Algorithm manages partitions of size n/p' with a probability which is a function of the number of candidates.
Load balance (measured)	$\pm 15\%$ of the optimal and in the worst case and on one processor.	$\pm 0.1\%$ of the optimal value	$\pm 0.01\%$ of the optimal value
Number of files created / proc	$15 + P + 1$	$15 + P + 1$	$15 + 4 * p' * \log_2(p')$
Sensitivity to duplicates	?	No, until a bound of n/p' duplicates	?
Message sizes	32KB	32KB	32KB
Allocated memory	$8K * sizeof(int) + 6 * p' * \log_2(p')$	$8K * sizeof(int) + p' * (p-1)$	$8K * sizeof(int) + \mathcal{O}(p')$

TAB. 5: *Summary of main properties of algorithms*

La Table 5 113 résume les propriétés essentielles de nos algorithmes et elle permet de les comparer en terme de ressources utilisées et d'équilibrage. Pour ce tableau le terme << H-PSS >> signifie *Heterogeneous Parallel Sample Sort*, le terme << H-PSRS >> signifie *Heterogeneous Parallel Sorting by Regular Sampling* et enfin le terme << H-PSOP >> signifie *Heterogeneous Parallel Sorting by OverPartitioning*. Le lecteur doit en particulier noter que l'utilisation de la RAM est très petite ainsi que le nombre de fichiers intermédiaires.

Sur la Table 5, la constante 15 est le nombre de fichiers intermédiaires utilisés par l'implémentation du polyphase mergesort., p' est la somme des valeurs contenues dans le vecteur de performance, P est le nombre de processeurs et n est la taille du problème.

Les résultats présentés dans les colonnes H-PSS et H-PSOP ont été mené conjointement avec Hazem Fkaier et Mohamed Jemni et ils sont en cours de pré-publication.

Ils peuvent éventuellement être consultés à l'adresse :

`http://www.laria.u-picardie.fr/~cerin/`

L'intérêt de l'algorithmique avec les disques tient à notre avis dans le fait que les paradigmes qui se sont avérés payants dans l'obtention de résultats, sont parfois un peu différents de ceux de l'algorithmique PRAM ou alors ils doivent être adaptés.

Ce qui différencie un peu plus encore l'algorithmique out-of-core parallèle et l'algorithmique parallèle PRAM tient à ce que l'on a encore plus intérêt en algorithmique avec les disques, à minimiser les échanges de données entre processeurs car les implémentations performantes de transfert disque à disque ne sont pas courantes. Autrement dit on doit avoir comme stratégie principale d'utiliser d'une part des solutions séquentielles out-of-core performantes qui vont traiter un sous problème du problème initial et d'autre part de reconstruire la solution finale à partir des solutions locales avec un minimum de communication entre les disques.

Un chapitre ultérieur, dans la partie perspective, traitera des problèmes de structures de données disques. Un autre chapitre de perspective s'intéressera aux différences entre les problèmes d'optimisation du cache processeur et les techniques d'optimisation des accès aux disques. Ces deux occasions permettront de distinguer et de comparer un peu mieux les techniques de gestion des disques des techniques de gestion des caches.

Nos recherches s'orientent principalement sur l'obtention de performances lorsque les disques doivent être pris en compte.

Cinquième partie

Entrées sorties hautes performances

CHAPITRE 1

Introduction

NOUS ENVISAGEONS, dans cette partie, les différentes poursuites d'étude. Elles concernent deux grands thèmes liés principalement par les problématiques des disques : le stockage parallèle dans les dispositifs de calcul global (chapitre 4), la collecte l'utilisation de traces d'événements pour le calcul global (chapitre 5). Le chapitre 2 donne quelques éléments d'évolution attendues des architectures de clusters et de systèmes de disques alors que le chapitre 3 introduit le calcul global.

En fait, dans les chapitres 3 et 4 il s'agit, de passer au calcul et au stockage parallèle à large échelle, lorsque le réseau de communication est Internet c'est à dire pour un système caractérisé par un débit de communication très inférieur au débit mémoire et caractérisé aussi par le fait que certains sites peuvent se déconnecter. De manière générale il s'agit d'étudier ce que l'on appelle maintenant le « calcul global » d'un point de vue algorithmique expérimentale, y compris l'étude les langages de programmation de calcul global pour Internet.

Les réalisations logicielles proposées devront s'intégrer dans un dispositif appelé XtremWeb qui est une plate-forme de calcul global générique développée à Orsay. XtremWeb est une des composantes du projet CGP2P (Calcul Global Pair à Pair). Ce projet fait partie de l'ACI (Actions Concertées Incitatives) GRID[1] labélisé fin 2001. Le laboratoire de recherche en informatique d'Amiens (LaRIA) s'occupe dans ce projet à la fois des problèmes de stockage (mise en œuvre des services de base comme les lectures et les écritures sur les disques distants) et de la fouille des données.

D'un point de vue général, XtremWeb adopte un style de calcul qui n'exclut ni ne dépend fortement de points de contrôle centralisés. XtremWeb implémente par exemple l'idée que les machines peuvent communiquer deux à deux sans l'entremise d'une machine intermédiaire qui négocierait la mise en relation.

Cette technologie accroît potentiellement l'utilisation des ressources disponibles dans Internet parce qu'elle n'utilise pas de serveurs : les machines peut aussi bien être clientes que serveurs. Par exemple, une architecture de routage distribué peut accroître la bande passante en équilibrant le trafic de sorte que les pics de charge soient amortis. Les dispositifs pairs à pairs peuvent aussi potentiellement améliorer la résistance aux défaillances en dupliquant les données.

Une liste non exhaustive des applications ou des services pairs à pairs inclut :

(1) les serveurs de nommage à base de répertoire distribué (DNS) ;
(2) les systèmes de fichiers qui supportent les mécanismes de déconnexions ;
(3) les systèmes massivement parallèles ayant besoin de beaucoup de ressources (CPU, disques...) sur les nœuds du réseau ;

1. Voir `http://www.recherche.gouv.fr/recherche/aci/grid.htm`

(4) les systèmes de messagerie électroniques qui ne sont pas seulement dépendants de serveurs de messagerie préconfigurés ;

(5) les systèmes à large échelle et tolérants aux fautes.

1. Application à un service de tri

Concernant le tri (voir le chapitre 5), nous pensons faire évoluer principalement notre travail afin de proposer des techniques de tri efficaces sur réseaux de PC et tenant compte de la hiérarchie mémoire. Une étude générale portant sur les relations et les techniques algorithmiques de gestion des caches et des disques a d'ores et déjà été lancée. Le problème du tri est une nouvelle fois utilisé pour ancrer une idée et essayer ensuite de généraliser les résultats obtenus à d'autres problèmes.

Les raisons principales quant à réutiliser le travail précédent sont les suivantes. Premièrement, le problème du tri est et restera un bon problème pour évaluer des systèmes parce qu'il fait intervenir des accès à la mémoire de manière non prédictible. Deuxièmement, la famille des tris par échantillonnage que nous venons d'étudier est une famille certainement plus avantageuse dans le cadre des études listées ci dessus que des tris pensés par exemple dans le cadre de PRAM parce qu'ils ne font qu'une phase de communication. Pour simplifier un peu, quand il s'agit d'échanger des données avec les disques ou via un support de communication comme Internet, moins on communique, meilleures sont les performances. Autrement dit, il nous semble que l'avenir est au calcul parallèle à *gros grain*, c'est à dire à des << grosses tâches >> qui communiquent le moins possible. Nous souhaitons montrer que ce qui fonctionne bien (au moins pour le tri), d'un point de vue algorithmique expérimentale pour des clusters de PC, fonctionnera bien quand les PC seront reliés par Internet au lieu d'Ethernet.

Il est connu en bases de données[2] que l'opération de jointure [UW02] de deux relations A, B s'implémente par deux techniques dont une est le tri.

Le pseudo code suivant implémente les différentes phases de l'opération de jointure des deux relations A et B :

```
foreach tuple r in A do
   foreach tuple s in B do
      if (A Job field)_r = (B Job field)_s then
         add <r,s> in result
```

2. Voir `http://www-2.cs.cmu.edu/afs/cs/academic/class/15721-f01/www/readings.html` qui recouvre une partie non négligeable des points abordés au final dans ce mémoire. Voir aussi les informations en ligne de Jeffrey D. Ullman sur `http://www-db.stanford.edu/~ullman/dscb.html`

dont le coût en nombre d'opérations est en $\mathcal{O}(n^2)$ (n étant le nombre de tuples dans chacune des deux relations). Mais on peut utiliser un tri de la façon suivante :

```
resA <-- Sort A according to the Job field
resB <-- Sort B according to the Job field
Merge ResA and ResB
```

dont le coût théorique est meilleur que le coût précédent. Dans ce cas, notre étude sur le tri externe en milieu hétérogène est un candidat sérieux pour l'implémentation de l'opération de jointure dans des bases de données parallèles tournant sur des processeurs à des vitesses différentes.

C'est donc dans ces mesures que le travail précédent sur le tri apportera un éclairage effectif.

2. Organisation de la partie perspective

Nous évoquons maintenant précisément nos perspectives de travail qui sont ciblées sur les entrées/sorties parallèles hautes performances. Nous nous intéressons plus particulièrement à l'obtention de performance dans un système de calcul global lorsque les disques sont en jeu.

Très peu de travail a été accompli pour cette thématique. Nous spécialisons nos études sur la *représentation des données sur disques* dans un système de calcul global (chapitre 4) et au *repérage des événements pertinents* de l'activités des machines connectées au système global (chapitre 5).

L'émergence de ces deux problématiques est très importante pour le futur des systèmes globaux puisque la plus grande partie des données résident sur disques

Les objectifs à moyen terme poursuivi dans le chapitre 3 sont de fournir un service de stockage pour des structures de données arborescentes. En effet, ces structures sont très largement utilisées en bases de données, par exemples les B-arbres ou en géométrie algorithmique (R-tree...). Il s'agira d'examiner comment ces structures peuvent « passer à l'échelle ». Le service offert inclura les opérations de base associées à toutes structures de données : insertion, suppression, consultation.

Les objectifs à moyen terme du chapitre 5 sont de fournir un service qui vient épauler l'ordonnanceur de travail du système global. En effet, il s'agira à terme de décider quelles sont les machines sur lesquelles ont pourra lancer un travail. Comme le placement de tâches est un problème très difficile, nous pensons mettre en place des heuristiques basées sur ce que serait l'état global du système.

Nous commençons dans le chapitre 2 par exposer quelques évolutions attendues des sytèmes de disques puis, dans le chapitre 3 nous présentons exhaustivement les

concepts des systèmes de calcul globaux et ceci au moyen de l'exemple de la plate-forme `XtremWeb`.

CHAPITRE 2

Quelques évolutions attendues

1. Introduction

CONCERNANT LES ÉVOLUTIONS en matière d'architecture des clusters, il y a deux faits durables que l'on peut observer depuis quelques années. Premièrement, c'est l'avènement des systèmes d'interconnexion performants (on dit des SAN pour System Area Networks) qui a permis la maturation de l'architecture des clusters, en particulier en ce qui concerne la bande passante et la latence. La technologie VIA (Virtual Interface Architecture ; voir `http://www.viarch.org`) est un récent standard qui incorpore la plupart de ces avancées.

Dans ce contexte et pour cette technologie, il devient très probant de construire des systèmes distribués à mémoire partagée. En 2000, nous avons vu l'arrivée d'un premier prototype (à notre connaissance) de clusters de PC offrant par l'intermédiaire de VIA, la vision d'une mémoire partagée. Il s'agit de l'outil logiciel disponible à l'adresse suivante :

`http://discolab.rutgers.edu/projects/dsm/index.html`

Les choix architecturaux majeurs effectués pour cet outil ont été d'une part d'implémenter l'abstraction de la mémoire distribuée (DSM) au dessus d'une librairie de communication rapide et d'autre part d'utiliser un modèle de cohérence mémoire qui est une variation du modèle "release consistency" classique. Pour le programmeur qui voit un unique espace d'adressage, la difficulté de translation d'un code BSP/MPI de la famille des algorithmes par échantillonnage n'est pas d'une grande difficulté. Comme tous nos algorithmes parallèles que nous avons présentés ne nécessitent qu'une seule étape de communication, on peut penser que les accès à la mémoire distante requis par l'algorithme DSM correspondant seront eux aussi peu nombreux comparés au nombre de références aux mémoires locales... ce qui est plutôt bon signe en terme de performance.

La seconde tendance, plus récente, est liée à un glissement du domaine d'application vers des applications à base de serveurs (vidéo à la demande, bases documentaires) alors que jusqu'à présent les clusters étaient conçus plutôt comme une alternative pour faire du calcul numérique intensif à moindre coût. Si l'on admet que le futur sera fait de dispositifs embarqués qui accèdent à des bases de données, alors ce sont les entrées / sorties (I/O) qui prennent de l'importance. De plus, on voit bien que le système informatique de gestion des I/O doit pouvoir s'adapter rapidement lorsqu'on ajoute un système embarqué supplémentaire dans le système. On peut même imaginer que le système se reconfigure automatiquement pour pouvoir assurer de bonnes performances d'entrées / sorties.

Les constructeurs de systèmes de stockage distribués ont à résoudre le problème suivant : qui effectue le travail de fournir des services de stockage ? Les approches hiérarchiques qui incluent la « virtualisation » du service de stockage prennent une importance nouvelle dans le cadre des systèmes pairs à pairs puisque toutes les machines sont susceptibles d'offrir un service de stockage.

Comme article récent sur le sujet, nous pouvons citer [SRK01] qui propose un service global de stockage dans le projet OceanStore. L'article fixe bien les problématiques centrales, à savoir :

(1) la localisation des objets et l'infrastructure de routage des messages ;
(2) la dispersion et le rassemblement d'objets fragmentés ;
(3) la mise en œuvre d'un protocole d'agrément bysantin pour sérialiser et authentifier les mises à jour d'objets ;
(4) la mise en œuvre d'un ensemble d'algorithmes pour observer l'usage du système.

Concernant les protocoles étudiés dans le cadre des systèmes de stockage, nous pouvons citer le protocole *Internet Backplane Protocol* (IBP) qui est discuté dans [JSPM01] par exemple. Ce protocole autorise les applications à contrôler les différentes étapes d'échange des données ce qui n'est pas observable avec OceanStore.

Examinons maintenant les évolutions prévues en matière de matériel.

2. Synthèse des architectures des systèmes de stockage

2.1. InfiniBand. L'idée d'interconnexion de serveurs de stockage fait son chemin par exemple dans l'architecture InfiniBand [1] qui est une technologie « d'inspiration cluster ». Rapidement, on peut dire que le bus des entrées/sorties (bus I/O) est remplacé par un dispositif d'interconnexion de canaux commutés et que les liens de communication interconnectant en particulier un réseau de disques sont prévus pour fonctionner à 2,5Gbits/s.

L'équipe de Dhabalewar K. Panda [2] étudie en ce moment l'intégration d'un émulateur d'InfiniBand au-dessus de Myrinet [WGAP01]. Le besoin d'un tel outil est motivé par l'absence (début 2001) de matériels implémentant InfiniBand ou de kits de développement ce qui retarde l'arrivée, en simultanée avec le matériel, d'applications tirant partie de la technologie. L'émulateur est construit au-dessus de l'implémentation VIA au dessus de Myrinet de Berkeley [3]. Parmi les défis relevés,

1. `http://www.infinibandta.org`
2. Voir `http://www.cis.ohio-state.edu/~panda/`
3. Voir : `http://www.cs.berkeley.edu/~philipb/via`

nous pouvons citer le support d'un très grand nombre de files de gestion des communications, les mécanismes de gestion des évènements, la programmation des drivers de programmation.

Un autre élément fondamental de l'émulateur est d'offrir un support pour faire des accès privilégiés aux ressources (des utilisateurs peuvent enregistrer leurs propres tampons dans l'espace noyau et contrôler de manière fine l'accès aux << ressources InfiniBand >>). Ceci est fait pour autoriser en particulier des performances de communication élevées en évitant des appels au système. Le concept d'accès privilégié pourrait être étendu afin de contrôler et de hiérarchiser l'accès au noyau par des utilisateurs qui restent à identifier et en qui le système doit avoir confiance. De même la protection des données (son cryptage) n'est pas assurée pour l'instant dans le simulateur. Ce sont des services indispensables si l'on veut développer à l'échelle d'une grande entreprise, d'une université, un cluster de machines sous InfiniBand.

De plus, ces services devraient fonctionner en dessous de IP. Pour l'instant, le protocole de communication se fait via Myrinet et il n'y a pas d'IP dans le simulateur. Son intégration est un challenge. De plus, si l'on veut que les flux de communication respectent des contraintes fortes liées au temps, la couche de transport est à revoir. L'IETF (Internet Engineering Task Force[4]), organisme chargé de la définition de standards pour Internet, a proposé une architecture nommée DiffServ[5], apportant des services améliorés pour la transmission de paquets temps-réel au niveau de la couche IP. Dans ce cadre, le laboratoire RESAM de l'université de Lyon 1 travaille sur un nouveau modèle de service nommé << Fair Forwarding >> , dont la mise en place est simple et immédiate par rapport aux recommandations de l'IETF. Le modèle offre deux services d'acheminement des paquets aux performances différentes. Au final, il serait intéressant d'envisager le type de service << à la DiffServ >> dans un outil comme l'émulateur de l'université d'Ohio afin de capter IP.

2.2. RapidIO. RapidIO est une autre technologie à l'étude pour accélérer les entrées / sorties. Ce qui distingue RapidIO et InfiniBand est assez profond quand on y regarde de très prêt. Le serveur Internet (ouvert ici à tous sans péage) consacré à RapidIO est `http://www.rapidio.org`.

RapidIO et InfiniBand partagent le fait que se sont des dispositifs d'interconnexion par commutation de paquets (packet switched) et sont proposés comme des standards industriels à la mi-2001. Originellement, RapidIO a été conçu comme un bus commun d'interface des différents processeurs de chez Motorola. L'interface devait tenir dans 20000 circuits logiques, implémentable en technologie FPGA. Au contraire, InfiniBand a été conçu comme un standard avec des caractéristiques plus riches que celles de RapidIO en matière de communications inter-processeurs. Ceci parce que plus de

4. Voir `http://www.ietf.org`
5. Voir `http://www.ietf.org/html.charters/diffserv-charter.html`

200 industriels participent au processus de standardisation parmi lesquels on trouve IBM, Sun Microsystems, HP, Intel, CISCO, Lucent, 3COM, Microsoft. RapidIO a été poussé initialement par Motorola et Mercury qui n'avaient pas d'expérience sur les technologies de commutation de paquets. A ces deux industriels on peut maintenant ajouter Alcatel, Cisco Systems, EMC Corporation, Ericsson, Lucent Technologies, et Nortel Networks.

2.3. Comparaison. Avec InfiniBand de nombreuses applications sont supportées grâce à des types de paquets définis par l'utilisateur ou en utilisant les types de paquets prédéfinis. Avec RapidIO, la taille des paquets (utiles) est de 256 octets ce qui est faible mais permet d'avoir des tailles de buffer FIFO faibles. Les entêtes et queue de messages totalisent 37 octets ce qui conduit à un surcoût (overhead) de $37/(256+37) = 12.6\%$.

On rapporte aussi que la latence de communication avec InfiniBand est supérieure à celle de RapidIO. Le problème c'est que l'on ne parle pas toujours de la même chose. En effet, quand on utilise InfiniBand pour émuler IP, InfiniBand fournit des services d'authentification, de vérification et de qualité de service. Il est clair que la latence des communications est plus grande que celle fournit par RapidIO mais RapidIO n'offre pas tous les services précédents.

Concernant l'interface physique, les liens de communication de RapidIO sont entre 500-622 Mbits/sec tandis que InfiniBand utilise des liens à 2.5 Gbit/sec. RapidIO a des liens séparés pour l'horloge alors qu'InfiniBand la code << dans les données >> sans ajouter de liens physiques. Pour atteindre le gigabit par seconde RapidIO nécessite une interface 16bits avec 40 fils (64 signal + 8 clock + 4 frame = 40) alors qu'il suffit de 16 fils avec InfiniBand pour le même taux de transfert.

Enfin, on peut ajouter qu'InfiniBand est prévu de s'interfacer avec les technologies optiques alors que RapidIO ne l'est pas.

Ce résumé rapide des caractéristiques matérielles des liaisons proposées nous incitent à conclure qu'InfiniBand a des avantages certains vis à vis de RapidIO... mais pour combien de temps? Cependant Infiniband est considéré par beaucoup comme impossible a implémenter en entier.

Spécifier des algorithmes avec cette technologie de stockage dans le cadre du modèle PDM ne trouve d'intérêt, à notre avis, que pour le cas où nous avons affaire à des réseaux hiérarchiques de tels dispositifs. Par exemple, il s'agirait d'écrire une spécification du tri parallèle lorsque le système parallèle est constitué de clusters de processeurs et de clusters de disques interconnectés par différents types de réseaux ou bien des clusters de processeurs accédant à des disques reliés par InfiniBand, ces clusters étant interconnectés par une hiérarchie de réseaux.

3. D'un point de vue algorithmique

Disposer d'un réseau rapide entre les disques a une implication certaine sur la façon de concevoir un algorithme << out of core >> . On peut même se demander si tout algorithme << in core >> bien écrit, développé pour un cluster d'aujourd'hui se comportera très bien, sans modification aucune, sur un cluster de technologie InfiniBand. Serait-il donc suffisant d'étudier les problèmes dans le cadre << in-core >> seulement ?

3.1. Tigris. Le projet `Tigris`[6] de Matt Welsh et David Culler se veut une implémentation au-dessus de `Java` pour gérer efficacement les problèmes de calcul haute performance et les entrées / sorties. `Tigris` est construit à partir de `Jaguar` [WC00a] qui permet l'accès aux ressources bas niveau - y compris les ressources réseaux (Myrinet en l'occurence) et les entrées sorties disques. L'implémentation est réalisée au dessus de la version VIA de Berkeley et les auteurs obtiennent des performances (round-trip time) de 73µs pour les messages courts et un pic de bande passante de 488 Mbits/second...ce qui est, à 1% près, la valeur obtenue pour les performances de 'Berkeley Linux VIA' accédé depuis un code `C`.

D'un point de vue technique, `Tigris` est un système de gestion des I/O ainsi qu'un modèle de programmation qui autorise les ressources à être automatiquement équilibrées sur le cluster ; c'est une ré-implémentation en Java d'un autre système de Berkeley, `River`[7]. `Tigris` est conçu avec l'idée d'un modèle de programmation 'dataflow' et implémente une file distribuée qui permet aux données d'être soit sur la machine qui les produit soit sur la machine qui en a besoin et ceci de manière dynamique. `Tigris` est donc un exemple d'un système de gestion des communications et des I/O pour cluster (à priori hétérogène) qui assure une forme d'équilibrage de charge dynamique.

Pour tester les performances de ces systèmes, le groupe de Berkeley a implémenté un tri externe. Les résultats sont commentés dans [ADAT+99] pour `River` et dans [WC00b] pour `Tigris`. La version du tri implémenté est un tri externe en une seule passe selon l'algorithme suivant : *(a)* un ensemble de pivots est choisi de manière statique au départ (on ne sait pas combien) *(b)* les données sur les disques sont réparties sur d'autres disques << définis >> par un pivot *(c)* sur chaque disque les données sont triées, et enfin elles sont ré-écrites pour former, en gros, un seul fichier. Aucune discussion n'est faite concernant le choix des pivots, ni leur nombre. Ainsi nous ne sommes pas dans un cas très défavorable concernant la gestion des demandes d'I/O - un tri à plusieurs phases serait un cas plus défavorable (c'est donc le choix que nous avons fait). De plus, le fait de ne pas maitriser la répartition (par une borne) a pour effet de créer artificiellement des déséquilibres de charge alors que l'on peut les maitriser. Ainsi les résultats présentés ne sont pas les meilleurs possibles vis à vis de l'équilibrage. La

6. Voir `http://www.cs.berkeley.edu/~mdw`

7. Voir : `http://www.cs.berkeley.edu/~remzi/Postscript/river.ps`

comparaison des performances ne pourra pas s'effectuer avec le meilleur cas possible - nous voulons dire avec le(s) meilleur(s) algorithme(s) possible(s) pour l'équilibrage.

Deuxièmement, la comparaison des résultats expérimentaux se fait avec un cas qualifié d'idéal et correspondant au cas où le tri parallèle lit et écrit les données avec la bande passante disque maximale, prend 0 seconde pour trier en mémoire, et tout ceci sans surcoût lié à la parallélisation. À priori, les systèmes sont conçus pour fonctionner en milieu hétérogène, disques compris. Que veut dire alors le terme « bande passante disque maximale » ? Ce n'est pas clair ! Au final, nous ne trouvons pas que le choix de la situation avec laquelle on compare soit un bon choix.

Troisièmement, la taille des problèmes traités avec `Tigris` est au plus de 20M octets sur 8 nœuds. Peut on parler de calcul 'out-of-core' ?

Quatrièmement, on nous présente des temps d'exécution moyens par nœud (de l'ordre de 730ms pour 20M octets) mais on ne nous parle pas de la variance qui, on peut le supposer, est grande car il n'y a pas de maitrise de l'équilibrage ni dans le cas homogène ni dans le cas hétérogène.

Cinquièmement, on ne précise pas comment est mesuré l'hétérogénéité de la plate-forme (les disques sont ils hétérogènes? les CPU ?...)

3.2. Conclusion. Les outils de Berkeley sont intéressants et prometteurs mais pour les évaluer avec un tri externe il faudrait soigner les implémentations et revoir les algorithmes sélectionnés pour réaliser le travail.

Nous pensons que tous les algorithmes présentés dans ce mémoire ont beaucoup plus d'avantages à être utilisés pour tester des systèmes comme `Tigris`. Le risque à utiliser la méthodologie d'expérimentation de `Tigris` est d'obtenir un retour non reproductible du fait du manque de garanties proposées. Les résultats d'évaluation de `Tigris` sont donc à prendre avec un certain recul.

Par contre, les garanties que nous offrons à travers tous nos codes de tri concernent à la fois les temps d'exécution mais aussi l'équilibrage des charges de chaque nœud du cluster. Par ailleurs les algorithmes sont conçus pour tenir compte de *l'hétérogénéité des processeurs* du cluster.

CHAPITRE 3

Calcul global : Internet comme support de communication

LES PLATES-FORMES DE MÉTA-COMPUTING et de calcul global (Global Computing) comme Seti@Home[1], Globus[2], Legion, Ninf [FK97,GG99,Gri00,GWF+94, SNS+97,SNM+00] fonctionnent selon le principe de la *jachère de calculs massivement distribués* sur des PCs connectés sur Internet. Il s'agit d'exécuter des calculs (numériques essentiellement à l'heure actuelle) quand les PC sont peu ou pas chargés. Dans ces systèmes il y a distinctement des postes clients et des postes serveurs. Les systèmes distribués pair à pair (Peer to Peer[3]) comme Napster[4], Gnutella[5], Freenet[6] [Mac00, Bar00] permettent de mutualiser les ressources de stockage de PCs connectés sur Internet. Dans ces systèmes les participants sont à la fois client et serveur.

Pour l'instant il n'y a pas vraiment à l'échelle d'Internet, de dispositif fusion des deux options c'est à dire qu'il n'y a pas d'outils génériques (pour l'instant le parallélisme est trivial à exploiter, la protection des données et le cryptage ne sont pas bien assurés) permettant à la fois d'exécuter sur son PC n'importe quel code et permettant l'échange poste à poste (pair à pair).

1. XtremWeb

La plate-forme XtremWeb[7] est une plate-forme d'expérimentation de calcul global développé au laboratoire de recherche en informatique d'Orsay (LRI). Ce système est opérationnel et il est destiné à servir de support pour des expériences de calcul global et de systèmes poste à poste (peer to peer). La plate-forme XtremWeb fonctionne déjà au laboratoire de l'accélérateur linéaire (le LAL à Orsay) de l'institut national de physique nucléaire et de physique des particules (IN2P3) pour la production de simulations dans le cadre du projet d'observatoire Pierre Auger[8]. D'autres installations d'XtremWeb sont en cours en Chine, à Toronto et à Melbourne. L'ensemble des logiciels qui composent XtremWeb sont gratuits et à codes sources ouverts.

XtremWeb est composé de serveurs et de clients. Les clients examinent en permanence la charge du PC connecté. Quand celle ci devient insignifiante, le PC commence sa participation à la résolution d'un problème. Un des objectifs consiste à faire en sorte que la plate-forme soit *générique* dans le sens où n'importe quelle application

1. Voir le site `http://setiathome.free.fr/index.html`
2. Voir aussi le site `http://www.globus.org/`
3. Voir `http://www.p2pwg.org`
4. Voir aussi le site `http://www.napster.com/`
5. Voir aussi le site `http://France.gnutellaworld.net/`
6. Voir aussi le site `http://freenet.project.free.fr/`
7. Voir : `http://www.xtremweb.org`
8. Voir : `http://www.auger.org`

puisse s'exécuter dans le système. Le modèle d'exécution actuel correspond donc à la situation où ce sont les clients qui initient le travail.

Le modèle de communication retenu correspond à la situation où un client peut déposer (`put`) ou prendre (`get`) une information. Nous ne sommes pas dans un modèle par envois de messages de type `send, receive`. L'implémentation se réalise soit pas les RPC (Remote Procedure Call) ou RMI (Remote Method Invocation qui a un coût minimum mesuré de 500µs dans l'implémentation actuelle en Java ce qui est important si l'on vise des performances de calcul élevées) selon le modèle de programmation choisi. Avec des communications de ce type, la sécurité des clients est garantie par une authentification via le serveur ce qui est bien maîtrisé. La distribution du travail se fait par un unique ordonnanceur sur le serveur. Ainsi, toutes décisions de contrôle se passent via le serveur.

Pour l'instant, les points suivants comme la sécurité des participants et des données, la certification de résultats, l'ordonnancement des calculs en présence de charges fluctuantes, la conception même et la vérification des protocoles de communication et la mise en œuvre d'une interface utilisateur simplifiant l'utilisation efficace du système ne sont pas entièrement satisfaisantes dans tous les dispositifs, y compris XtremWeb. Tous ces points représentent de véritables défis auxquels on peut ajouter la possibilité de faire communiquer entre eux les PC. Autrement dit, la notion de serveur s'estompe au profit d'une architecture plutôt de type réseaux de PC (grande échelle) où tout le monde est serveur et client à la fois.

Concernant la sécurité avec XtremWeb, bien qu'actuellement le code chargé par les clients soit crypté et protégé par une clef publique privée, il faut aussi s'assurer que le code transmis ne tente pas de faire << tomber >> le client ou encore que toutes les sections du code puisse s'exécuter à partir des entrées passées en paramètre au code.

2. Le rôle d'un émulateur d'un réseaux grande échelle

Dans ce contexte, que veut dire << trier à l'échelle d'Internet >> , quand il n'y a même pas de modèle de calcul pour Internet. La mobilité [MDW99, Mil99], qu'il faut prendre ici dans un sens très large, nous parait important à modéliser. Les processseurs sur lesquels on envoie des codes souhaitent arrêter le calcul quand ils deviennent par exemple trop chargés. Les processeurs (de téléphones portables, d'assistants personnels ou de PC classiques connectés par une liaison filaire) se déconnectent du système. Dans tous ces cas, on est amené à déplacer les codes [MDP+00], voire les données pour reprendre les calculs autre part.

Pour apprécier le comportement de nos algorithmes de tri lorsque les nœuds sont des machines reliées à Internet, un outil de simulation, d'expérimentations sur réseau peut s'envisager.

L'outil Emulab[9] [Wa00] rassemble dans un même environnement les trois techniques généralement employées à ce jour : la simulation, l'émulation, et la technique de l'observation en temps réel. La simulation évalue les protocoles réseaux sous différentes conditions, par exemple de charge. L'émulation permet de coupler une simulation avec des nœuds du réseaux physique. L'observation en temps réel se fait en considérant des liens, routeurs, sites, applications, utilisateurs bien réels et bien sur le passage à l'échelle d'une telle technique est parfois problématique.

Emulab est un outil incontournable. Cependant, il ne semble pas fournir par exemple de boites à outils pour capter la mobilité, la duplication des données. Cela constitue sans doute un travail de recherche important.

3. Quel modèle de calcul?

Une idée serait d'émuler un réseaux local (LAN) sur un réseau grande échelle (WAN). Pour cela, au niveau du modèle on a besoin :

- de masquer les fluctuations de la bande passante : des garanties de service les élimine mais elles introduisent des échecs d'accès aux ressources ; cependant la mobilité permet potentiellement de découper une application afin d'établir des schémas de communication de prédilection pour l'application et donc de réduire les effets de fluctuation de la bande passante ;
- de découvrir à l'avance les échecs d'accès : difficile car Internet est un réseau asynchrone ; la construction d'un oracle est elle envisageable ?
- de masquer les sites physiques : impossible car atteindre un site ne se fait pas, dans la réalité, à un coût nul ! cependant on peut concevoir qu'un code mobile va permettre d'optimiser le placement de l'application sur les sites physiques ;
- de masquer les sites virtuels : ici se pose le problème de l'identité (ou le propriétaire) du site (ou du réseau virtuel) ; ainsi se pose la question de la garantie de l'intégrité du code mobile qui va partir dans le réseau virtuel.

Ainsi, pour construire un langage de programmation de calcul global sur Internet, il est impératif de commencer par se fixer (de manière raisonnable) ce que l'on veut observer et en particulier quelles sont les garanties techniques et contractuelles d'accès au réseau. Comme exemple de garantie technique demandée nous pouvons souhaiter par exemple que les routeurs ne soient jamais défaillants (mais uniquement les machines de calcul). Comme exemple de garantie contractuelle nous pensons que le programmeur peut souhaiter qu'en cas de migration du code celui ci ne soit pas migré dans un domaine (au sens Internet) particulier parce que le programmeur n'a pas

9. Voir : `http://www.emulab.net`

confiance en ce domaine ou parce qu'il compte privilégier des domaines particuliers. On introduit alors, au niveau de l'application, des informations qui devraient aider (à priori) l'ordonnanceur ou de manière générale le système d'exploitation sur lequel le code doit tourner à mieux utiliser les ressources du système global, XtremWeb par exemple.

Luca Cardelli présente dans une série d'articles [CG00c, CG00a, Car97, CG99a, GC99, CGG99, CGG00, Car00, CG00b, BCB99] un modèle d'algèbre de calcul à propos de la mobilité : the ambient calculus[10]. Il s'agit d'un modèle où les "computational ambients" que l'on pourrait traduire par les univers de calcul ou encore des domaines administrés[11], sont structurés hiérarchiquement et où ces univers se déplacent sous le contrôle d'agents confinés à des univers. La nouveauté par rapport aux techniques classiques qui n'envisagent la mobilité que d'objets individuels ou d'agents est que l'on propose le mouvement d'environnement (éventuellement imbriqués) qui comportent des données et des codes en train de s'exécuter. L'objectif avoué est de rendre possible le calcul sur un réseaux global, distribué avec des parties du réseau qui sont connectées au reste par intermittence et où on contrôle la mise à l'échelle c'est à dire le nombre de machines connectées.

Luca Cardelli propose alors quelques opérations pour capter ces idées :

- inclure un univers (ambient) dans un autre ;
- faire sortir un univers (ambiant) d'un environnement ;
- ouvrir un univers (rendre disponible des entités) ;
- copier un univers ;
- communiquer essentiellement par des opérations de lecture.

D'autres vues d'esprit sont présentées dans les travaux de Cardelli. Par exemple les notions de *barrières* permettent de capter :

a: la localité : elle est donnée par du code qui s'exécute entre deux barrières ou plus généralement elle est vue comme un ensemble de barrières ;

b: la mobilité de processus : c'est le fait qu'un code franchit une barrière ;

c: la sécurité : c'est la capacité ou l'incapacité qu'a un code de franchir une barrière.

Il ne serait pas inutile d'étendre les constructions de barrières au sens BSP avec ces vues ce qui permettrait d'aller vers un « BSP mobile ». PUB7 qui offre déjà la notion de groupe de synchronisation sur une barrière est un candidat potentiel à l'implémentation.

10. Voir : `http://www.luca.demon.co.uk/Ambit/Ambit.html`

11. "An ambient is like a republic with border guards" (Luca Cardelli)

4. Quel système de fichiers?

Examinons maintenant comment des enjeux du calcul numérique sur Internet peuvent être captés au niveau des systèmes de fichiers. Prenons comme exemple le système `Coda`[12] qui est un système de fichier distribué[13]. `Coda` est un projet de Carnegie Mellon qui est développé depuis 1987 par le groupe de recherche de M. Satyanarayanan et qui offre les fonctionnalités suivantes (nous laissons les termes anglo-saxons) :

(1) disconnected operation for mobile computing ;
(2) freely available under a liberal license ;
(3) high performance through client side persistent caching ;
(4) server replication ;
(5) security model for authentication, encryption and access control ;
(6) continued operation during partial network failures in server network ;
(7) network bandwith adaptation ;
(8) good scalability ;
(9) well defined semantics of sharing, even in the presence of network failures ;

Intéressons nous aux points 1 et 9 et montrons les principales différences conceptuelles entre `Coda` et les travaux plus théoriques de Cardelli tels que nous les avons présentés au paragraphe précédent. Les références [Lee00, LLS99, SKM+93, KS91] sont utiles pour fixer les termes. La notion *d'opération déconnectée* est introduite pour faire en sorte qu'un système de fichiers distribué puisse s'utiliser quand un client se déconnecte du serveur. On considére *l'avant déconnexion*, le *pendant la déconnexion* et *l'après déconnexion*. Autrement dit, on sous-entend qu'il y aura un message avertissant de la << panne temporaire du serveur >> . On considère que c'est le côté serveur qui devient inaccessible.

Ainsi, juste avant une déconnexion, le client cache les données qu'il utilisait du système de fichier distribué ; pendant la déconnexion, le client continue de fournir des services en utilisant les données cachées pour les demandes de lecture et en enregistrant les demandes de mise à jour pour les demandes d'écriture ; après la déconnexion, le client envoie les demandes de mise à jour au serveur. Parmi les précautions à prendre, mis à part les problèmes de défaut (de cache), nous pouvons citer les problèmes de mise à jour qui ne peuvent se dérouler qu'au moment de la reconnexion du serveur. Les délais pour << voir >> les dernières mises à jour doivent être gardés aussi faibles que possible. Avec des mises à jour qui représentent des fichiers de plusieurs dizaines de kilo octets et avec Internet comme support de communication,

12. Voir : `http://www.coda.cs.cmu.edu/`
13. Voir aussi d'autres logiciels libres sur `http://sourceforge.net/foundry/storage`

il n'est pas toujours raisonnable d'envoyer au serveur la mise à jour (les données modifiées) comme l'exemple suivant le montre. Imaginons un fichier de mises à jour de $95K$ octets de données à faire transiter sur une ligne de communication à $9,6Kbits/s$. Il faut $79,2$ secondes pour la mise à jour. C'est inacceptable pour l'utilisateur.

L'idée présente aujourd'hui dans `Coda` est d'intercepter les commandes lancées par l'utilisateur et, en cas de déconnexion il s'agit de relancer la commande au lieu d'effectuer une mise à jour des données modifiées ce qui est très souvent plus coûteux en temps dans la pratique. L'idée est simple et attrayante car elle réduit à priori le trafic sur le réseau. Cependant un certain nombre de questions apparaissent rapidement :

(1) Comment sont enregistrées les opérations lancées par l'utilisateur ? Quel(s) type(s) d'opérations lancées par l'utilisateur peuvent être enregistrées ? Qui est responsable de l'enregistrement ?
(2) Comment les opérations lancées par l'utilisateur sont ré-exécutées ? Est-ce que cela a un impact sur la mise à l'échelle du système de fichier ?
(3) Qu'en est-il de la correction du protocole de mise à jour ?

Les réponses apportées dans `Coda` au point 1 reposent sur la notion de *d'interception et d'interposition*. Une couche de logiciel intercepte et réinterprète les opérations qui transitent sur un support physique potentiellement « non fiable ». Un *mandataire* (appelé "surrogate" dans `Coda`) se charge de l'exécution des opérations demandées par le client en lieu et place du serveur et ceci pour des raisons d'allègement de la charge de travail du serveur ce qui favorise un passage à l'échelle.

Pour conclure rapidement, on peut dire que `Coda` est plutôt un système d'agents mobiles transférant du code (par une variante de la technique des RPC), des données et de manière générale de l'autorité et le contrôle des opérations à un tiers. À propos de la technique des RPC utilisée, on ne sait pas si elle permettrait d'accéder à une station mobile via une adresse IP fixe qui induisent des problèmes difficiles comme par exemple la gestion des régimes transitoires.

Par rapport aux idées de Cardelli, il n'y a pas dans `Coda` la notion et la gestion d'environnement aux sens « domaine Internet ». Le passage à l'échelle d'Internet de `Coda` reste à démontrer en particulier pour ce qui est des stratégies de migration des données qui dans les exemples proposés restent de taille modiques (quelques dizaines de kilo octets). L'idée de duplication des données est bien présente dans `coda` mais pas explicitement dans les idées de Cardelli. On peut se demander dans quelle mesure la maîtrise de la redondance d'information (dans un système de fichiers parallèle) couplé à l'interposition d'un tiers pour rendre des services permet de traiter des problèmes de très grande taille (plusieurs dizaine de téra-octets) ?

Il n'y a pas non plus dans `Coda` de cryptage des informations sytème échangées. Pour cela il faut aller voir le dispositif SFS (Self-certifying File System) disponible

sur `http://www.fs.net/`. Ce système de fichiers global (des utilisateurs peuvent accéder à n'importe quel serveur dans le monde et partager des fichiers avec n'importe qui) améliore aussi la sécurité du réseau local sur lequel est connecté la machine (éventuellement).

Beaucoup de systèmes de fichiers réseaux font confiance aux réseaux avec lesquels ils collaborent alors qu'un intrus peut attaquer une machine sur un brin Ethernet en exploitant les protocoles du système de fichiers qui n'est pas sécurisé. Par exemple, NFS émet << en clair des fichiers secrets >> à chaque requête qui, décodés peuvent servir à accéder à l'ensemble du système de fichiers. L'outil `SFS` ne le permet pas !

5. Ordonnancement avec les disques

Les travaux de Anna Hać [Hac89] sont une référence en matière d'algorithme distribuée pour optimiser les performances d'un système par l'utilisation conjointe de techniques de duplication de fichiers, de migration de fichiers et de processus. L'auteur propose un algorithme qui détermine s'il faut migrer un processus ou un fichier et l'endroit où il faut migrer. Il donne aussi des conditions pour dupliquer un fichier. Les décisions dépendent de l'utilisation des ressources du système, du nombre d'accès en lecture et en écriture des fichiers et des tailles des fichiers.

Les expérimentations sont faites en considérant différents "workload" (types des processus et leurs demandes en terme de ressources du système). La machine distribuée utilisée dans les expérimentation de l'article n'est plus d'actualité et la taille des fichiers traités est de quelques kilo-octets seulement. Le patron d'algorithme proposé peut servir aussi bien à des décisions de placement statiques de fichiers ou dynamiquement quand une prédiction des ressources systèmes est possible. Dans ce dernier cas, on suppose que le nombre de processus est constant pendant la phase d'évaluation du système.

Le système distribué est modélisé par 18 paramètres... ce qui n'est pas très raisonnable ! Nous n'en présentons que quelques uns afin de donner une idée de certains critères permettant la migration, ou la duplication. Ces paramètres permettent de quantifier par exemple les sites qui sont des goulets d'étranglement en s'intéressant au temps moyen d'activité de la ressource système $\mathtt{m}$ sur le site $\mathtt{h}$. Le temps moyen de service du processus de type $\mathtt{i}$ sur le site $\mathtt{h}$ est aussi défini. On dit alors que le site $\mathtt{h}$ est dans un mode de fonctionnement *non saturé* si le temps moyen de service requis par chaque type de processus sur le site $\mathtt{h}$ est plus grand que le temps d'activité moyen de chaque ressource système. Autrement dit il n'y a pas de mise en file d'attente des processus.

Quand on examine de près l'algorithme, on remarque qu'une seule ressource peut être un goulet d'étranglement et un étranglement est associé à un site, pas à un type de processus. Plusieurs autres points sont à relever. Par exemple, l'algorithme n'autorise

pas la replication de fichiers si le nombre d'accès en écriture d'un fichier est plus grand que le nombre d'accès en lecture (multiplié par la taille relative du fichier) ceci afin de favoriser les écritures locales (une écriture coûte toujours plus cher qu'une lecture).

Certains points ne sont pas bien captés dans l'algorithme. Par exemple dans le cas d'un système non homogène (au sens de nos études), il n'est pas sûr que la replication sur un site qui serait moins performant provoque une amélioration. En fait l'algorithme considère que le système distribué est homogène. C'est une base de travail.

6. Nos perspectives en ordonnancement

6.1. Nouvelles propositions pour l'ordonnanceur d'XtremWeb. Si l'on cherche à adapter l'algorithme de Anna Hać [Hac89] et à l'implémenter concrètement sur une plate-forme `Linux` comme ordonnanceur nous avons les problèmes techniques suivants. Le système de fichiers `/proc` dans lequel `Linux` range à la volée des statistiques sur l'état de la machine ne fournit qu'une information globale sur le nombre de lectures et d'écritures faites sur les disques. Or pour l'algorithme il faut distinguer les opérations d'entrées sorties sur chaque fichier ouvert. Les informations disnonibles à l'heure actuelle dans `/proc` ne permettent pas de l'observer. Par contre on peut observer, de manière globale, les lectures, lectures asynchrones, écriture, écritures asynchrones mais cela a moins d'intérêt pour ce que l'on veut faire. Il y aurait donc un travail important de développement à assurer au niveau du noyau.

Que pouvons nous alors proposer pour XtremWeb ? L'ordonnanceur ne capte pas tout ce qui a attrait aux disques : aucune décision n'est prise si le trafic des entrées sorties est important. Nous proposons de le modifier dans le sens de l'algorithme de [Hac89] et en ajoutant des critères d'ordonnancement supplémentaires : si le trafic mémoire devient important (ou que la ressource mémoire principale devient « rare ») alors on peut essayer de lancer une version out-of-core du programme.

L'idée est de repousser dans le temps la migration et de continuer à calculer localement. Réciproquement, si le trafic avec les disques locaux à une machine est important et si l'algorithme utilisé est un algorithme out-of-core nous suggérons d'essayer de lancer une version de remplacement in-core. Il s'agit éventuellement d'autoriser l'utilisation de tampons en mémoire principale de plus grande capacité. La aussi il s'agit de retarder les migrations.

La validité de cette nouvelle approche repose grandement sur la précision et l'estimation des métriques de performance choisies.

6.2. Les langages passent des informations au système d'exploitation. Reprenons, pour terminer les discussions techniques, un morceau de code pour trier

en parallèle sur disque selon la méthode PSRS par exemple. Nous cherchons à illustrer que potentiellement, la connaissance du code peut amener certaines fonctionnalités du système d'exploitation au niveau de l'application afin de mieux contrôler les performances.

Prenons plus exactement la partie du code qui traite de la sélection des pivots dans l'implémentation du tri externe. Nous avons d'abord un appel à la procédure de tri disque en local, puis la sélection des pivots qui sont rangés dans un vecteur intermédiare et en mémoire, puis le rassemblement sur le processeur 0 des pivots sélectionnés localement. Toutes ces étapes, exceptéela première se déroule en mémoire RAM.

Le coût en temps de la gestion des pivots peut en pratique, dans de nombreux cas, être très inférieur au coût du tri séquentiel car il ne s'agit que de lire quelques dizaines de kilo-octets et de les envoyer à un processeur dédié.

Aussi, dans le cas où une demande de déconnexion interviendrait après le tri séquentiel, le programmeur pourrait vouloir spécifier que la duplication des données du vecteur des pivots n'est pas avantageuse parce que ces données sont en très petit nombre vis à vis des données concernées par le tri séquentiel et qu'on pourrait migrer le vecteur des pivots en cours de construction et reprendre l'exécution sur un autre site là ou elle s'est arrêtée.

Par contre, si une demande de déconnexion intervient au cours du tri disque séquentiel, alors comme les données sont en nombre plus important, on aurait plutôt intérêt à les dupliquer sur un autre site et reprendre le tri sur ce site par un RPC par exemple.

Si l'on accepte de spécifier également au niveau des instructions de synchronisation des garanties contractuelles d'accès au réseau, on peut alors préciser qu'au moment d'une déconnexion on accepte de migrer, sur des << sites lointains >>, les données peu nombreuses et qu'il semble préférable d'avoir des redondances de données pour le tri séquentiel dans un << voisinage réseau >> et ceci pour optimiser la gestion des duplications. L'étude générale et les compromis à faire entre duplication et migration de données pour le cas du tri nous paraissent un challenge intéressant à relever.

Ces formes de contraintes exprimées au niveau du langage de programmation, parce que le programmeur connaît bien les caractéristiques de son code, marquent elles l'inclusion de plus en plus prononcée du système d'exploitation au sein des applications? La construction d'un dispositif intégrant les discussions conduites dans ce paragraphe de perspectives et opérant sur le cas du tri comme étude de cas devrait à notre avis servir à montrer si le fait de guider le système d'exploitation au niveau de l'application a un avantage pratique pour le calcul distribué à grande échelle vis à vis de la situation ou le système d'exploitation découvre par lui même ce qui est préférable de faire à un instant donné.

Nous proposons également d'étudier et de construire un oracle permettant de << prédire le futur >> de plusieurs milliers de machines inter-connectées à Internet à partir des métriques de performances comme celles présentées dans les derniers paragraphes. Cet oracle prendra la décision du groupe de machines sur lesquelles l'application distribuée, qui entre dans le dispositif, sera exécutée.

Pour évaluer l'ensemble du dispositif, nous proposons de confronter les résultats d'expérimentation où de simulation obtenus en prenant le système d'une part en configuration ou l'oracle prend initialement une décision de placement des processus et des données et avec le dispositif de migration offrant le moins de fonctionnalités possibles (éventuellement sans migration évoluée) et d'autre part en prenant le système configuré avec le maximun de fonctionnalité. L'application visée serait le tri parallèle. Il n'est pas acquis qu'un dispositif très élaboré de migration apporte un avantage significatif par rapport à un dispositif reposant principalement sur un placement initial intelligent des processus et des données.

CHAPITRE 4

Structures de données adaptées aux problèmes avec des disques

1. Introduction

LA PROBLÉMATIQUE du stockage haute performance lorsque les demandes d'entrées/sorties se déroulent en parallèle est abordée maintenant. Il s'agit ici de la problématique de la *représentation des données sur les disques* que nous abordons plus précisément dans le cadre du projet CGP2P[1] de calcul global pair à pair dans lequel Amiens est impliqué. Nous développons également un ensemble d'idées plus générales qui permettent de fixer la problématique.

Les applications logicielles pour Internet comme les librairies digitales, les laboratoires virtuels, la vidéo à la demande, le commerce électronique, les services WEB (fouille de données), les systèmes collaboratifs demandent de maîtriser le stockage des données, les entrées sorties, l'équilibrage des charges des systèmes, la protection des données et des communications longues distantes. Il est bien connu que la puissance des processeurs double tous les 18 mois!. C'est la loi de Moore. Concernant les disques, bien que la densité du stockage ait augmenté de l'ordre de 60 à 80% par an, les temps d'accès aux données qui dépendent en grande partie de facteurs mécaniques, n'ont connu qu'un accroissement de l'ordre de 10% par an.

Les bases de données, séquentielles ou distribuées, font un usage important des disques. Dans les bases de données, une *structure d'indexation* est une structure de données où l'on accède à un ensemble de données via des clés. Il s'agit assez souvent d'un arbre. Il s'agit de construire des structures de données de stockage qui comprennent des *méthodes d'accès*, typiquement, recherche, lecture, modification.

Les méthodes d'accès doivent être évaluées dans un contexte particulier consistant en une instance (généralement un sous ensemble fini d'un domaine particulier) et un ensemble de requêtes. Un contexte et un ensemble de requêtes est appelé un *workload*. Par exemple, dans le cas des B-arbres qui est une structure pour le stockage d'informations à une seule dimension, une instance est un sous ensemble << ordonné >> et les requêtes considérées sont des requêtes du type << recherche d'intervalles >> . Nous pouvons aussi considérer [KRVV93] le cas général de l'indexation dans des modèles de données particuliers (données contraintes par exemple). Dans la référence [ASV99], les auteurs s'intéressent au cas à deux dimensions.

Le *workload* joue, en théorie de l'indexation, le rôle des langages partiellement récursifs en théorie de la complexité ou de la décidabilité. C'est l'unité dont la complexité doit être caractérisée. Par analogie, on peut considérer qu'un langage correspond à une famille de *workload*. Il s'agit de caractériser, pour chaque *workload* un << espace >> de

1. Voir `http://www.lri.fr/~fci/CGP2P.html`

schémas d'indexation possibles. La théorie de l'indexation a été formalisée par Hellerstein, Koutsoupias et Papadimitriou [HKP97]. Elle consiste à borner le nombre de blocs du (des) disque(s) contenant une réponse à une requête étant donnée une borne sur le nombre de blocs utilisés pour stocker l'instance considérée. Il est possible de jouer avec de la redondance d'information mais dans ce cas là, on s'intéresse au *facteur de redondance* qui est défini intuitivement comme le rapport entre le cardinal de l'union des sous-ensembles de l'instance utilisée sur le cardinal de l'instance utilisée. Cette métrique vaut 1 lorsqu'il n'y a pas de redondance. Il s'agit donc de trouver les bons compromis entre nombre d'accès et place disque utilisée.

Il s'agit pour nous d'étudier expérimentalement une famille de structures de données adaptées aux disques lorsque la structure est implémentée de manière distribuée avec Internet comme réseau d'interconnexion. Il s'agit donc pour nous de dire, parmi les schémas d'accés, quels sont ceux qui peuvent être utilisés dans la pratique et s'il faut en imaginer d'autres (par rapport à ceux connus qui souvent ont été majoritairement construits dans un cadre un processeur, un disque).

Ce chapitre est donc organisé de la façon suivante. Dans le paragraphe 2 nous rappelons comment on réalise du prallélisme au niveau matériel pour les disques. Dans le pragraphe 4 nous introduisons la notion de fouille de données. Dans le paragraphe 3 nous présentons les services de base à déployer dans une plateforme de calcul globale pour assurer un service de stockage. Dans le paragraphe 4 nous présentons un ensemble de structures de données connues adaptées aux disques et qui sont candidates pour s'intégrer dans la plateforme de clacul globale envisagée. Dans le paragraphe 5 nous exposons principalement la problématique de recherche d'une information dans un dispositif pair à pair (Gridella) de stockage : il s'agit de contrôler et de borner le nombre de messages échangés pour accéder à une donnée. Par ailleurs il faut construire des tables de routage locales car on s'interdit, par définition d'un dispositif pair à pair, de disposer dans les sites d'informations globales. Enfin, dans le paragraphe 6 nous précisons comment nous projetons d'intégrer les structures arborescentes présentées au paragraphe 4 dans un dispositif de stockage global ayant les propriétés exposées aux paragraphes 5 et 3.

2. Parallélisme au niveau des disques

Comme le gain obtenu en utilisant un ordinateur << plus performant >> est limité par le composant de l'ordinateur avec les plus petites performances, alors le système de disques est devenu un goulet d'étranglement. Plus que jamais il est important d'étudier des techniques pour améliorer les performances des accès disques.

La première technique consiste à multiplier les disques et à agencer les calculs de sorte qu'ils accèdent de manière concurrente aux données. De manière idéale cette solution permet potentiellement de stocker de grande masse de données tout en minimisant

les pertes de performance. C'est ce principe que nous avons retenu dans le cas du tri disque en milieu hétérogène sur clusters.

La techniqque du RAID (Redundant Arrays of Inexpensive Disks) est bien connue pour palier aux problèmes de performances. Un RAID consiste simplement en un système de plusieurs disques connectés à un seul controleur offrant un unique espace d'adressage. Cette configuration améliore la bande passante des opérations de lecture et d'écriture en masquant les latences dûes aux inerties des disques. Rappelons au passage qu'une carte PC avec une interface matérielle RAID 0-1 coûte aujourd'hui environ 200 €, par exemple avec la carte ABIT KG7-RAID.

En fait, différents niveaux de RAID existent (voir la Figure 1 pour les principaux niveaux de RAID). Ils permettent d'offrir aux utilisateurs, en fonction du niveau (de 0 à 5), de la performance couplée à une sécurité en cas de panne.

Le RAID-0 matériel apporte uniquement une performance en terme de débit sans sécurisation de données. En effet, le RAID-0 permet au contrôleur de sous systèmes disques d'éclater les informations (voir la Figure 1) et les éparpiller sur les différentes unités de disques mises à sa disposition. Sur la Figure 1, on peut imaginer que le fichier A est stocké sur le premier disque, le fichier B sur le deuxième disque etc.

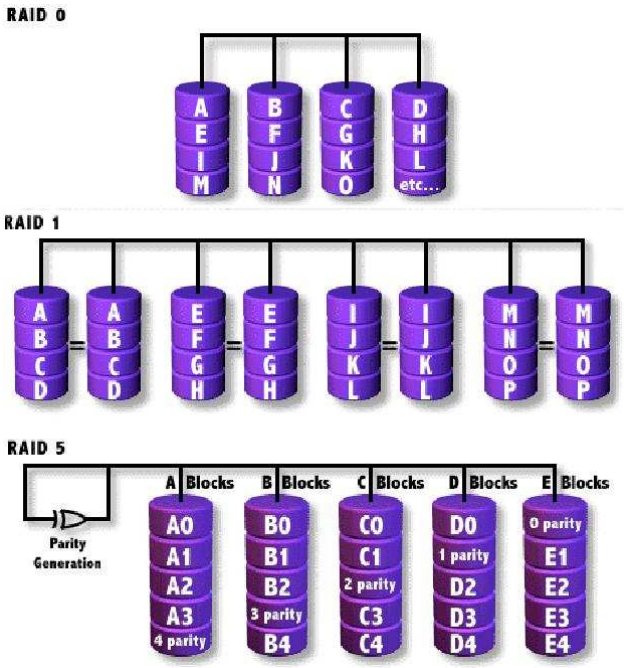

FIG. 1: *Quelques RAIDs au niveau matériel*

de manière cyclique. On peut donc accéder en parallèle aux fichiers A, B, C, D par exemple.

Le RAID-1 est le premier mode avec redondance. Le RAID-1 s'employe à partir de deux disques auxquels viennent éventuellement se greffer des disques de secours. Ce mode duplique les informations d'un disque sur l(es) autre(s). Jusqu'à $N - 1$ (sur la Figure 1 on a $N = 2$) disques otés (ou défectueux), les données restent intactes et si le contrôleur (SCSI, IDE, etc...) survit, la reconstruction sera immédiatement entamée sur un des disques de secours après détection de l'anomalie. Les performances en écriture sont légèrement inférieures à celles d'un disque unique vu que les données doivent être écrites sur chaque disque. Les performances en lecture sont en général bien plus mauvaises qu'en écriture en raison de la mise en oeuvre d'une stratégie d'équilibrage.

Le RAID-1 matériel ou 'mirroring' (voir la Figure 1) permet donc de dupliquer les données sur deux unités disques. Il est le moyen de sécurisation maximale mais reste très cher car il consomme 50% de l'espace disque total pour la sécurisation des données.

Le RAID-5 est sûrement le mode le plus approprié quand on souhaite combiner un grand nombre de disques tout en conservant de la redondance. Le RAID-5 s'employe à partir de trois disques avec éventuellement des disques de secours. À la différence du RAID-4, l'information de parité est répartie équitablement entre les différents disques, évitant ainsi le goulot d'étranglement du RAID-4 quand il s'agit d'une écriture. Rappelons que l'information de parité autorise la reprise après la panne d'au plus un disque. Autrement dit, si un des disques tombe en panne les données sont reconstruites. La reconstruction peut commencer immédiatement si des disques de secours sont disponibles. Si deux disques rendent simultanément l'âme, toutes les données sont perdues. Le RAID-5 ne survit pas à la défaillance de plus d'un disque. Les performances en lecture/écriture s'améliorent mais il est difficile de prévoir de combien.

L'architecture RAID-5 matérielle (voir la Figure 1) utilise la complémentarité du bit de parité qui est éclaté sur l'ensemble des disques. Ainsi, en cas de défaillance de l'un des disques, la parité sera utilisée pour reconstruire le disque manquant. Le RAID-5 offre donc une sécurisation supplémentaire par rapport aux niveaux 0 et 1.

La notion de RAID est donc très attractive. Après avoir brièvement présenté les premiers éléments de réflexion sur les services de base que l'on envisage d'intégrer à la plate-forme XtremWeb, nous considérerons les structures de données adaptées au stockage sur disques et qui potentiellement peuvent apporter des performances.

On peut remarquer que le stockage de fichiers depuis un PC connecté à Internet, c'est à dire depuis sa maison, pourrait être un service accessible rapidement au grand public. En effet, plusieurs annonces concernant le stockage via Internet ont été faites

en 2001, dont les solutions de la société `Xdrive` (voir `http://www.xdrive.com`) qui propose du stockage à 4.95 dollars par mois pour stocker jusqu'à 75M octets.

On comprend aisément que les enjeux financiers liés à cette proposition sont énormes puisque Internet se généralise dans les foyers et qu'un service << de masse >> produit des retombées financières importantes, en général.

3. Mise en œuvre des services de base

De manière générale et en particulier pour le projet CGP2P, les objectifs et les garanties demandés au dispositif à construire sont alors les suivants :

(1) *partager les espaces de stockage* ;

(2) *fiabilité* dans le sens que l'information est pérenne (par exemple, les fichiers n'ont pas une date d'expiration comme cela est implémenté dans certains dispositifs d'échange de fichiers audio) et aussi dans le sens << disponibilité >> ;

(3) *performance* : les requêtes pour une recherche, une mise à jour de fichiers doivent se faire le plus vite possible. Les applications visées sont par exemple des applications de calcul numérique out-of-core pour lesquelles nous avons une expérience ; Contrairement à Gnutella (une requête est envoyée à un ensemble de machines qui regardent localement si elles ont l'information), nous pensons qu'il faut introduire un mécanisme d'indexe pour aiguiller de manière plus fine les recherches ;

(4) *discret* : le dispositif ne doit pas perturber le fonctionnement souhaité par l'utilisateur. Autrement dit, l'utilisation des ressources des machines des particuliers sera << raisonnable >> ;

(5) *garder le contrôle* : les données seront cryptées mais on devra pouvoir localiser à tout moment les données (dans certains systèmes pair à pair on n'a pas besoin de cette garantie) ;

(6) *traiter les informations* : s'agit-il de stocker l'information pour des traitements ultérieurs qui auront lieu à l'endroit où se trouvent les données ou s'agit-il uniquement d'avoir un stockage, le traitement, l'utilisation se fera depuis un nœud distant ?

La démarche poursuivie sera une approche système avec un découplage entre le stockage (il s'agira de construire un fournisseur de blocs), la gestion des (méta)données et notamment leurs représentations et enfin le traitement des données.

Pour la partie stockage on réutilisera des parties de systèmes de fichiers existants comme NFS, GFS, xFS, DAFS, Coda.... Cependant on distinguera :

Les machines qui fournissent des blocs: elles devront assurer des propriétés de tolérance aux pannes, d'équilibrage des charges, de disponibilité, de certification et d'authentification ;

Les machines gestionnaires globaux des données: elles devront assurer les services de nommage, de cohérence et d'équité.

Concernant le traitement des données, on envisagera de distribuer les calculs là où se trouvent les données par l'intermédiaire d'un modèle de calcul du type des Remote Procedure Call (RPC) ou du type « parallélisme de données » : on produit les données au moment où on en a besoin. Cette question n'est pas tranchée.

Examinons maintenant quelles sont les représentations des données que l'on pourrait utiliser.

4. Structures de données arborescentes connues

Pour cerner quelques difficultés, nous proposons ici un état des lieux des structures arborescentes pour des problèmes « à la volée » (on-line) principalement. Nous entendons par « à la volée » des problèmes où la structure de données est modifiée au cours du temps. Il peut s'agir par exemple d'un dictionnaire, d'une base de données où les opérations de base comme l'insertion, la suppression, la recherche arrivent à n'importe quel moment. La structure de données n'est plus figée ; elle est ici dynamique. Nous n'envisageons pas dans l'immédiat de nous intéresser au stockage des structures de données statiques du type des matrices.

Tout comme en algorithmique pour les disques, les critères de performance sont une combinaison du temps d'exécution, du nombre d'opérations d'entrées sorties (nécessaires aux opérations de base) et de l'espace de stockage nécessaire. Les défis sont :

(1) de répondre aux requêtes de recherche en $\mathcal{O}(\log_B N + z)$ I/Os ;
(2) d'utiliser un espace de stockage linéaire ;
(3) de faire les mises à jour (dans le cadre « à la volée ») en $\mathcal{O}(\log_B N)$ I/Os ;

avec les notations suivantes : N : la taille du problème en entrée (items), Z : taille du problème en sortie, M : taille de la mémoire principale, B la taille d'un bloc, D : le nombre de disques indépendants. De plus on demande :

$$M < N, \quad 1 \leq D.B \leq M$$

$$n = \frac{N}{B}, \quad z = \frac{Z}{B}, \quad m = \frac{M}{B}$$

Si l'on n'y prend pas garde, on peut facilement construire des structures de données complètement inadaptées au calcul avec les disques. Par exemple, supposons que pour un problème, sa solution "in-core" passe par un arbre de recherche binaire. Le temps CPU d'une requête est en $\mathcal{O}(\log_2 N + Z)\ldots$ mais comme il faut une opération d'I/O pour passer d'un nœud à un autre, on a aussi $\Omega(\log_2 N + Z)$ I/Os ce qui est excessif ! De plus c'est le gestionnaire de mémoire virtuelle qui travaille et nous perdons du contrôle.

La réalisation de structures de données adaptéees aux problèmes nécessitant les disques vise donc à construire de la localité directement dans la structure de donnée et à gérer explicitement les opérations d'I/Os au lieu de laisser faire le système d'exploitation.

Pour notre exemple précédent d'arbre binaire on vise un gain relatif de $(\log N + Z)/(\log_B N + z)$ qui est au moins de $(\log N)/(\log_B N) = \log B$ ce qui est significatif en pratique.

Certaines structures de données pour les disques sont connues depuis 30 ans ; d'autres inventées récemment, pour un besoin particulier. Nous pouvons citer un certain nombre de structures << briques de base >> pour les problèmes suivants :

Recherche 1-D: B-trees (qui sont des arbres avec un nombre de fils compris entre $B/2$ et $B - 1$ fils) ;
Batch (dynamique): Buffer trees ;
Multi dimensionnel: R-trees ;
Problèmes de reconstruction: Weight balanced B-trees ;

Un certain nombre de problèmes On-line sont bien connus et bien traités pour les disques, parmi lesquels (nous laissons certains termes anglo-saxon - noter de plus qu'il s'agit de problèmes en géométrie algorithmique) :

(1) Proximity queries, nearest neighbor, clustering ;
(2) Point Location ;
(3) Ray shooting ;
(4) Calculs d'intervalles (External inverval trees) ;

4.1. B-tree et variantes. C'est une structure connue depuis 30 ans [BM72]. Ses propriétés sont les suivantes et un exemple est donné à la Figure 2 (en fait il s'agit d'un B^+-tree - voir plus loin) :

– Chaque nœud est rangé dans un bloc disque ;

- $\frac{B}{2} \leq$ degré nœud $\leq B - 1$ (sauf pour la racine) ;
- Les nœuds internes stockent les clés et les pointeurs pour guider la recherche.
- Les opérations d'insertion, suppression, suppression du plus petit se réalisent en $\mathcal{O}(\log_B N)$ operations d'entrées sorties ;
- La suppression ou l'insertion sont des opérations qui présentent un certain degré de technicité lorsqu'un nœud devient vide ou plein. Mais on sait le traiter comme cela est rappelé partiellement à la Figure 3.

4.1.1. *Quelques structures proches du B-tree.* Il s'agit en particulier :

B^+-tree : tous les items sont stockés dans les feuilles ; les nœuds internes stockent les clés et les pointeurs (pour une même taille on peut stocker << plus >>). La conséquence est que la hauteur de l'arbre est plus petite que pour un B-arbre et donc la recherche sera plus rapide ; les feuilles sont chainées pour faciliter les "range queries" et/ou la recherche séquentielle ;

B^*-tree : quand un nœud est plein, il s'agit de retarder le ré-équilibrage : on redistribue quand on a deux nœuds pleins dans trois nœuds occupés au 2/3 (pour le B-arbre, on redistribue quand un seul nœud est plein) ; Il s'agit de Réduire la fréquence de création de nouveaux nœuds ce qui est coûteux dans la pratique ;

Prefix B-tree : considérons un B^+-tree avec un indexe de type chaînes de caractères : on garde les plus petits préfixes comme séparateurs (gain en place) et on a un facteur de branchement augmenté ;

Ce qui est impraticable : B-arbres binaires ($B = 2$) et arbres 2-3 qui ne sont praticables qu'en mémoire principale.

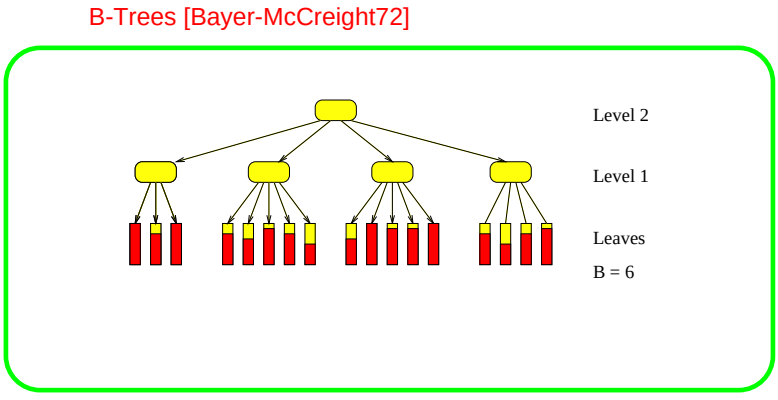

FIG. 2: *Représentation d'un B-arbre*

4.2. Buffer-tree et R-tree. Ce sont des structures de données utilisées en géométrie algorithmique et inventées en 1995 par Arge [Arg95] et en 1984 par Guttman [Gut84] respectivement. Un exemple de R-tree est donné à la Figure 4 et un exemple de Buffer-tree est donné à la Figure 5. Les propriétés importantes des R-trees sont les suivantes :

Utilité : recherche dans des espaces dimensions supérieure à 1 ; ils permettent de garder un espace de stockage linéaire ; l'idée est de coder l'information sur les << boites >> contenant un certain nombre de rectangles. Par exemple, sur la Figure 4 le nœud racine stocke la boite R_1 de dimension définie par les points supérieur gauche et inférieur droit et R_1 contient ainsi les rectangles A, B, C, D ;

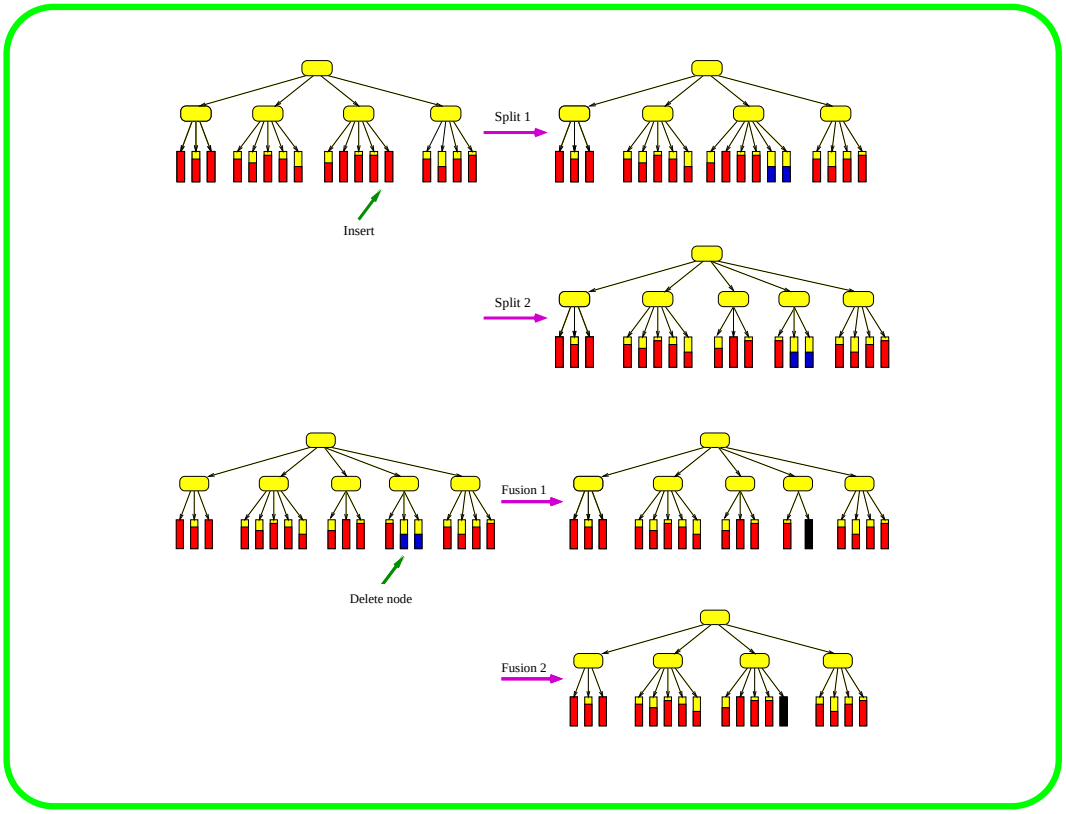

FIG. 3: *Opérations de ré-équilibrage dans un B-arbre*

Degré nœuds internes : $\Theta(B)$ et les feuilles stockent $\Theta(B)$ items. À chaque nœud est associé a une boite dite *bounding box* de tous les éléments dans les sous arbres ; Ainsi, sur la Figure 4, la boite R2 (donnée par exemple par ses coordonnées supérieure gauche et inférieure droite) contient tous les points qui sont à l'intérieur de cette boite. En considérant un nøeud de l'arbre, on est donc capable de dire en temps constant $\Theta(B)$, si un point est « dans le sous arbre considéré » ;

En pratique : ils ont de bons comportements pour des « petites dimensions »

Le Buffer-tree (voir la Figure 5) a été conçu pour remédier à la question suivante : « Que peut-on faire si les réponses aux requêtes peuvent arriver dans n'importe quel ordre ? ». L'idée principale est la suivante : on regroupe logiquement des nœuds et on ajoute des buffers. Ainsi, quand un buffer devient plein, les items descendent d'un niveau. Les propriétés essentielles d'un Buffer-tree sont les suivantes :

(1) C'est un arbre équilibré de degré $\Theta(m)$ avec un tampon de taille M pour chaque nœud ;
(2) Les items dans un nœud descendent d'un niveau quand le buffer est plein et non pas quand un nœud devient plein ;
(3) Vider un tampon requiert $\mathcal{O}(m)$ I/O's ce qui amorti le coût de distribution de M items vers les $\Theta(m)$ fils ;
(4) Pour chaque item, il y a un coût amorti de $\mathcal{O}(m/M) = \mathcal{O}(1/B)$ I/O par niveau. La recherche et la mise à jour se font en $\mathcal{O}((1/B)\log_m n)$ I/O.

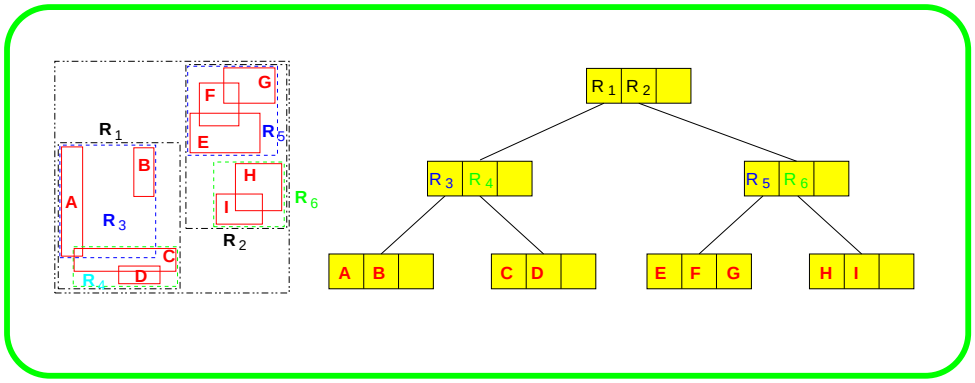

FIG. 4: *Représentation d'un R-arbre*

4.3. Structures adaptées à la recherche de chaînes de caractères. Les chaînes de caractères sont la structure de données de prédilection en bio-informatique. Par exemple, la struture de l'ADN peut se coder sur un alphabet de quatre lettres. L'algorithmique séquentielle de recherche de motifs en mémoire principale [ACGM92, BYP92] et dans une moindre mesure en parallèle (également en mémoire principale) a été largement étudiée dans le passé pour de nombreux problèmes [CH99] comme l'alignement de séquences.

Des structures de données arborescentes comme les arbres Patricia, les arbres digitaux (voir [pat91] pour une synthèse) ont été proposées il y a maintenant 30 ans. Ces structures ont été conçues dans l'idée de les implémenter en mémoire principale. Dans la littérature, il y a très peu de structures de données, à notre connaissance, pour la recherche de chaînes en mémoire externe. Nous pouvons citer principalement deux études : les String B-Tree de Ferragina et Grossi [FG99] et les Suffix Trees [CM96] de Clark et Munro.

4.3.1. *La structure String B-tree.* Il s'agit d'une structure de données adaptée au stockage de chaînes de caractères arbitrairement longues, qui, contrairement aux structures connues (arbres Patricia, arbres suffixes) ne tiennent pas en mémoire principale. Elle doit intéresser en principe les chercheurs en biologie, cristallographie etc

Pour obtenir des bonnes complexités pour la recherche ou la mise à jour par exemple et dans le pire cas, la structure de données se présente comme une combinaison des B-tree et des arbres Patricia auquel on adjoint des pointeurs. En fait, il s'agit de la première structure de données gérant des chaînes arbitrairement grandes qui offre les mêmes complexités que les B-tree en ce qui concerne les opérations élémentaires (recherche, mise à jour).

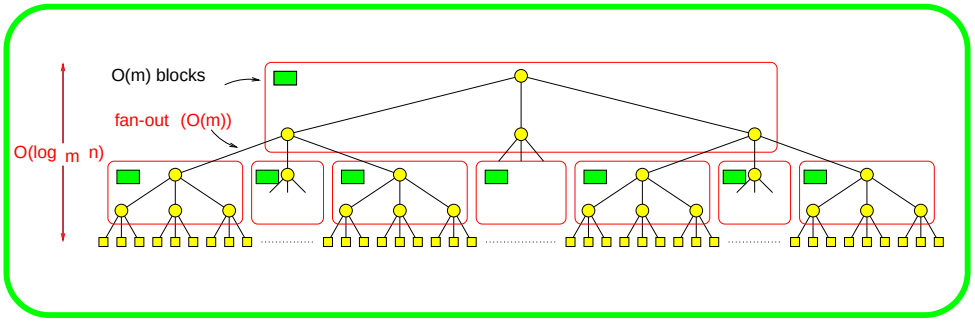

FIG. 5: *Représentation d'un Buffer-tree*

Mais notre impression générale est que les biologistes, les crystallographes pensent que l'utilisation de $n > 1$ processeurs permet d'accroître d'une part la vitesse de traitement et d'autre part la taille de la mémoire centrale disponible. Nous ne sommes pas certains que l'idée de stocker de l'information sur plus de disques et de traiter par la sorte des problèmes de taille très supérieures à ce qui est fait aujourd'hui soit bien ancrée. L'idée de pouvoir concevoir des strutures de données[2] et des algorithmes efficaces adaptés à des problèmes de grande taille n'est pas encore passée, à notre avis. C'est sans doute à nous de démontrer la pertinence de l'approche.

4.4. Structures de données non équilibrées. Toutes les structures arborescentes vues juqu'à présent sont équilibrées dans le sens où la hauteur des sous arbres issus de la racine diffèrent d'au plus une unité. Nous allons retrouver cette caractéristiques également dans les structures arborescentes envisagées à ce jour pour faire du stockage de données à travers Internet. Nous allons insister dans le paragraphe qui suit sur le potentiel que peut avoir une structure arborescente non-équilibrée.

5. Structures arborescentes et calcul pair à pair

5.1. Introduction. Commençons par rappeler quelques arguments complémentaires à ceux déjà donnés (à savoir les coûts logarithmiques des opérations de base) en faveur des structures arborescentes. Si l'on choisit de mettre les disques au niveau des feuilles de l'arbre et que les nœuds internes de l'arbre ne comportent que les indexes (que l'on pourrait à la limite stocker dans les routeurs de communication) et si l'on admet que les noeuds (les routeurs) ne peuvent pas tomber en panne, alors, en algorithmique distribuée [Awe87], on sait construire des algorithmes tolérants aux pannes et ou auto-stabilisants pour élire une racine [Dol00], par exemple. Ainsi, on pense qu'à priori les structures arborescentes peuvent amener des propriétés nouvelles, importantes lorsque le réseau de connexion est Internet.

5.2. Gridella. Dans ce paragraphe nous présentons dans un premier temps l'approche Gridella [KAS02] pour gérer l'accès aux données dans un dispositif pair à pair, puis, dans le prochain paragraphe nous faisons une critique de cette approche afin de proposer une alternative pour le projet XtremWeb, notamment en terme de structure de données facilitant la recherche, la mise à jour...

Le nom Gridella[3] vient de Gnutella : il s'agit d'un dispositif sans base de données centralisée et requiert donc un mécanisme élaboré de recherche. Gridella est décrit principalement dans [KAS02] et une version Java sera disponible au printemps 2002 affirment les concepteurs. Il s'agit d'une plate-forme de calcul global qui fait intervenir

2. Le stockage en génomique par exemple se fait << à plat >> si bien qu'à chaque requête, c'est tout le fichier qu'il faut parcourir ; Source : tutoriel de Srinivas (Iowa State university) à HiPC'2000

3. Voir `http://www.p-grid.org`

une structure arborescente pour localiser les données : il s'agit en fait de gérer un indexe pour optimiser la recherche. De plus le contrôle est distribué ce qui en fait une plate-forme unique en son genre.

Gridella est construit à partir de P-Grid [Abe01] que l'on peut voir comme un dispositif qui assure la distribution des données (éventuellement repliquées) dans une communauté de machines de stockage. Conceptuellement, P-Grid[4] peut être vu comme un arbre binaire de recherche de telle sorte que la recherche ou le nombre de messages générés pour une recherche croit de façon logarithmique avec le nombre de données stockées dans le dispositif.

Le contrôle des opérations de recherche, mise à jour... dans Gridella est fait de manière décentralisée et l'environnement est considéré comme non fiable. Pour cela, Gridella effectue de la redondance d'informations selon un principe qui est un peu long pour être expliqué dans tous les détails. Il semblerait, bien que l'on ne puisse pas l'assurer à 100%, que la gestion de la duplication repose sur un algorithme de Plaxton, Rajaraman, Richa [PRR99]. Il s'agit d'un algorithme qui vise à satisfaire une requête d'accès à un objet distribué à partir de la copie la plus proche du site qui fait la demande. Dans l'article, il y a une analyse du schéma d'accès pour différentes fonctions de coût qui capturent la nature hiérarchique des réseaux dits << à grande échelle >>. L'article est aussi centré sur les problèmes de lecture uniquement et pas sur les opérations d'écriture qui nécessitent la gestion de la cohérence des copies. De même la question du choix pratique du nombre de copies n'est pas traité.

Le cœur du schéma d'accès, c'est-à-dire l'ensemble des algorithmes pour effectuer une lecture, une insertion et une suppression, repose sur un mécanisme élégant et simple dans le principe qui maintient et trouve une copie d'objet. Il s'agit de plonger le réseaux physique dans un arbre << virtuel >> équilibré en hauteur[5]. Ainsi, quand un nœud cherche à faire un accès à un objet, il commence par regarder s'il est disponible localement, puis dans ses sous-arbres, sinon il fait une requête à son père. Une propriété importante du schéma d'accès fait que lorsqu'une copie migre on n'a pas besoin de renommer la copie : c'est donc un schéma indépendant du nommage des copies.

Après cette introduction, revenons à la description de la Figure 6. Chaque machine participant au dispositif est repérée par un chemin dans l'arbre binaire. Par exemple, sur la Figure 6 la machine de numéro 3 est responsable pour stocker les données dont la clef débute par $10_{base\ 2}$ (4 en décimal) mais on peut aussi avoir la machine de numéro 3, qui appartient au même groupe et qui aura aussi la charge de stocker les données des clefs débutant par 10. Un groupe peut contenir plusieurs machines. C'est ainsi que l'on réalise la duplication et que l'on assure une certaine robustesse vis à vis des problèmes de déconnexion.

4. Voir aussi l'article introductif en ligne sur `http://lsirwww.epfl.ch/lsirpubl.htm`

5. En fait il y a autant d'arbre virtuels qu'il y a d'objets

L'approche du routage nécessaire à une recherche par exemple est simple : pour chaque niveau dans l'arbre, chaque machine stocke l'adresse d'au moins une machine qui est responsable du stockage pour l'autre coté de l'arbre binaire (voir la Figure 6 et les carrés en grisé). Par exemple sur la Figure 6, pour les clés commençant par 1, c'est la machine 3 qu'il faut consulter ; pour les clés débutant par 01_2 c'est la machine 2.

La construction des tables de routage locales à chaque machine assure qu'il y a pour toutes les machines au moins un chemin (une liste de machines) permettant d'accéder à la machine stockant les données. On remarquera sur la Figure 6 qu'il n'y a pas de chemin direct entre la machine 3 et la machine 1. Cependant, il y a un chemin de la machine 3 à la machine 6 et 6 appartient au même groupe que 1 (6 contient une copie des informations de 1).

Ainsi, une requête est envoyée à une machine tirée au hasard. Dans la pratique, à un prefixe correspond une liste de machines (et non pas une seule) et la machine recevant la requête tire au hasard celle qui poursuivra la recherche.

Passons maintenant à la description de la construction des tables de routage. Comme il n'y a pas de contrôle global, la construction se fait par des interactions locales. La première machine qui entre dans le système a la charge de contrôler l'ensemble de l'espace de recherche[6]. Quand une deuxième machine arrive dans le système global et cherche à rentrer en contact avec la première machine, l'espace de recherche est divisé en deux. Et ainsi de suite.

6. Notons qu'ici il y a besoin d'une connaissance globale de l'état du système

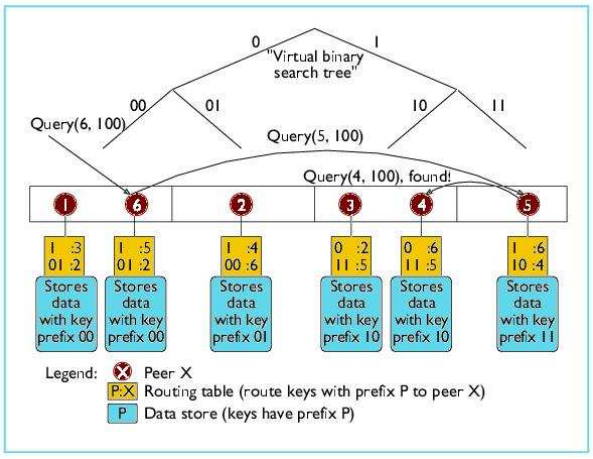

FIG. 6: *Structure de données dans le projet Gridella. La Figure provient de [KAS02]*

Les auteurs [KAS02] avancent que la simulation assure :

(1) que la vitesse de convergence vers un arbre complet est indépendant du nombre total de machines (peers) ; chaque machine participe à un nombre constant d'échanges indépendant de la taille de l'entrée du problème ;
(2) que le passage à l'échelle se passe bien (l'augmentation de la longueur maximale du chemin de recherche n'introduit pas de perturbation) ;
(3) que le nombre de machines responsables pour la même clé est uniformément distribué par rapport au nombre moyen attendu de machines responsables de cette clé.

Comme argument supplémentaire, les auteurs [KAS02] donnent les performances suivantes : pour une recherche et pour un système de 200000 machines, il y a seulement 72 messages échangés avec Gridella alors qu'il en faut 78728 avec Gnutella (chaque machine a une probabilité de 30% d'être en ligne).

Les résultats sont impressionnants mais on ne voit pas très bien l'impact de la duplication (et d'ailleurs sous quels critères on l'applique) dans les performances. C'est un point à éclaircir. Il est juste mentionné dans [Abe01] que les sites sont dupliqués environ 20 fois pour que cela fonctionne bien.

De même on ne voit pas comment la structure proposée gère les "range query", par exemple on recherche toutes les données avec un prefixe 1011 et un suffixe 0100. Que faut-il modifier dans la structure de donnée pour bien gérer ce type de recherche ?

5.3. Plan d'expérimentation. Le prototype et les expérimentations que nous viserons à construire dans le cadre du projet CGP2P, est alors le suivant : $n > 1$ machines participent au stockage d'une structure de données à la volée comme un dictionnaire ou une base de données. Chaque machine ne met pas à disposition forcément la même quantité d'information que les autres. Ces machines sont chez des particuliers et peuvent décider à tout moment de participer ou de ne plus participer au stockage. Les canaux et les nœuds de communication (routeurs) sont fiables et non défaillants.

Globalement, on fait le choix à priori de B-arbre comme structure de données pour assurer les mêmes services que dans [PRR99]. En effet, la structure de données employée dans [PRR99] est une structure en mémoire centrale. Certes un théorème de [PRR99] affirme que si q est le nombre d'objets qui peuvent être stockés dans la mémoire de chaque nœud alors le surplus en mémoire qui est nécessaire à l'algorithme est $\mathcal{O}(q \log^2 n)$ mots où un mot est codé sur $\log n$ bits, ce qui est acceptable dans la pratique. Mais ce qui limite l'usage pratique c'est bel est bien le facteur q. En stockant sur disque les tables, on peut espérer réduire d'un ordre de grandeur le nombre de nœuds à visiter pour une recherche (au prix d'opérations d'entrées sorties).

Ensuite, il s'agit d'étudier une famille de structures arborescentes sur disques plutôt que du seul B-arbre. En effet, si l'on sait par exemple que la copie recherchée est du type « point dans un espace 2D », alors les Buffer-trees ou les R-trees sont plus intéressants. Il conviendrait alors de fournir une *famille de protocoles* de recherche, spécialisés selon le type concret des objets.

De plus, on distinguera les nœuds qui assurent un service de routage grâce à des clés de recherche et des tables de routage et que l'on considérera fiables et sans déconnexion et les nœuds de stockage qui peuvent se déconnecter. L'idée est de compter sur un groupe de machines fiables pour déterminer qu'elles est (sont) les machines qui contiennent l'information. Cette approche est donc distincte de Gridella qui ne considère qu'une seule classe de machines (peers).

Il nous semble qu'un avantage de notre approche qui distingue des nœuds fiables et des nœuds qui peuvent se déconnecter permettra de capter avec des difficultés moindre les problèmes de déconnexion non négociée. Pour l'instant, il a été sous entendu que lorsqu'un site souhaitait quitter le système global, il le faisait de manière négociée en envoyant un message dédié. À priori, dans notre cas, comme il est sous-entendu que les données sont dupliquées et que la détermination des machines sur lesquelles se trouvent les données est fiable, on peut envisager de capter et de traiter par des algorithmes « raisonnables » (dans le sens où les coûts d'implémentation et de fonctionnement sont faisables pratiquement) les problèmes de pannes non négociées des machines possédant les informations.

Par contre, la construction de l'arbre (les nœuds internes) devra se faire de manière non coordonnée de manière globale en ne considérant que des interactions locales, comme ce qui est fait dans Gridella.

Les difficultés sont alors les suivantes :

(1) l'arbre sera tel que le nombre de passerelles physiques entre un « nœud père et un nœud fils » sera maintenu le plus bas possible (voir [PRR99]). Eventuellement, on ne recherchera pas forcément à construire des arbres équilibrés mais des arbres qui vont minimiser le nombre de nœuds de communication à traverser pour une recherche.

(2) implémenter les opérations d'ajout, de suppression, de recherche ; garantir les coûts en terme de messages échangés, de données dupliquées et d'opérations d'entrées sorties ;

(3) implémenter les opérations de rééquilibrage de l'arbre lorsqu'un seuil de remplissage aura été atteint dans une feuille.

(4) implémenter un mécanisme de tolérance aux déconnexions des machines de stockage en « jouant sur la duplication » des données ;

(5) autoriser les "range queries" ;

(6) évaluation du dispositif : il s'agira de confronter cette approche avec celle qui consisterait à utiliser uniquement les services de base (voir le paragraphe 3) et qui gèrent aussi la redondance, les défaillances[7]. Autrement dit, il s'agirait de confronter une approche système où la sécurisation, la redondance est assurée au niveau matérielle avec une approche où l'on gère explicitement le placement des données de type T dans le protocole de stockage des objets de type T.

Notons que l'approche RAID-5 a un coût de reconstruction des données lorsqu'une machine de stockage se déconnecte alors qu'avec l'approche << à la Gridella >> il n'y a pas ce coût. Par contre, avec une approche << à la Gridella >> la duplication entraîne, dans le cas d'une écriture, des mises à jour sur les copies (que l'on peut gérer sans doute par des algorithmes de cohérence) alors qu'avec l'approche RAID-5 nous n'avons pas de problème de cohérence à gérer.

6. Travail connexe et conclusion

Le projet de recherche qui approche le plus l'étude que nous proposons de mener est le projet GiST[8]. Il s'agit d'un projet de Berkeley qui vise l'étude mathématique et d'ingénierie à accomplir dans les schémas d'indexation de grande masse d'information. Le champ disciplinaire étudié est plus précisement << les bases de données classiques >> dans le sens ou les implémentations proposées sont intégrées dans la base `PostgreSQL`. Il ne s'agit pas d'études réalisées avec en tête les systèmes de calcul global ou pair à pair. Ainsi, certaines fonctionnalités et problèmes que nous avons mentionnés ne sont pas captés : duplication des données et leurs accès et mise à jour, placement des copies. Cependant les structures de données visées sont les arbres sur disques (B-arbres et variantes).

Le projet `GiST` considère une structure de recherche généralisée (Generalized Search Tree (GiST)), qui est un patron qui autorise des experts de différents domaines (chercheurs en géométrie algorithmique, en bio-informatique, en base de données...) de modifier les méthodes d'indexation d'une base de données selon le type concret des éléments qui sont des variations autour des B-arbres (R-trees, B+-trees etc).

Le projet `GiST` unifie dans une même bibliothèque d'accès la possibilité de traiter différents types d'arbres et différentes formes de requêtes. Par exemple, `POSTGRES` supporte les B+-arbres et les R-arbres ce qui signifie que vous pouvez utiliser `POSTGRES` pour construire un B+-arbre ou un R-arbre dont le type concret des éléments est quelconque.

7. On semblerait s'orienter, pour les services de base, vers un dispositif basé sur la redondance << à la RAID-5 >>

8. Voir le lien `http://gist.cs.berkeley.edu/`

Mais les B+-arbres ne supportent que les requêtes avec un nombre limité de connecteurs ($<, =, >$), et les R-arbres ne supportent que des recherches du type : Contient, Est Contenu, Est Egal. Ainsi, si l'on cherche à indexer un lot de films avec un B+-arbre de `POSTGRES`, alors on ne peut poser que des questions du type « trouver tous les films $<$ T2 » avec $<$ pouvant signifier « moins chers, moins violent... ». Par contre avec un tel schéma d'indexation il n'est pas possible de poser des questions du type « trouver tous les films avec un course de moto » ou encore « trouver tous les films avec Bruce Lee ».

La bibliothèque `GiST` autorise de telles indexations et cela constitue un apport certain. La structure de données, comme pour un B-arbre contient des paires $<$ clé,pointeur $>$ mais les clés ne sont pas des entiers comme pour les B-arbres mais des fonctions membres d'une classe définie par l'utilisateur. Ces clés représentent une certaine propriété qui est vraie pour toutes les données pointées par le pointeur associé à la clé.

Par exemple, les clés dans un B+-arbre façon `GiST` sont des plages d'entiers signifiant par exemple que « tous les objets pointés sont des entiers compris entre 4 et 6 » ; les clés dans un R-arbre façon `GiST` sont des "boundingBox" pour signifier par exemple que « toutes les données pointées sont localisées en Californie ». Le programmeur a alors à décrire quatre méthodes pour la classe des clés considérées afin d'aider à réaliser les insertions, suppressions et recherches.

Les sémantiques des quatre méthodes de base à implémenter par l'expert sont laissées en anglais pour éviter les contre sens :

Consistent: This method lets the tree search correctly. Given a key p on a tree page, and user query q, the Consistent method should return NO if it is certain that both p and q cannot be true for a given data item. Otherwise it should return MAYBE.

Union: This method consolidates information in the tree. Given a set S of entries, this method returns a new key p which is true for all the data items below S. A simple way to implement Union is to return a predicate equivalent to the disjunction of the keys in S, i.e. "p1 or p2 or p3 or...".

Penalty: Given a choice of inserting a new data item in a subtree rooted by entry $<$ p, ptr $>$, return a number representing how bad it would be to do that. Items will get inserted down the path of least Penalty in the tree.

PickSplit: As in a B-tree, pages in a GiST occasionally need to be split upon insertion of a new data item. This routine is responsible for deciding which items go to the new page, and which ones stay on the old page.

Il nous semble donc particulièrement pertinent de capitaliser sur les résultats obtenus dans le projet `GIST`. La difficulté majeure reste à notre avis la gestion de la duplication ou de la redondance des données. Est-ce que l'on doit masquer sa gestion au niveau de

l'interface des méthodes? Peut-on réaliser des implémentations des quatre méthodes de base de sorte qu'il n'y ait que des interactions deux à deux sans passer par un mécanisme centralisé?

CHAPITRE 5

Analyse d'événements

1. Introduction

Dans ce chapitre nous examinons la problématique de la collecte et de l'analyse des événements donnant l'état de machines participant à un calcul global. La problématique est nécéssairement une problématique liée au stockage sur disques car il faut imaginer que l'on analyse l'état de plusieurs millier de machines ; les états du système correspondent à un échantillonnage toute les minutes, par exemple et ils peuvent concerner une quinzaine d'événements à chaque mesure.

Les traces que nous étudions sont celles qui peuvent s'obtenir dans le cadre du proget `XtremWeb`. [1]

Les traces ont été collectées par un serveur Windows NT en utilisant des outils standard disponibles sous Windows. Toutes les machines tournent Windows et sont des machines de bureau. Les traces peuvent inclurent des trous dûs aux connections, déconnexions, reconnexions des machines.

La trace donne l'activité d'une centaine de machines, sur 14 jours avec (au plus) une mesure tous les quart d'heure. Cette trace représente $78,2$Mo de place disque (décompressé). Par approximation, on atteindra 1.1Go en échantillonnant toutes les minutes et 11Go de données pour 1000 machines échantillonées toutes les minutes. C'est un ordre de grandeur.

Le problème se décompose en plusieurs sous-problèmes parmi lesquels :

- *la collecte :* quels événements mesurer ? avec quels outils ? comment faire en sorte que la collecte se passe sans introduire de coûts de gestion élevés ? Comment gérer l'oubli (les machines ont des disponibilités irrégulières) ?
- *la représentation des données collectées :* sous quel(s) format(s) enregistrer les mesures ? comment faire en sorte que la représentation puisse autoriser des recherches et l'insertion de nouvelles mesures ?
- *les algorithmes d'analyse ou de fouille :* peut-on compter sur les algorithmes de fouilles existants et/ou sur ceux de "pattern matching" pour détecter qu'une séquence d'événements arrive fréquemment ? Qu'est ce qui différencie l'algorithmique en fouille de données de l'algorithmique de "pattern matching" ?

Le problème du placement et de l'ordonnacement étant des problèmes algorithmiques difficiles, le passage par une approche à base d'heuristiques permettra d'obtenir un premier retour sur les événements influençant à priori un système global de calcul.

1. Voir : `http://www.xtremweb.org`

En effet, les techniques connues en ordonnancement ne se prêtent pas à une adaptation dans le cas des grilles car elles ne prennent pas en compte les problèmes de déconnexion.

Nos objectifs sont à terme de fournir une infrastructure capable de collecter, d'analyser en temps réel des traces réelles de machines et de proposer à l'ordonnanceur de travaux d'une plate-forme de calcul global, des heuristiques pour l'aider à placer les travaux. Il s'agira de fournir par exemple des règles du type : « `si` la charge CPU des machines du sous domaine internet `xxx.fr` est supérieur à 50% `alors` le nombre d'opérations d'entrées sorties de chacune de ces machines est toujours inférieur à X ». Cette règle donne donc une corrélation entre la charge CPU observée avec le nombre d'opérations d'entrées sorties observées.

Nous examinerons ensuite comment ces heuristiques pourraient servir à construire un ordonnanceur de travail. La nature, la fréquence des heuristiques devraient aussi permettre d'isoler les évènements ayant un impact certain sur les performances des machines et alors on pourrait construire des ordonnanceurs en connaissance de cause. Pour l'instant, il n'y a pas beaucoup d'études sur le comportement (mesuré) de plate-forme à grande échelle si bine qu'il est difficile de justifier qu'un certain évènement est plus pertinent qu'un autre.

2. Le problème de la collecte des métriques de performances

En programmant des calculs dans un réseau de communication comme Internet, pourra t-on se passer de scruter de temps à autre le réseau afin de prédire les performances du calcul ? Rien ne permet de le dire.

Parmi les projets assez avancés ou l'on trouve un dispositif de collecte des performances des processeurs connectés à Internet, nous pouvons citer le projet `NWS` (Network Weather Service) qui vise à prévoir de manière précise les performances dans un milieu changeant (les nuages bougent !). Les difficultés résident principalement dans l'intégration de techniques de programmation et une conception architecturale qui permettent de tenir compte de l'extension (en nombre de machines) du système. Pour l'instant la plate-forme fonctionne sous Unix en utilisant les sockets TCP/IP. Les principales ressources Internet sont :

http://nws.npaci.edu/NWS/
http://www.cs.ucsd.edu/groups/hpcl/apples/hetpubs.html#NWS
http://apples.ucsd.edu/
http://www.cs.utk.edu/ browne/perftools-review/

Il serait alors intéressant de comparer les approches de `NWS` et celles d'XtremWeb quant aux collectes d'informations liées à la charge. Peut être qu'il serait pertinent d'intégrer à XtremWeb un mécanisme qui donne de l'information sur l'état

des connexions entre routeurs, sur le nombre de routeurs traversés à l'image de ce que l'on obtient avec les commandes Unix `traceroute, ping, netstat, xload`. Par exemple, il y a 12 passerelles à traverser depuis ma machine à la maison pour arriver à `mars.laria.u-picardie.fr` et 14 passerelles pour arriver sur `lhpca.univ-lyon1.fr`. Si l'on considère que la traversée d'une passerelle quelconque se fait à un même coût, alors il est plus favorable d'envoyer du code à `mars` depuis chez moi.

Des heuristiques pourraient alors être trouvées pour placer, par un ordonnanceur, les prochains travaux à distribuer. Il s'agirait par exemple de répondre à la question suivante : une machine souhaite déposer un calcul parallèle dans le système et le faire exécuter sur $Z > 1$ machines. Trouver ces Z machines de sorte que le pourcentage d'utilisation du CPU de chacune d'entre elles ne sera pas supérieur à $x\%$ dans les prochaines heures, que chacune des machines dispose d'au moins r ressources mémoire, que le nombre maximum de routeurs entre deux machines quelconque n'est pas plus grand que n avec des temps d'echo de paquets ICMP garantis entre deux sites quelconques inférieurs à zms dans les prochaines heures.

Examinons maintenant quelques caractéristiques de `LSF` (Load Sharing Facility) qui est un produit de Platform Computing Corporation[2] qui fait de l'ordonnancement de travaux lancés dans des fichiers "batch". `LSF` trouve son origine dans Utopia qui est un système développé par l'université de Toronto [ZZWD93].

Le calcul de priorité des travaux est effectué en gros en deux temps. Premièrement un ensemble de sites candidats est choisi à partir d'une spécification associée au processus devant migrer. Cette spécification prend par exemple la forme suivante :

```
select [ sparc && swap >= 120 && mem >= 64 ]
```

qui indique que les sites à sélectionner devront être une machine de type SPARC avec au moins 120Mo de zone d'échange avec le disque et au moins avec 64Mo de mémoire principale[3]. Dans un deuxième temps le site élu résulte d'un calcul de priorité donné par l'équation 9 :

$$(9)\quad \mathsf{priority} = \mathsf{user\_share}/(0.01 + \mathsf{cpu}_t * \mathsf{CPU\_TIME\_FACTOR} + \mathsf{run}_t * \mathsf{RUN\_TIME\_FACTOR} + \mathsf{run}_j * \mathsf{RUN\_JOB\_FACTOR})$$

avec cpu_t et run_t étant le temps depuis le lancement et le temps où le "job" a été actif, respectivement ; run_j étant le nombre de travaux lancés. En fait le terme cpu_t est pondéré afin de faire en sorte que le temps CPU récent compte plus que le temps CPU du passé. Les termes CPU_TIME_FACTOR, RUN_TIME_FACTOR et

2. Voir `http://www.platform.com/`

3. On peut aussi spécifier des critères sur le taux de pagination, le débit des IO

RUN_JOB_FACTOR sont des constantes qui par défaut prennent les valeurs $0.7, 0.7$ et 3.0, respectivement.

En fait LSF effectue uniquement de la migration de données et de code résident en mémoire principale via un mécanisme de "checkpointing" (voir [PBKL95] pour un exemple d'implémentation disponible dans le domaine public). On suppose plutôt que l'application est en mémoire et ne travaille qu'avec la mémoire. Si des fichiers sont ouverts, les descripteurs sont migrés mais pas les contenus des fichiers. On suppose en fait dans LSF qu'il y a un système de disques indépendant des processeurs alors que ce que nous cherchons à lier de la meilleure manière possible sont des processeurs associés à des disques (des PC). LSF n'est pas de notre avis complètement réutilisable dans notre cas comme système de répartition de charge car nous supposons de plus que les applications travaillent avec leurs disques locaux et donc, qu'en cas de demande de déconnexion, il sera nécessaire de migrer les << données disque >> (soit physiquement, soit en jouant sur la redondance). LSF ne permet pas non plus aux applications de demander explicitement la migration.

3. Représentation des données

Le développement d'un vocabulaire commun en matière de représentation de l'information donnée par la collecte d'événements est essentiel pour permettre l'interagissement d'un nombre croissant de projets et de tests de performance des systèmes globaux de calcul.

Pour illuster notre problématique de recherche, prenons comme exemple la trace disponible dans le projet XtremWeb. L'information disponible se présente (en partie) sous la forme suivante pour une certaine journée :

```
Time;      CPU%;Interrupts/sec;Mem Kb available;Mem Page faults/sec;
1018453478;   0;      79250486;           21188;            3245252;
1018454066;   8;         71532;           21208;               1509;
1018455865;   2;         69204;           18148;               6442;
.......
```

Le premier problème qui se pose est lié au temps. En effet, le temps mesuré dans la colonne "Time" correspond au temps local sur une machine donnée. Si l'on veut pouvoir dire avec précision si un événement à lieu après un autre sur deux machines distinctes il est nécessaire de se référer à un temps global ou universel.

Plusieurs techniques sont connues comme les horloges de Lamport [Lam78] afin d'estampiller des événements de manière unique. Ce n'est pas exactement ce que l'on cherche. On veut pouvoir dire quels sont les événements qui sont arrivés sur l'ensemble des machines pendant les X dernières milli-secondes. Eventuellement il peut y avoir plusieurs événements qui se sont passés de manière concurrente.

Il s'agit alors d'un problème de synchronisation. Le projet `NTP`[4] (Network Time Protocol) est opérationnel depuis de nombreux mois et il fournit des outils pour synchroniser l'horloge de machines connectées à Internet.

L'idée est simple : des serveurs qui échangent des informations de synchronisation entre eux garantissent à des clients une heure « universelle » . Le protocole est donc un protocole client/serveur. La réalisation est déliquate. Cependant, il est spécifié que l'heure obtenue est l'heure universelle avec une garantie de l'ordre de la milliseconde. Pour notre dispositif de claclul global, il semblerait que cette précision soit très supérieure à ce que l'on veut. À priori le problème de l'horloge commune est donc réglé.

Pour en revenir au format des traces données à l'exemple ci dessus, on peut rapidemment noter que l'information donnée peut être exploitée de manière rusée si par exemple on organise pour le champs "Time" un arbre virtuel afin d'accélérer par exemple une recherche. La racine de l'arbre pointe par exemple sur un fils contenant les traces antérieures à une date, l'autre fils pointe sur les traces postérieures à une certaine date.

3.1. Le Grid Forum et les traces. Cette idée simple n'est pas prise en compte dans le groupe de travail "Discovery and Monitoring Event Description (DAMED-WG)" du `Grid Forum`. Pourtant, la charte du groupe de travail stipile que "the working group aims to define a basic set of monitoring event descriptions. These descriptions, or schemas, will describe the information (attributes) associated with a particular data element and will describe conventions for the representation of the value associated with it. The aim of this group is also to develop standard representations of the most widely used measurement values (the "top N".) From this we envision the emergence of a set of conventions and recommendations that will ease the task of defining richer, domain-specific schemas."

Le groupe DAMED a proposé sur ces thèmes une notation pour les événements. Il s'agit d'une notation poinfixée du type :

```
<Target Type>.<Event Name>.<Event Refinement>

With (example):

o Target Type = Host | DiskPartion | Scheduler | ProcessId | NetworkLink
o Event Name  = CPU load | System uptime | Disk size | Disk used
o Event refinement = non volatile | disk
```

4. Voir `http://www.ntp.org/`

Ce genre de notation est aussi discutée dans la RFC 1514.[5] Pour l'instant les travaux du groupe DAMED se limitent à cet aspect.

Plus ambitieux, le projet `Spitfire`[6] est un "Work Package 2" dans le projet européen DataGrid qui vise à fournir un moyen uniforme pour accéder à des bases de données au moyen de protocoles standards et d'interfaces communes. Spitfire est donc un intergiciel pour les grilles qui vise à uniformiser les accès, dans un langage commun, aux bases les plus courrantes. Les services sont implémentés comme un servlet Java. Il ne s'agit pas d'imaginer de nouveaux moyens d'accès ou de nouveaux formats de représentation des données. Le projet mérite qu'il soit cité pour les ambitions qu'il traite.

3.2. Quelles vues des données? Si le projet précédent s'attache à uniformiser pour un utilisateur, l'accès aux bases de données, il peut y avoir un intérêt, au niveau des langages qui permettront ces abstractions, à offrir des supports de programmation pour lire ou écrire certaines parties du fichier sans se préoccuper de la connaissance totale de l'organisation du fichier de données.

Par exemple, un support de programmation utile dans ce contexte nous semble être celui offert par MPI-2 pour la gestion de fichiers. En MPI-2, le programmeur qui fait des accès à des fichiers peut considérer qu'un fichier est une collection de types de données (la définition que l'on peut donner est éventuellement récursive) qui commence à partir d'un certain offset dans le fichier et qui est éventuellement répétée tous les X bytes dans le fichiers.

On appelle une telle description une *vue*. Plusieurs *vues* peuvent être données pour un même fichier afin d'accéder à différentes parties du fichier. Les *vues* peuvent se chevauchées et une sémantique de l'accès est alors à préciser. Ce support de programmation est très commode.

Le format CFD (The Common Data Format)[7] offre par certains aspects les fonctionalités précédentes. C'est un format conçu pour des données multi-dimensionnelles. Aucune discipline scientifique n'est visée en particulier : c'est un format qui se veut générique. On peut choisir (programmer) les formats des fichiers de données ainsi que le format des fichiers de "méta-données" qui contiennent des informations sur la structure des données.

Certaines limitations à l'usage de CFD sont connues :

(1) la compression des données n'est à priori pas autorisée (il y a de l'information contradictoire à ce sujet) ;
(2) on ne peut pas décrire des données creuses ;

5. Voir aussi `http://www.ietf.org/rfc/rfc1514.txt`
6. Voir `http://spitfire.web.cern.ch/Spitfire/`
7. Voir `http://nssdc.gsfc.nasa.gov/cdf/cdf_home.html`

Les autres formats largement employés dans la communauté scientifique pour stocker des données sont netCDF [8] et HDF [9]

Par exemple, les fichiers de données HDF sont délicats à mettre à jour. Les données sont physiquement rangées de manière contiguë. Ainsi si un enregistrement demande à être augmenté, c'est l'ensemble du fichier qui doit être ré-écrit. Le modèle de données `netCDF` est basé sur celui de `CDF`. Comme il est postérieur, il offre un certain nombre d'extensions comme par exemple la possibilité de lier `netCDF` avec du langage `C`, il est portable sur un grand nombre de systèmes, le format des données est indépendant de la machine (il utilise XDR). Des outils permettent maintenant de convertir une description `CDF` en `XML` afin d'assurer une inter-opérabilité entre les formats de données scientifiques.

Il serait alors tout particulièrement intéressant d'avoir un intergiciel comme `Spitfire` permettant d'uniformiser les requêtes d'accès aux fichiers de données scientifiques. C'est un enjeu majeur dans le cadre des projets à base de grilles de calcul. Pour l'instant, l'état des recherches est celui que nous venons de décrire : il y a coexistence de diffférents formats mais pas d'interface commune pour effectuer les opérations de base que sont les mises à jour, la recherche, la suppression de données.

4. Fouille de données

4.1. Introduction. Une des motivations spécifiques pour s'intéresser aux entrées sorties hautes performances peut être la fouille des données (Data Mining) en parallèle. La fouille des données peut se définir comme étant la recherche d'informations pertinentes dans une grande quantité de données en utilisant des techniques de calcul et de stockages efficaces.

On trouve dans la littérature d'autres définitions (voir les liens Internet dans l'encart en fin de document) qui varient un peu selon que l'on considère ou pas que les données sont organisées dans une base. On peut aussi considérer *qu'il s'agit de l'exploration et de l'analyse d'une masse très importante d'informations.*

Par exemple, il pourrait s'agir de trouver le sous ensemble des attributs qui apparaissent fréquemment dans les enregistrements d'une base ou dans des transactions. Il peut s'agir d'extraire les règles qui précisent comment un sous ensemble d'items conditionne la présence d'un autre sous ensemble (par exemple « 90% des clients qui achètent le produit A, achètent aussi le produit B »). Dans ce cas, la fouille de données s'intéresse à la segmentation des clients, à la conception de catalogues, à l'agencement des magasins et elle recouvre alors ce que l'on appelle « l'association rule mining » [Zak25] que l'on pourrait traduire par la « découverte de règles d'association ».

8. `http://www.unidata.ucar.edu/packages/netcdf/index.html`
9. `http://hdf.ncsa.uiuc.edu/`

Il s'agit de produire des règles qui décrivent les relations entre différentes séquences. Par exemple, si la séquence (BF) arrive 4 fois tandis que (ABF) arrive 3 fois on produit la règle « si BF arrive alors il y a 75% de chance de voir arriver A ».

Nous pouvons nous représenter la *fouille de données* comme un sous ensemble du processus de gestion des connaissances incluses dans une base (Knowledge discovery from databases or KDD). Ce sous ensemble inclue aussi la compréhension de l'application, la réduction de la taille utile des informations, la correction des données corrompues.

Notre à-priori de départ est donc celui ci : *la façon de stocker et de représenter l'information conditionne la qualité et la vitesse d'analyse de cette même information.*

Le type de l'information déterminera quel sera l'algorithme (parmi un ensemble d'algorithmes) à utiliser pour une technique de fouille de données : regoupement (clustering), découverte de règles d'association (association rule discovery), classification, découverte de motifs, détection.

4.2. Points de terminologie et objectifs. Nous proposons maintenant d'expliciter des points de terminologie et de présenter nos objectifs. Il s'agit de découvrir les relations et les motifs qui apparaissent dans de grandes masses d'information. Nous reprenons ici le cas de la fouille de traces donnant les événements qui se sont passés sur un ensemble de machines de bureau.

Il est d'usage de caractériser les données en entrées des algorithmes de fouille à partir de trois champs : les champs objet, temps et évènement. Par exemple, pour le jeu de données suivant issu de la machine A :

```
Time;      CPU%;Interrupts/sec;Mem Kb available;Mem Page faults/sec;
1018453478;   0;      79250486;           21188;            3245252;
1018454066;   8;         71532;           21208;               1509;
1018455865;   2;         69204;           18148;               6442;
.......
```

l'objet sera la machine A et on peut s'intéresser au temps 1018453478 et à l'événement « CPU ».

Bien que le problème soit important, nous ne connaissons qu'un seul article qui analyse les traces de plusieurs dizaines de PC interconnectés. Il s'agit de l'étude de Bolosky et Ely [BDET00]. Les auteurs s'intéressent à l'usage des disques, de la charge des processeurs et à quelques autres métriques. Il s'agit de mesurer les performances d'un système de fichiers distribué sans serveur. La conclusion principale est que le cluster de PC utilisé n'est pas adapté à l'usage d'un tel dispositif.

Pour estimer la disponibilité des 64610 machines du cluster, les auteurs ont utilisé la commande Unix *ping*, une fois par heure et ceci pendant cinq semaines. Un calcul

de corrélation est ensuite mis en œuvre. Toutes les courbes présentées dans l'article [BDET00] sont des études statistiques. Ce n'est pas notre approche ici.

Généralement on recherche les *relations séquentielles* entre les événements. Si A, B sont deux événements, on recherche les associations ou les motifs séquentiels << Quand A arrive alors B arrive dans un certain futur >> .

Une *relation séquentielle* se représente par un graphe acyclique comme par exemple celui de la Figure 1. Sur la Figure 1(a), l'événement (A) arrive avant l'événement (N). Éventuellement, les nœuds peuvent contenir plusieurs événements comme par exemple le nœud (C,B). L'arête (appelée aussi la contrainte) entre (A) et (N) est étiquetée par $\{a4, b1\}$ pour spécifier (pour $a4$ qui est un entier) le temps minimum entre le dernier événement observé dans le nœud (A) et le premier événement du nœud (N) et le temps maximun (pour $b1$ qui est aussi un entier) entre l'événement qui arrive en premier dans le nœud (A) et l'événement qui arrive en dernier dans le nœud (N).

À chaque nœud on peut aussi associer une *contrainte de nœud* notée [w2] pour le nœud (A) de la Figure 1(a) et qui permet de préciser quand un événement peut être associé à un nœud c'est à dire le temps maximal autorisé pour que les événement du nœud apparaissent.

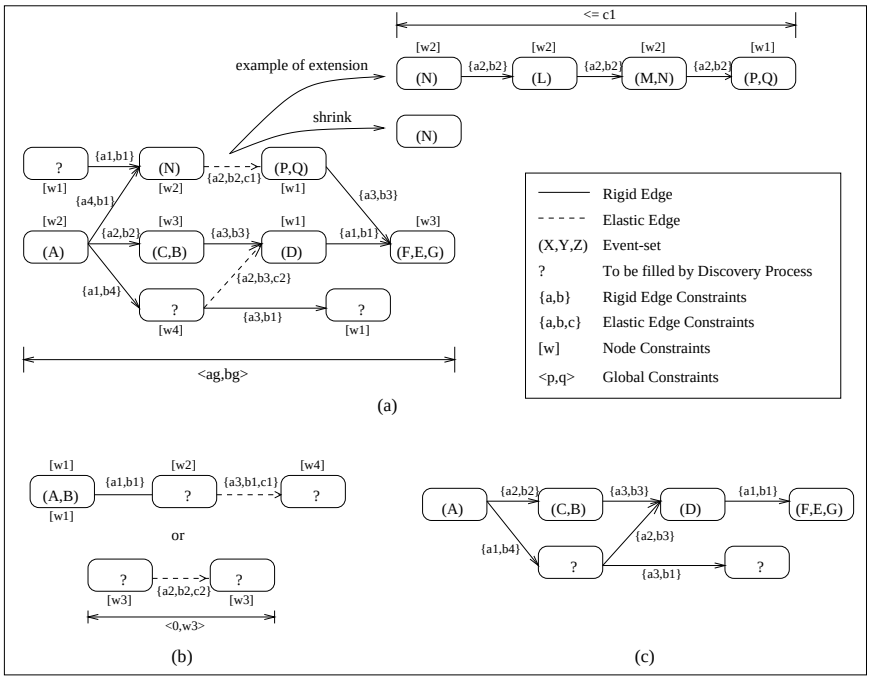

FIG. 1: *Représentation des relations séquentielles*

Enfin, il peut y avoir une contrainte globale sur le motif, notée $< \mathtt{ag}, \mathtt{bg} >$ sur la Figure 1(a) qui rend compte du temps maximal autorisé pour découvrir le motif.

Les arêtes peuvent être *élastiques* ou *rigides*. Pendant le processus de découverte, les premières peuvent être dilatées en ajoutant des nœuds de manière dynamique. Pour les secondes, on n'a pas cette possibilité. Le terme $\mathtt{c}$ dans la notation $\{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ spécifie alors le temps maximum autorisé pour une dilatation.

Sur les Figures 1(b) et 1(c) nous avons d'autres approches [SA96, MTV97] pour spécifier les motifs séquentiels. On restreint la relation à un seul chemin. De plus on restreint aussi à une seule arête élastique (Voir la Figure 1(b)) ou on n'a pas de contrainte sur les événements (Voir Figure 1(c) - [SA96]).

Étant donnée la structure de donnée (séquence) présentée à la Figure 1, il s'agit alors de mettre en place des algorithmes qui vont repérer la relation dans la donnée qui est soumise.

Une séquence est *intéressante* si elle apparait suffisamment de fois dans l'entrée et si elle satisfait les contraintes.

Il y a alors deux points à préciser : 1) que veut dire « suffisamment de fois » 2) comment les occurrences d'une séquence sont elles comptées ?

Pour le point 1), on considère généralement un seuil qui est un paramètre donné de l'algorithme. L'usage d'un seuil a généralement un avantage lié à la complexité des algorithmes : le seuil d'une séquence ne peut pas être plus grand que le seuil de toutes ses sous-séquences.

Pour le point 2), on distingue trois méthodes de comptage. Soit on recherche une seule occurrence, soit on compte le nombre de fenêtres (intervalles de temps) dans lesquelles la séquence apparaît, soit on compte les occurences distinctes de la séquence. Les différentes méthodes de comptage sont illustrées à la Figure 2.

Sur la Figure 2 on a indiqué que les méthodes CWIN et CDIST_O sont similaires car elles comptent toutes les fenêtres ou les occurences. On a aussi indiqué que CMIN-WIN et CDIST sont similaires car elles comptent un nombre minimal de fenêtres ou d'occurences.

Sur la Figure 2, la séquence recherchée est (A)(B) et elle doit se situer dans une fenetre de $\mathtt{ms} = 2$ unités de temps. L'entrée du problème est la séquence (A)(A)(A,B)(B)-(A,B)(B,A)(B). On a que la sous-séquence (A,B), où les événements A et B arrivent simultanément, apparaît aux temps $3, 5, 6$.

L'utilisation de l'une ou l'autre des méthodes dépend du domaine d'expertise de l'utilisateur. Il est très important de noter qu'en plaçant des contraintes dans les méthodes de cpmptage, par exemple en spécifiant que le motif doit apparaitre dans un intervalle de temps défini, on réduit le temps de génération de la solution.

Examinons maintenant un algorithme de fouille qui prend en entrée une description de sequence. Il s'agit de WINEPI [MTV97].

4.3. Algorithmes de fouille. La complexité des algorithmes de fouille de données est lié au fait que le nombre maximum de sequences ayant k événements est $\mathcal{O}(m^k.2^{k-1})$ où m dénote le nombre total d'événements distincts dans l'entrée. Il n'est donc pas envisageable d'énumérer toutes les séquences. Les algorithmes capitalisent d'une part sur la notion de seuil et sur les contraintes temporelles imposées pour réduire les coûts en temps de la découverte des motifs.

Un grand nombre d'algorithmes, dont WINEPI [MTV97], fonctionne généralement selon le principe suivant qui décrit une itération de l'algorithme :

(1) on génère une liste de sous-séquences fréquentes de longueur k (appelées des candidates) à partir du résultat de l'itération précédente qui a généré des candidats de longueur $k-1$;

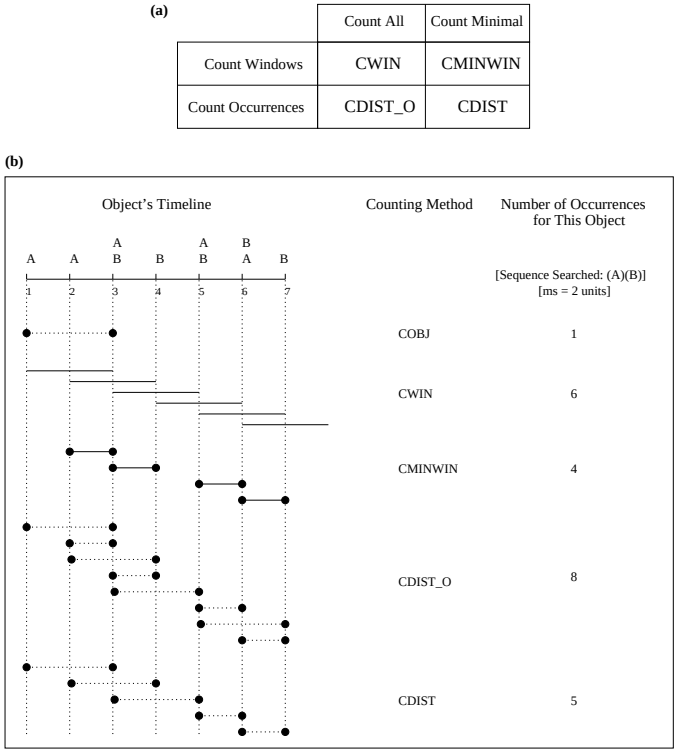

FIG. 2: *Méthodes de comptage des sequences fréquentes. (a) Relations entre méthodes comptant les occurrences multiples (b) Différences entre méthodes.*

(2) on sélectionne parmi les candidates celles qui respectent les contraintes et le seuil ; on les appelle les sous-séquences fréquentes de longueur k.

Cette approche est motivée par le fait que pour qu'une séquence soit fréquente il faut que toutes les sous-séquences soient également fréquente. On contrôle ainsi la complexité du calcul de découverte.

WINEPI est un algorithme qui a été conçu originellement pour découvrir les épisodes fréquents dans un réseaux d'alarme en télécommunication.

Les séquences recherchées peuvent être des séquences *séquentielles* ou *parallèle* avec la signification que dans ce dernier cas les événements peuvent apparaitre au même instant. Dans l'algorithme WINEPI, il y a donc la recherche des « candidats parallèles » et des « candidats séquentiels » .

4.4. Recherche de motifs. Les algorithmes de recherche de motifs [CH99, BYP92, ACGM92] peuvent potentiellement s'employer pour détecter la présence d'un motif dans un jeu de données.

Nous ne connaissons pas d'études comparant ces approches avec les approches discutées au précédent paragraphe. Par exemple, aucune étude permettant de séparer les complexités de chaque approche n'a été faite. Probablement parce que le problème est difficile et que l'on ne sait pas comment exprimer la complexité : en fonction de quoi ? Du nombre de contraintes ? De la « longueur » du motif recherché ?

De nombreux algorithmes sont connus pour les problèmes d'alignement de séquences, de recherche de la plus grande sous chaîne commune à deux chaînes... Toutes ces solutions sont potentiellement réutilisables moyennant quelques adaptations pour notre cas. En effet, dans note cas, les données sont en très grand nombre (plusieurs giga octets) : il faut donc inter-agir avec les disques. Nous ne connaissons pas d'études qui permettent de quantifier le nombre d'opérations d'entrées sorties optimal pour chacun des problèmes d'alignement ou de sous chaîne commune.

Ensuite, il semblerait que les outils actuellement disponibles soient capables de retrouver qu'un motif séquentiel. Quels sont les outils permettant de repérer un motif dans un espace multidimensionnel.

Si l'on prend l'outil `nrgrep`[10] de Gonzalo Navarro qui est une extension de la commande `grep` d'Unix tout en se comportant mieux, on remarque que l'on spécifie les motifs au moyen du formalisme des expressions régulières. L'outil est conçu pour faire de la recherche approchée (approximate string matching) c'est à dire que l'on autorise éventuellement k erreurs dans le motifs. C'est un point remarquable par rapport aux autres outils similaires.

10. `http://www.dcc.uchile.cl/~gnavarro/pubcode/`

Voici des exemples d'utilisation de `nrgrep` qui montrent certaines abstractions offertes et qui sont appliquées à notre cas. Par exemple, pour rechercher le nombre de fois où le processeur de la machine `pc101` est chargé entre 10 et 19%, on tape :

```
> cut -d ";" -f 2 pc101.log | nrgrep -n -c '\"1.\"'
12:13:27:37:Total: 4 matching records
```

Pour rechercher si on a au moins une occurence du motif "91" suivi de "0" :

```
> nrgrep -d XXX -n -c '\"91\"\n.\"0\"' titi
1:Total: 1 matching records
>
titi:
CPU%
 "91"
 "0"
 "49"
 "3"
  .
  .
  .
 "91"
 "0"
 "0"
XXX
```

Sur cet exemple on remarque que l'on n'a pas les moyens avec `nrgrep` de compter toutes les occurences d'un motif. C'est une limitation importante pour notre problème. Sur cet exemple il faut aussi remarquer le symbole `.` dans le motif qui permet de spécifier qu'après le symbole `\n` il peut y avoir n'importe quel caractère.

On peut aussi spécifier des motifs qui sont des expressions régulières. Par exemple le motif `ab(cd|e)*fg?h` précise qu'il commence par `ab` puis il est suivi de `cd` ou `e`, zéro fois ou plus (notifié par l'étoile) puis arrive le symbole `f` puis éventuellement le symbole `g` (ce qui est notifié par le symbole `?`) puis enfin arrive le symbole `h`.

Pour notre trace de 100 machines et au moyen des outils `nrgrep, python` principalement nous avons trouvé que 87% du temps, les processeurs avaient une charge comprise en 0 et 20%. Le résultat complet est :

00% - 09%	45932	observations
10% - 19%	5994	observations
20% - 29%	1801	observations
30% - 39%	838	observations
40% - 49%	1350	observations
50% - 59%	789	observations
60% - 69%	635	observations
70% - 79%	850	observations
80% - 90%	327	observations
90% - 99%	591	observations

Il y a donc un grand potentiel à exploiter !

Les expressions régulières sont utilisées dans SPIRIT [GRS99] pour spécifier les contraintes sur le motif à rechercher. L'approche faite par SPIRIT offre un haut niveau d'abstraction pour spécifier les contraintes et il sert aussi à accélérer le travail de recherche. L'originalité de l'approche tient aussi à ce qu'elle permet de s'affranchir de la propriété d'inclusion (un motif est fréquent si toutes les sous-séquences sont fréquentes).

4.5. Limitations. Les algorithmes et les formulations des motifs présentés ici considèrent uniquement des objets à une dimension : le seul attribut utilisé est le temps. Dans la réalité, la situation est plus complexe. Pour notre problème de fouille des événements donnant l'état des machines, il y a une quinzaine d'attributs potentiels. Nous avons donc à faire une analyse multidimensionnelle [PHP+01].

On peut envisager d'attaquer le problème de deux manières : soit un pré-calcul permettra de garder des événements utiles et on ré-applique un des algorithmes connus, soit on intègre une analyse de l'aspect multidimensionnel directement dans l'algorithme.

La première idée revient à faire du *regroupement* (clustering), la deuxième idée revient de concevoir complètement de nouvelles solutions.

Par exemple, pour l'algorithme MINEPI [MTV97] qui est une variante de WINEPI, les événements peuvent avoir un nombre arbitraire d'attributs mais l'algorithme ne les utilise que pour contraindre les séquences en entrée mais ces attributs ne sont pas utilisés dans la méthode de comptage des épisodes fréquents.

L'algorithme SPADE [Zak01] a une extension pour les données qui ont une information d'appartenance à une classe. L'algorithme détermine les séquences fréquentes pour toutes les classes.

L'algorithme SPIRIT [GRS99], parce que les contraintes se présentent sous la forme d'expression régulières, peut très certainement être utilisé pour traiter le problème de fouille en considérant plusieurs dimensions. Pour l'instant cela reste un problème de recherche.

On remarque donc que de nombreuses approches ont été proposées récemment pour la fouille de données. Souvent les approches sont développées pour un problème

spécifique. Elles sont donc limitées à un domaine d'expertise. Les études que nous avons à conduire dans le cadre de la fouille d'événements issus de l'état de machines connectées à Internet pourront capitaliser sur l'existent. Nous n'avons pas encore arrêté de choix définitif quand à l'approche préférentielle choisie. L'objectif étant de fournir des règles d'association pour guider l'ordonnanceur de travaux de la plate-forme de calcul global.

5. Nos perspectives en ordonnancement

Il est clair qu'aucune des techniques d'ordonnancement actuelles n'est adaptée au calcul global car aucune d'entre elles ne captent complètement le fait que certaines informations proposées ci dessus peuvent être incomplètes et s'obtenir de manière dynamique. L'ordonnancement statique ne semble pas une technique appropriée. Pour le tri en parallèle, les méthodes que nous avons présentées dans les chapitres précédents et qui pré-déterminent une charge maximale par processeur sont à re-visiter, de ce point de vue.

L'histoire d'Internet et de ces dispositifs fondamentaux comme le routage et les annuaires montre que les structures hiérarchiques fonctionnent à grande échelle. Doit on envisager un système de régulation de charge organisé hiérarchiquement ou chaque niveau de la hiérarchie serait responsable de répondre de la charge pour un groupe de machines?

Avant cela se pose le problème de la collecte de l'activité d'un grand nombre de machines, de la représentation de cette activité. Aucun travail d'ampleur n'a été conduit à ce jour par le Grid Forum concernant le format d'échange des données. C'est un problème important qui mérite plus d'attention. De même, l'algorithmique de la fouille de données à large échelle est très largement sous-étudiée dans le cadre du Grid Forum. Ces deux exemples montrent bien l'ampleur du travail qui reste à réaliser.

CHAPITRE 6

Conclusion générale

Notre démarche scientifique vise à définir les concepts et les outils pour résoudre efficacement les problèmes avec de grandes masses de données sur des architectures de grappes et grilles.

Dans ce contexte, les problématiques que nous étudions sont au nombre de quatre : l'architecture des *machines multiflots* ainsi que l'impact de la hiérarchie mémoire sur les performances, l'algorithmique sur mémoire commune, l'algorithmique et la programmation de grappes et grilles de calcul et enfin les entrées sorties performantes (problèmes nécessitant les disques) dans les grilles.

Notre démarche est à double sens : elle est expérimentale dans le sens ou nous cherchons à implémenter nos idées (nos algorithmes) et nous fournissons en ensemble exhaustif de mesures. Nous cherchons aussi des garanties algorithmiques prouvées, en particulier pour l'équilibrage des charges et les temps d'exécution. Ce dernier point permet de donner une fondation solide aux résultats obtenus y compris dans l'étude de l'impact du cache dans les algorithmes de tri séquentiel.

La demande croissante de puissance de calcul s'accompagne également d'une demande croissante en matière de stockage. La vidéo à la demande, les bases de données (stockage de traces, fouille de données), la génomique, le calcul numérique (la modélisation) font parties des domaines scientifiques et techniques les plus demandeurs.

Parmi les solutions matérielles discutées récemment dans la littérature afin d'offrir soit des puissances de calcul de l'ordre du téra-flops soit des capacités de stockage de l'ordre du péta-octets nous trouvons l'interconnection des grands centres de calcul ou la mutualisation des ressources des PC connectés à Internet. Le premier dispositif revient à s'intéresser aux grilles, le second aux systèmes à large échelle. Cette classification des systèmes distribués est très pragmatique et pas encore définitive.

Les logiciels à concevoir pour ces types d'architecture doivent avoir des propriétés supplémentaires par rapport aux propriétés habituelles (correction). Il faut concevoir des solutions qui vont interagir sur des réseaux longue distance, il faut tenir compte des déconnections (et de différentes fautes plus difficiles à gérer), il faut prévoir que les logiciels vont être utilisés simultanément par plusieurs dizaines de milliers d'utilisateurs et que différentes versions persisteront au cours du temps.

Le dispositif `Seti@Home` de l'université de Berkeley a popularisé le concept de *jachère de calcul* : un particulier connecté à Internet offre son temps CPU pour participer à un calcul d'astrophysique qui recherche de la vie extra-terrestre (SETI: Search for Extraterrestrial Intelligence). L'utilisateur télécharge un code qui s'exécute sur sa machine. De manière générale, dans le cadre de la mutualisation des ressources des PC, il faut faire en sorte que l'usage des logiciels ne puisse pas altérer les machines qui

offrent des ressources et que les codes s'exécutant sur les PC ne fassent pas écrouler leurs performances.

Notre démarche est aussi une démarche intégratrice : nous dégagons des techniques d'usage général pour traiter un problème en mémoire centrale ou sur disque. Notre plan d'intégration est systématiquement le suivant, quelque soit la thématique abordée. Premièrement nous comparons l'existant afin de mettre en évidence des conflits par rapport à des objectifs assignés. Deuxièmement nous mettons en conformité des conflits. Troisièmement nous effectuons une fusion des schémas connus. Enfin nous restructurons les algorithmes afin d'améliorer, au sens des critères retenus, le schéma global.

Cette démarche a été apliquée, par exemple, pour la problématique de l'impact du cache dans les algorithmes de tri séquentiels.

Pour les PC qui sont les nœuds de calcul des systèmes à large échelle, nous examinons l'impact du cache dans les performances. Comme les processeurs actuels ont plusieurs niveaux de cache afin de réduire les temps d'accès à la mémoire et qu'ils peuvent faire de l'éxécution spéculative dans plusieurs unités de calcul de manière simultanée, nous avons dû trouver des compromis algorithmiques permettant d'exploiter au mieux les processeurs.

Nous avons sélectionné plusieurs algorithmes qui tiennent compte du cache et nous avons développé notre propre algorithme en visant la minimisation du nombre de défaut de cache. Cet algorithme fusionne plusieurs idées et il a plusieurs propriétés : c'est un tri par fusion, on peut donner le nombre exact de défauts de cache qu'il provoque et il peut ainsi être comparé avec une borne d'optimalité.

Nous proposons aussi une conjecture à l'encontre des idées préconçues permettant au concepteur de programmes de balancer le travail productif (opérations de comparaisons) vis à vis du travail non productif (traitement de la gestion des défauts de cache).

La conjecture est la suivante : « Soient X et Y deux programmes répondant à la même spécification. Si le programme X a une IPC (nombre d'Instructions Par Cycle) au moins deux fois plus grand que Y mais qu'il a un nombre de défauts de cache L1 et un nombre d'instructions exécutées deux fois plus important que Y, alors le programme X a quand même une exécution en temps meilleure que Y ».

Cette conjecture est très importante parce qu'elle ouvre des pistes de travail afin de construire des algorithmes qui exhiberont suffisamment d'instructions indépendantes pour alimenter les différentes unités d'exécution des processeurs. Les algorithmes actuels tel que QuickSort même optimisé pour le cache ne les exploitent pas de manière efficace. Notre approche bat ce ttype d'algorithme par un facteur de 9 à 15%. Plus les optimisations de code sont nombreuses, meilleur est notre algorithme. Nous pensons

encore pouvoir améliorer ce gain en combinant les points forts des différentes approches visitées. Cela nous amène légitimement à revisiter le travail des 40 dernières sur le tri séquentiel dans le sens d'une meilleure exploitation du cache et des unités fonctionnelles d'exécution.

Les nœuds de calcul dans une grille sont constitués, par exemple, de machines parallèles dédiées comme les machines à mémoire communes (carte bi-quadri ou octo processeurs, nœud d'une SGI Origin 2000) ou des machines multiflots (machine TERA de Cray). Pour tirer les performances vers le haut, il faut se poser des questions relatives à l'architecture matérielle sous-jacente.

Par exemple, une question qui nous intéresse pour les machines multiflots est de savoir s'il vaut mieux utiliser une machine multiflots à 64 contextes matériels et 4 unités d'exécution plutôt qu'une machine à 32 contextes matériels et 8 unités d'exécution.

Notre problématique d'évaluation de performances des machines multiflots vise à fournir au concepteur un retour sur les performances attendues. Notre but a été de fournir des fondations théoriques à cette problématique. Pour l'instant les choix architecturaux faits par les architectes sont plutôt empiriques.

Notre approche ne fait aucune hypothèse sur les lois d'arrivée des événements contrairement aux précédentes études d'analyse des performances basées sur les files d'attente. Ainsi, nous ne mesurons pas une valeur moyenne mais nous fournissons un encadrement (une borne inférieure et une borne supérieure) aux performances. Ceci nous distingue très clairement des précédents travaux en la matière.

Étant donné un modèle à un seul processus et des contraintes précisant par exemple combien d'unités d'exécution on veut modéliser, nous calculons la meilleure et la moins bonne performance attendue pour une machine à $2, 3, \cdots, n$ processus. L'aire comprise entre les courbes de la meilleure et de la moins bonne performance représente la qualité du système multiflot.

Pour la problématique du calcul en mémoire commune, nous nous posions des questions en matière à la fois de conception d'algorithmiques et d'expressivité des modèles, en particulier les modèles PRAM (Parallel Random Access Memory) et BSR (Broadcast with Selective Reduction). Le modèle BSR permet d'exprimer le parallélisme dans une abstraction très forte ce qui conduit à des spécifications très condensées. Nous avons étudié finement le modèle BSR afin d'en comprendre les limites et de capter certains problèmes qui n'offriraient pas de solutions optimales c'est à dire qui ne se paralléliseraient pas très bien dans le modèle. Nous avons donné un algorithme optimal pour le problème du segment de somme maximale à une et deux dimensions dont l'intérêt a été introduit par Ulf Grenander à l'université de Brown (USA) dans le cadre de la recherche de motifs dans des images digitalisées. Notre apport est l'utilisation d'un nombre d'instructions BSR moins important que les autres solutions connues à ce jour.

Dans le modèle PRAM, nous avons étudié la mise en correspondance de parenthèses qui est un problème qui intervient dans l'évaluation en parallèle des expressions arithmétiques. Nous avons obtenu un algorithme qui égale en complexité le meilleur algorithme connu mais, de par son organisation par vague, est favorable à une implémentation simple et rapide dans un langage de programmation par passage de messages.

Pour les clusters de PC qui peuvent être des nœuds de calcul d'une grille, nous posons la problématique du tri lorsque les processeurs du cluster ne vont pas tous à la même vitesse. À notre connaissance, ce problème n'avait jamais été étudié dans le passé. Toutes les nombreuses études que nous connaissons ne traitent que du cas homogène. Ce problème est très important pour les personnes qui composent un cluster à partir de processeurs d'ancienne et de nouvelle génération.

Les approches classiques du tri en parallèle ne fonctionnent plus dans le cadre hétérogène sur des clusters. En effet, il faut à la fois minimiser le nombre d'échanges entre nœuds car le support de communication est supposé avoir des performances modérées (classiquement on fait du recouvrement calcul communication) mais surtout les volumes des communications doivent se faire en fonction des performances de chaque nœud de calcul.

Nous avons complètement défriché ce problème et nous avons apporté des solutions qui sont implémentées, testées, validées. Nous avons privilégié une approche statique.

Au départ les données sont réparties de manière proportionnelle aux vitesses des processeurs. Cette répartition n'effectue aucun travail utile pour le tri : les données ne sont pas réparties en fonction de critères spécifiques, elles sont réparties en fonction du critère du nombre uniquement. Aucun ordonnancement à l'exécution n'est effectué. Ces remarques définissent la pré-condition du problème.

Les post-conditions obtenues sont que les processeurs ont au final une quantité de données triées qui diffère de ce qu'ils avaient au début par un facteur de quelques pour-cents dans la pratique. La difficulté majeure consiste à garantir qu'à à aucun moment un processeur ne traite un ensemble de données plus important de ce qu'il avait au départ.

Nous avons imaginé et justifié des techniques permettant de faire de l'échantillonage dans un contexte hétérogène. Il s'agit de sélectionner des valeurs dans l'entrée (des valeurs appelées des pivots) de sorte que les valeurs sélectionnées partitionnent l'entrée en portion de tailles proportionnelles aux vitesses des processeurs.

Notre résultat d'équilibrage a été acquis en garantissant des bornes théoriques au travail effectué par chacun des processeurs hétérogènes. Des expérimentations ont également été accomplies et nos codes se comportent d'excellentes façons dans la pratique : les équilibrages obtenus sont proches des optimaux. Parfois, l'équilibrage mesuré diffère de l'optimal par un facteur inférieur à 1%.

Les stratégies de développement d'algorithmes parallèles qui prédéterminent la distribution initiale des données uniquement en fonction du critère du nombre de données sont donc des stratégies effectives y compris pour des problèmes irréguliers où la prédiction des accès à la mémoire ne peut pas s'envisager sans un coût élévé.

Cette stratégie est générale et nous avons montré qu'elle permettait de capter aussi bien des problèmes qui tenaient en mémoire que des problèmes qui nécessitaient les disques. C'est une des leçons importantes que nous pouvons donner en conclusion de l'étude du tri. Nous confirmons ainsi la porté générale de notre stratégie.

Par ailleurs, nos toutes premières études sur la disponibilité des ressources dans un système de plus de 100 machines de bureau (cf. trace disponible au Laboratoire de l'accélérateur linéaire à Orsay concernant une activité mesurée sur 15 jours) montrent que l'ensemble des ressources (CPU, disques, réseau) sont sous exploitées. Par exemple, 85% des observations de la charge CPU des machines sont entre 5 et 10% de charge. Cela indique que potentiellement les ressources sont disponibles et que l'on peut préférer utiliser des placements statiques de codes sachant que les ressources seront disponibles plutôt que d'utiliser des mécanisme de placement et d'ordonnancement qui vont s'intéresser dynamiquement à la charge CPU.

La deuxième conclusion générale que nous tirons de l'étude du tri en hétérogène vient des techniques utilisées pour partitionner les données en entrée.

Le regroupement (clustering) est un enjeu fort en fouille de données et certaines instances du problème consistent précisément à construire une partition à k clusters d'une base à n objets, les k clusters doivent optimiser un critère choisi.

Notre approche du partitionnement permet d'appréhender certains algorithmes de regroupement sous un angle différent de celui d'aujourd'hui en particulier lorsque la machine utilisée est hétérogène. À notre connaissance, le partitionnement de données dans un contexte hétérogène (l'architecture matérielle est un cluster) n'a jamais été exploré.

Certains algorithmes de regroupement commencent par affecter chaque objet O à un cluster « au milieu » duquel O est le plus proche. L'algorithme CURE (Clustering Using REpresentatives) [GRS98] fonctionne de la manière suivante : prendre un sous-ensemble de données s, partitionner s en p partitions de taille s/p ; dans chaque partition, créer s/pq clusters ; éliminer les exceptions (points aberrants) et enfin regrouper les clusters partiels. Les premières phases peuvent s'appréhender avec nos approches du partitionnement si elles sont prévues de s'exécuter sur des plates-formes hétérogènes. Nous isolons ainsi des perspectives en matière de fouille de données qui pourront conduire à fusionner nos approches du partitionnement avec les approches actuelles du partitionnement pour le regroupement.

Nos statégies de partitionnement pourraient aussi servir comme outils pour des algorithmes sur les graphes. En particulier, la recherche en visualisation de graphes

s'est penchée sur l'étude des graphes partitionnés de manière hiérarchique comme approche pour la visualisation des graphes de grande taille. De façon similaire, la représentation de la structure d'une plante passe par un découpage hiérarchique en constituants. Les découpages possibles d'une plante sont a priori infinis et la nature des découpages retenus dépend du domaine applicatif dans lequel cette structure est utilisée. Ces représentations nécessitent le développement d'outils algorithmiques adaptés. Sur des clusters hétérogènes, nos stratégies de partitionnement devraient permettre de traiter ces problèmes de manière efficace.

D'autres débouchés potentiels de nos résultats sur le tri en milieu hétérogène sont les suivants. En bases de données, l'opération de jointure peut s'implémenter avec un tri. Nos résultats sont donc des candidats pour l'implémentation de cette opération dans un contexte de bases de données fonctionnant sur des architectures hétérogènes.

Nos tris peuvent remplacer l'implémentation offerte du tri dans les NAS Parallel Benchmark 2.3 qui sont largement employés dans la communauté pour mesurer les performances d'une machine. En effet nos tris sont par exemple indépendants de la taille des données ce qui n'est pas le cas des algorithmes des NAS. Nos tris sont prévus pour fonctionner dans un contexte hétérogène ou homogène, sur disque ou en mémoire. Les NAS ne sont pas conçus pour le cas hétérogène et uniquement pour le cas en mémoire. On ne connait d'ailleurs pas de benchmarks prévus pour capter un niveau d'hétérogénéité des machines. C'est une problématique en soi. Elle est très importante pour le futur.

Enfin, nous pensons valider avec nos tris, la bibliothèque de communications rapides de disques à disques Read[2] qui est développée à Amiens par Gil Utard et Olivier Cozette. Cette interface permet de faire en sorte que l'ensemble des disques d'un cluster de PC et d'Alpha soient vus comme un unique disque. Le cluster disponible à Amiens est donc hétérogène, et si l'on veut montrer que la bibliothèque offre des performances, il faut utiliser des tests qui soient reproductibles d'un cluster héhérogène à un autre. C'est ce que permettent nos tris.

Les perspectives s'inscrivent aussi dans la problématique des *entrées sorties hautes performances*. Nous cherchons à réduire le temps d'accès aux données qui résident sur les disques dans le cadre des clusters et les grilles de calcul.

Pour la problématique du stockage, nous nous intéressons à la représentation des données, à la fouille des données dans des dispositifs de type clusters ou « à grande échelle » (Xtremweb). Il s'agit de construire un dispositif permettant de stocker, par exemple des traces de l'activité de PC. On cherche à construire un dispositif générique, le plus indépendant possible du type concret des données.

Pour exploiter la localité des données avec les disques, des structures de données particulières (B-tree et variantes) ont été conçues depuis 30 ans. Elles permettent d'effectuer un nombre logarithmique d'accès aux disques ce qui est considéré comme

acceptable pour rapport à un nombre d'accès linéaire. Ces structures seront intégrées à notre dispositif.

Nous exploitons la localité des données dans le cadre de la gestion des caches internes des processeurs conduit. Dans le cas du tri en mémoire, l'augmentation du nombre d'instructions à exécuter afin d'exploiter de meilleure façon les unités d'exécution qui fonctionnent en parallèle sur tous les processeurs actuels, procure quand même un gain. Ceci n'est pas intuitif. Nous devons rechercher le meilleur compromis entre le nombre de défauts de cache et le nombre d'instructions exécutées.

Les deux précédents exemples montrent des spécificités, tant au niveau algorithmique qu'au niveau des structures de données, des problèmes d'entrées sorties (disques ou mémoire). Les techniques utilisées ainsi que les structures de données sont spécifiques à un domaine. Par exemple, on n'a jamais utilisé la notion de B-Arbre dans les problèmes de cache. Pourtant on peut par exemple voir l'ensemble des registres du processeur comme définissant la racine du B-arbre, les nœuds internes du B-Arbre au premier niveau étant situés dans le cache L1, et les feuilles du B-Arbre étant situés dans le cache L2.

Les problématiques que nous envisageons en entrées sorties haute performance sont la représentation des données au sens large (scientifiques, commerciales) dans une grille de calcul et la problématique de la fouille, de la collecte des événements liés à l'activité de machines connectées à une grille ou à un système de jachère de calcul comme `Xtremweb`.

Une première problématique vise à construire une infrastructure à large échelle permettant l'échange, l'interrogation de grandes masses de données codées dans des formats utilisés (CDF, HDF...) voire à imaginer de nouveaux codages comme par exemple celui qui est nécessaire pour représenter les informations provenant de l'activité des machines. Dans ce dernier cas, les structures de données proches des B-arbres (R-tree...) sont des candidats et certainement suffisant pour notre étude. En effet chaque évènement reçu ne comporte pas de multiples indexations vers d'autres sructures de données ce qui nécessiterait à priori des techniques beaucoup plus élaborées pour rechercher et modifier.

Une deuxième problématique concerne la fouille de données. Ici, les techniques ou algorithmes connus comme Winepi [MTV97] sont des points de départ mais bien d'autres algorithmes ont été conçus et sont des candidats potentiels comme cSPADE [Zak01], SPIRIT [GRS99]. Chaque algorithme a son propre codage en entrée de l'information relative aux contraintes imposées sur les événements que l'on fouille. Cela influe significativement sur les performances mais nuit à la comparaison des méthodes entre elles.

Notre objectif est de fournir, à partir des événements fréquents découverts dans des traces de comportement de machines formant une grappe ou une grille, des heuristiques à l'ordonnanceur de travaux dans un dispositif de grilles de calcul. Nous avons choisi d'abord de particulariser nos études pour le cas où les données à traiter sont des informations sur l'état des machines participant à la grille. À terme, tous les types de données pourront être fouillés.

Dans les algorithmes de fouille à large échelle l'information n'est pas continuellement présente car une machine déconnectée peut revenir dans le système quleques instants après. Ce fait rajoute nécessairement un surcoût au niveau des algorithmes. Pour le placement de tâches, faut-il envisager de ne pas tenir compte de toutes les machines qui se déconnectent ou bien peut-on quand même compter sur celles qui ne se sont pas déconnectées << trop longtemps >> ? À partir de quel seuil, de quelles informations peut-on décider qu'une machine ou un ensemble de machines va << fausser >> les heuristiques de placement et d'ordonnancement générées?

Les algorithmes de *recherche de motifs* sont des candidats pour fouiller des données provenant de différentes machines, en particulier ceux où l'on peut spécifier des omissions dans les motifs ce qui permet potentiellement de capter les déconnexions. Nous avons utilisé `nrgrep` pour fouiller, sans contrainte temporelle, évènement par évènement, les traces de 110 machines. Cela fonctionne à raison de 45 secondes par événement fouillé, la trace représentant 78Mo et elle était stockée sur un disque IDE d'Athlon 1800. Cependant, l'algorithmique et les outils de recherche de motifs sont difficilement exploitables pour retrouver une information multidimensionnelle. De même, si l'on veut un temps de réponse inférieur à la minute pour fouiller les 15 évènements de la trace, il faudra repenser les algorithmes.

Les travaux que nous menons doivent trouver une place au sein du Grid-Forum qui est une des entités internationales importante regroupant des acteurs travaillant dans le domaine des grilles. Le groupe "Discovery and Event Description" a été approché. Leurs travaux actuels portent sur la définition d'un ensemble d'événements de monitoring et sur leurs représentations afin de se munir d'un vocabulaire commun. J'ai dès à présent proposé d'investir la thématique de la découverte et de la fouille d'évènements dans les grilles sous l'angle des algorithmes et des infrastructures matérielles.

Sixième partie

Liens Internet utiles

La liste de sites WEB suivante concerne les entrées/sorties. Il s'agit de serveurs d'informations généraux. Elle concerne également la fouille des données.

http://www.buyya.com/superstorage: une page de présentation de l'ouvrage "High Performance Mass Storage and Parallel I/O" (Hai Jin, Toni Cortes, Rajulmar Buyya) paru chez IEEE Press. (http://www.ieee.org/press) ;

http://www.ieeeftcc.org: le site de la Task Force IEEE on Cluster Computing ;

http://www.msstc.org/: le site du "Mass Storage Systems Technical Committee" ;

http://www.laria.u-picardie.fr/PALADIN/: groupe de recherche à Amiens (nombreux liens) ;

Evaluer les performances: les liens suivants concernent des tests (débit d'I/O (disques), en communication avec TCP) :

- ftp://ftp.arl.mil/pub/ttcp/ttcp.dpk/ : Tester la couche réseaux TCP ;
- http://www.textuality.com/bonnie/ : tester le système de fichier et les disques ;
- http://www.acnc.com/benchmarks.html : Ce site est un MUST car il concerne Unix et Windows !
- ftp://samba.org/pub/tridge/dbench (Dbench) ;
- http://www.netapp.com/tech_library/3022.html (postmark) ;
- http://www.coker.com.au/bonnie++ (Bonnie++) ;
- http://www.netapp.com/tech_library/3022.html (PostMark: A New File System Benchmark).
- http://www.acnc.com/04_02.html : un site qui recense un grand nombre de benchmarks disques ;
- http://www.namesys.com/benchmarks/mongo/mongo_readme.html (Mongo) ;
- http://etestinglabs.com/benchmarks/netbench/netbench.asp (NetBench is a portable benchmark program that measures how well a file server handles file I/O requests from 32-bit Windows clients). De manière générale, voir la page http://etestinglabs.com/ (Etestinglabs) ou encore http://www.zdnet.com/ (ZDnet).

http://www.laria.u-picardie.fr/~cerin/=paladin/: Algorithmes de tris et leurs implémentations en BSP, MPI ;

http://www.mkdata.dk/click/index.htm: A Complete illustrated Guide to PC Hardware ; Voir aussi http://www.tomshardware.com/

http://www.snia.org: dictionnaire sur la terminologie technique en matière de stockage ;

http://www.raid-advisory.com: Raid Advisory Board (cet organisme teste la conformité des systèmes de stockage vis à vis de spécifications) ;

http://www.minet.net/linux/HOWTO-fr/Software-RAID-HOWTO: le HOW-TO du RAID en logiciel ;

http://www.viarch.org: la technologie VIA (Virtual Architecture Interface) ;

http://www.infiniband.org: InfiniBand (interconnexion de systèmes de disques) ;

http://sourceforge.net/foundry/storage: accès à des systèmes de fichiers (distribués) : CODA, XFS, GFS... ;

http://www.p2pwg.org/: Peer-to-peer working group. On y trouvera des "white papers" sur les systèmes de fichiers ; voir aussi les langages Jxta (`http://www.jxta.org`) de Sun et HailStorm : `http://www.microsoft.com/net/hailstorm.asp` ;

http://www.canarie.ca: une initiative canadienne pour stocker dans le réseau ;

http://oceanstore.cs.berkeley.edu/: The OceanStore Project - Providing Global-Scale Persistent Data - UC Berkeley Computer Science Division ;

http://www.mojonation.net: What is Mojo Nation? Mojo Nation is a revolutionary new peer-driven content distribution technology. While simple data distribution architectures like Napster or Gnutella may be sufficient to allow users to trade mp3 files they are unable to scale up to deliver rich-media content while still taking advantages of the cost savings of peer-to-peer systems. Mojo Nation combines the flexibility of the marketplace with a secure "swarm distribution" mechanism to go far beyond any current filesharing system – providing high-speed downloads that run from multiple peers in parallel. The Mojo Nation technology is an efficient, massively scalable and secure toolkit for distributors and consumers of digital content.

http://www.acm.org/sigkdd: ACM Special Interest Group on Knowledge Discovery in Data and Data Mining ;

http://www.research.microsoft.com/research/datamine/acm-contents.htm: Comm. ACM Special Issue on Data Mining (Vol. 39, No. 11, Nov. 1996) ;

http://www.cs.bham.ac.uk/~anp/TheDataMine.html: The Data Mine ;

http://www.research.microsoft.com/datamine: Data Mining and Knowledge Discovery Journal ;

http://www.informatik.uni-trier.de/~ley/db/groups.html: Database Research Groups ;

http://www.mlnet.org/community/projects-index.html: The Machine Learning network Online Information Service (projects related to knowledge discovery in databases, data mining, machine learning, case-based reasoning, and knowledge acquisition) ;

http://www.isi.edu/nsf: NSF Data Mining Workshop ;

http://www.pitt.edu/~csna/software.html: On-Line Software for Clustering and Multivariate Analysis ;

http://www.almaden.ibm.com/cs/quest: Quest Data Mining Home Page ;

http://lib.stat.cmu.edu/~bill/DMLIST.html: URLs for Data Mining ;

http://www.research.microsoft.com/~fayyad/advances-kdd/fayap.html: Usama Fayyad et al., Advances in Knowledge Discovery and Data Mining, AAAI Press, Menlo Park, Calif., 1996; MIT Press, Cambridge, Mass., 1996 ;

http://www-users.cs.umn.edu/~mjoshi/hpdmtut/index.htm: V. Kumar and M. Joshi, Tutorial on High Performance Data Mining, Dept. of Computer Science, Univ. of Minnesota, 1999;

http://www.bell-labs.com/project/serendip/Talks/tutorial.ps.gz: K. Rastogi and K. Shim, "Tutorial on Scalable Algorithms for Mining Large Databases," Proc. Fifth ACM SIGKDD Int'l Conf. Knowledge Discovery & Data Mining, ACM Press, New York, 1999;

http://www-db.stanford.edu/~ullman/mining/mining.html: les notes de Jeffrey D.Ullman sur la fouille de données.

http://www.acm.org/sigmod/databaseSoftware/: liste de logiciels du domaine public concernant les bases de données.

http://gist.cs.berkeley.edu/: The GiST Indexing Project. The GiST project studies the engineering and mathematics behind content-based indexing for massive amounts of complex content. The project consists of a number of components: The basis of our work is the Generalized Search Tree (GiST), a template indexing structure that allows domain experts (e.g in computer vision, bioinformatics, or remote sensing) to easily customize a database system to index their content.

http://www.emulab.net: "Emulab Classic," a key part of Netbed, is a universally-available time- and space-shared network emulator which achieves new levels of ease of use. Several hundred PCs in racks, combined with secure, user-friendly web-based tools, and driven by ns-compatible scripts or a Java GUI, allow you to remotely configure and control machines and links down to the hardware level. Packet loss, latency, bandwidth, queue sizes– all can be user-defined. Even the OS disk contents can be fully and securely replaced with custom images by any experimenter; Netbed can load ten or a hundred disks in 2.5 minutes total.

http://www.isi.edu/nsnam/ns/: The Network Simulator - Ns. Ns is a discrete event simulator targeted at networking research. Ns provides substantial support for simulation of TCP, routing, and multicast protocols over wired and wireless (local and satellite) networks.

Bibliographie

[Abe01] Aberer (Karl). – P-Grid: A self-organizing access structure for P2P information systems. *Lecture Notes in Computer Science*, vol. 2172, 2001, pp. 179–192.

[ACFS94] Alpern (Bowen), Carter (Larry), Feig (Ephraim) et Selker (T.). – The uniform memory hierarchy model of computation. *Algorithmica*, vol. 12, n° 2/3, août/septembre 1994, pp. 72–109.

[ACGM92] Apostolico (A.), Crochemore (M.), Galil (Z.) et Manber (U.) (édité par). – *Combinatorial Pattern Matching.* – Berlin, 1992.

[ACVW01] Arge (Lars), Chase (Jeff), Vitter (Jeffrey S.) et Wickremesinghe (Rajiv). – Efficient sorting using registers and caches. *Lecture Notes in Computer Science*, vol. 1982, 2001, pp. 51–61.

[ADAT+99] Arpaci-Dusseau (Remzi H.), Anderson (Eric), Treuhaft (Noah), Culler (David E.), Hellerstein (Joseph M.), Patterson (David) et Yelick (Kathy). – Cluster I/O with River: Making the fast case common. *In : Proceedings of the Sixth Workshop on Input/Output in Parallel and Distributed Systems.* pp. 10–22. – Atlanta, GA, mai 1999.

[AFL93] Akl (Selim) et Fava-Lindon (Lorraine). – An optimal implementation of broadcasting with selective reduction. *IEEE Transaction on Parallel and Distributed Systems*, 1993.

[AG89] Akl (Selim) et Guenther (Grant R.). – Broadcasting with selective reduction. *In : Information Processing.* – Elsevier Science Publishers, 1989.

[AG91] Akl (Selim) et Guenther (Grant R.). – Application of broadcasting with selective reduction to the maximal sum subsegment problem. *Int. Journal of high Speed Computing*, 1991.

[Aga89] Agarwal (Anant). – *Performance tradeoffs in multithreaded processors.* – Rapport technique n° 89-566, Cambridge, MA, USA, Massachusetts Institute of Technology, Microsystems Program Office, 1989.

[Aga96] Agarwal (Ramesh C.). – A super scalar sort algorithm for RISC processors. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, vol. 25, n° 2, june 1996, pp. 240–246.

[AHNR95] Andersson (Arne), Hagerup (Torben), Nilsson (Stefan) et Raman (Rajeev). – Sorting in linear time? *In : Proceedings of the Twenty-Seventh Annual ACM Symposium on the Theory of Computing*, pp. 427–436. – Las Vegas, Nevada, 29 mai–1 juin 1995.

[Akl85] Akl (Selim). – *Parallel Sorting Algorithms.* – Academic Press, 1985.

[Akl89] Akl (Selim). – *The Design and Analysis of Parallel Algorithms.* – Prentice Hall, 1989.

[Akl97] Akl (Selim). – *Parallel Computation, Models and Methods.* – Prentice Hall, 1997.

[Amd67] Amdahl (G.). – Validity of a single processor approach to achieving large scale capabilities. *In : Proceedings of the AFIPS Spring joint Computer Conference,*, pp. 463–485.

[Arg95] Arge (Lars). – The buffer tree: A new technique for optimal I/O-algorithms (extended abstract). *In : Algorithms and Data Structures, 4th International Workshop*, éd. par Akl (Selim G.), Dehne (Frank K. H. A.), Sack (Jörg-Rüdiger) et Santoro (Nicola). pp. 334–345. – Kingston, Ontario, Canada, 16–18 août 1995.

[ARM95] A.Shirazi (Behrooz), R.Hurson (Ali) et M.Kavi (Krishna). – *Scheduling and Load Balancing in Parallel and Distributed Systems.* – IEEE CS press, 1995.

[ASV99] Arge (Lars), Samoladas (Vasilis) et Vitter (Jeffrey Scott). – On two-dimensional indexability and optimal range search indexing. *In : Proceedings of the Eighteenth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems: PODS*

1999: Philadelphia, Pennsylvania, May 31–June 2, 1999, éd. par ACM. pp. 346–357. – New York, NY 10036, USA, 1999.

[AV88] Aggarwal (Alok) et Vitter (Jeffrey Scott). – The input/output complexity of sorting and related problems. *Communications of the ACM*, vol. 31, n° 9, septembre 1988, pp. 1116–1127.

[AV99] Abello (James) et Vitter (Jeffrey Scott) (édité par). – *External Memory Algorithms and Visualization.* – Providence, RI, American Mathematical Society Press, 1999, *DIMACS Series in Discrete Mathematics and Theoretical Computer Science.*

[Awe87] Awerbuch (Baruch). – Optimal distributed algorithms for minimum weight spanning tree, counting, leader election and related problems (detailed summary). *In : Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, pp. 230–240. – New York City, 25–27 mai 1987.

[Bar00] Baran (Nicholas). – News and views: Freenet: More anarchy for the Internet?; magnetic properties key to nanoengineering; nanoseconds not fast enough? here come femtoseconds; Caltech leads U.S. field in ACM programming contest; robotic surgeons may make fewer mistakes; free software for designing ICs. *Dr. Dobb's Journal of Software Tools*, vol. 25, n° 6, juin 2000, pp. 18–28.

[Bat68] Batcher (K. E.). – Sorting networks and their applications. *Proceedings of AFIPS Spring Joint Computer Conference*, 1968, pp. 307–314.

[BC97a] Bergogne (Laurent) et Cérin (Christophe). – Application of bsr model of computation for subsegment problems. *In : Proceeding of 4th International Conference on High Performance Computing (HiPC'97)*, pp. 126–131.

[BC97b] Bergogne (Laurent) et Cérin (Christophe). – A new parallel algorithm for the parentheses matching problem. *In : Proceeding Second Aizu International Symposium on Parallel Algorithms/Architectures Synthesis (PAS'97)*, pp. 364–369.

[BCB99] Bal (Henri), Cardelli (Luca) et Belkhouche (Boumediene) (édité par). – *Internet programming languages: ICCL'98 Workshop, held in Chicago, IL, USA, May 13, 1998; proceedings.* – New York, NY, USA, Springer-Verlag Inc., 1999, vii + 143p.

[BDET00] Bolosky (William J.), Douceur (John R.), Ely (David) et Theimer (Marvin). – Feasibility of a serverless distributed file system deployed on an existing set of desktop pcs. *In : Proceedings of the international conference on measurement and modeling of computer systems (SIGMETRIC 2000).* pp. 34–43. – ACM.

[BH82] Borodin (A.) et Hopcroft (J. E.). – Routing, merging, and sorting on parallel models of computation. *In : ACM Symposium on Theory of Computing (STOC '82).* pp. 338–344. – Baltimore, USA, mai 1982.

[BJvOR99] Bonorden (O.), Juurlink (B.), von Otte (I.) et Rieping (I.). – The paderborn university bsp (pub) library - design, implementation and performance. *In : 13th International Parallel Processing Symposium and 10th Symposium on Parallel and Distributed Processing, 12 - 16 April, 1999, San Juan, Puerto Rico, available electronically through IEEE Computer Society.*

[Ble93] Blelloch (Guy E.). – *Prefix Sums and Their Applications, in Synthesis of Parallel Algorithms, Edited by John H. Reif.* – Morgan Kaufmann Publishers, 1993.

[BLM91] Blelloch (G.), Leiserson (C.) et Maggs (B.). – A Comparison of Sorting Algorithms for the Connection Machine CM-2. *In : Proceedings of the ACM Symposium on Parallel Algorithms and Architectures.*

[BM72] Bayer (Rudolf) et McCreight (Edward M.). – Organization and maintenance of large ordered indices. *Acta Informatica*, vol. 1, 1972, pp. 173–189.

[BOV85] Bar-On (I.) et Vishkin (U.). – Optimal generation of a computation tree form. *ACM Transaction of Programming and Language Systems*, vol. 7, n° 2, 1985, pp. 659–663.

[BR93] Blackston (David T.) et Ranade (Abhiram). – SnakeSort: A family of simple optimal randomized sorting algorithms. *In : Proceedings of the 1993 International Conference on Parallel Processing. Volume 3: Algorithms and Applications*, éd. par Hariri, Salim; Berra (P. Bruce). pp. 201–204. – Syracuse, NY, août 1993.

[Buy99a] Buyya (Rajkumar). – *High Performance Cluster Computing, Volume 1: Architectures and Systems.* – Englewood Cliffs, NJ 07632, USA, P T R Prentice-Hall, 1999,????p.

[Buy99b] Buyya (Rajkumar). – *High Performance Cluster Computing, Volume 2: Programming and Applications.* – Englewood Cliffs, NJ 07632, USA, P T R Prentice-Hall, 1999,????p.

[BYP92] Baeza-Yates (Ricardo A.) et Perleberg (Chris H.). – Fast and practical approximate string matching. *In : Combinatorial Pattern Matching, Third Annual Symposium*, éd. par Apostolico (Alberto), Crochemore (Maxime), Galil (Zvi) et Manber (Udi). pp. 185–192. – Tucson, Arizona, 29 avril–1 mai 1992.

[Car97] Cardelli (Luca). – Global computation. *ACM SIGPLAN Notices*, vol. 32, n° 1, janvier 1997, pp. 66–68.

[Car00] Cardelli (Luca). – Mobility and security. *In : Foundations of Secure Computation*, éd. par Bauer (Friedrich L.) et Steinbrüggen (Ralf). pp. 3–37. – IOS Press.

[Cér02] Cérin (Christophe). – An out-of-core sorting algorithm for clusters with processors at different speed. *In : 16th International Parallel and Distributed Processing Symposium (IPDPS), Ft Lauderdale, Florida, USA*, p. Available on CDROM from IEEE Computer Society.

[CG99a] Cardelli (Luca) et Gordon (Andrew D.). – Types for mobile ambients. *In : POPL '99. Proceedings of the 26th ACM SIGPLAN-SIGACT on Principles of programming languages, January 20–22, 1999, San Antonio, TX*, éd. par ACM. pp. 79–92. – New York, NY, USA, 1999.

[CG99b] Cérin (Christophe) et Gaudiot (Jean-Luc). – Algorithms for stable sorting to minimize communications in networks of workstations and their implementations in bsp. *In : Proc. of IWCC'99 (1th International Workshop on Cluster Computing)*. pp. 112–120. – Melbourne, Australia, 2-3décembre 1999.

[CG00a] Cardelli (Luca) et Gordon (Andrew D.). – Anytime, anywhere: Modal logics for mobile ambients. *In : Proceedings of the 27th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POLP-00)*. pp. 365–377. – N.Y., janvier 19–21 2000.

[CG00b] Cardelli (Luca) et Gordon (Andrew D.). – Anytime, anywhere. modal logics for mobile ambients. *In : Proceedings of the 27th ACM Symposium on Principles of Programming Languages*, pp. 365–377.

[CG00c] Cardelli (Luca) et Gordon (Andrew D.). – Mobile ambients. *Theoretical Computer Science*, vol. 240, n° 1, juin 2000, pp. 177–213.

[CG00d] Cérin (Christophe) et Gaudiot (Jean-Luc). – Evaluation of two bsp libraries through parallel sorting on clusters. *In : Proceedings of WCBC'00 (The Second International Workshop on Cluster-Based Computing) in conjuction with ICS'00 (International Conference on Supercomputing, sponsored by ACM/SIGARCH)*, pp. pp 21–26. – Santa Fe, New Mexico, 6mai 2000.

[CG00e] Cérin (Christophe) et Gaudiot (Jean-Luc). – An over-partitioning scheme for parallel sorting on clusters running at different speeds. *In : Cluster 2000. IEEE International Conference on Cluster Computing. Technische Universität Chemnitz, Saxony, Germany. (Poster).*

[CG00f] Cérin (Christophe) et Gaudiot (Jean-Luc). – Parallel sorting algorithms with sampling techniques on clusters with processors running at different speeds. *In : HiPC'2000. 7th International Conference on High Performance Computing. Bangalore, India.* – Springer-Verlag.

[CG01] Cérin (Christophe) et Gaudiot (Jean-Luc). – Benchmarking clusters of workstations through parallel sorting and bsp librairies. *Soumis à Parallel Processing Letter (PPL journal)*, 2001.

[CG02] Cérin (Christophe) et Gaudiot (Jean-Luc). – On a scheme for parallel sorting on heterogeneous clusters. *FGCS (Future Generation Computer Systems*, vol. 18, n° issue 4, 2002. – The special issue is preliminary scheduled for publication in future vol.

[CGG99] Cardelli (Luca), Ghelli (Giorgio) et Gordon (Andrew D.). – Mobility types for mobile ambients. *In : 26th Colloquium on Automata, Languages and Programming (ICALP) (Prague, Czech Republic)*, éd. par Wiederman (Jiří), van Emde Boas (Peter) et Nielsen (Mogens). pp. 230–239. – Springer.

[CGG00] Cardelli (Luca), Ghelli (Giorgio) et Gordon (Andrew D.). – Ambient groups and mobility types. *In : Theoretical Computer Science: Exploring New Frontiers of Theoretical Informatics, Proceedings of the International IFIP Conference TCS 2000 (Sendai, Japan)*, éd. par van Leeuwen (J.), Watanabe (O.), Hagiya (M.), Mosses (P. D.) et Ito (T.). IFIP, pp. 333–347. – Springer.

[CH97] Cormen (Thomas H.) et Hirschl (Melissa). – Early experiences in evaluating the parallel disk model with the ViC* implementation. *Parallel Computing*, vol. 23, n° 4–5, mai 1997, pp. 571–600.

[CH99] Crochemore et Hancart. – Pattern matching in strings. *In : Algorithms and Theory of Computation Handbook, CRC Press, 1999.* – 1999.

[CIS93] Cottingham, Jr. (R. W.), Idury (R. M.) et Schaffer (A. A.). – Faster sequential genetic linkage computations. *American Journal of Human Genetics*, vol. 53, 1993, pp. 252–263.

[CM96] Clark (David R.) et Munro (J. Ian). – Efficient suffix trees on secondary storage (extended abstract). *In : Proceedings of the Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 383–391. – Atlanta, Georgia, 28–30 janvier 1996.

[Col88] Cole (R.). – Parallel merge sort. *SIAM Journal of Computer*, vol. 17, aug. 1988, pp. 770–785.

[CP98] Cérin (C.) et Petit (A.). – Application of algebraic techniques to compute the efficiency measure for multithreaded architecture. *In : Proc. Workshop on Multithreaded Execution, Architecture and Compilation (MTEAC'98), Las Vegas, NV, USA, Jan. 1998*, éd. par Bohm (W.) et Najjar (W.). – Proceedings published as Technical Report CS-98-102, Colorado State University.

[CR99] Cappello (Franck) et Richard (Olivier). – Performance characteristics of a network of commodity multiprocessors for the NAS benchmarks using a hybrid memory model. *In : Proceedings of the 1999 International Conference on Parallel Architectures and Compilation Techniques (PACT '99).* pp. 108–116. – Newport Beach, California, octobre 12–16, 1999.

[CRE99] Cappello (F.), Richard (O.) et Etiemble (D.). – Performance evaluation of two programming models for a cluster of PC biprocessors. *In : Proc. of the Int'l Conf. on Parallel and Distributed Processing Techniques and Applications (PDPTA'99).*

[CRE00] Cappello (Franck), Richard (Olivier) et Etiemble (Daniel). – Investigating the performance of two programming models for clusters of SMP PCs. *In : Proceedings of the Sixth International Symposium on High-Performance Computer Architecture.* IEEE Computer Society TCCA, pp. 349–359. – Toulouse, France, janvier 8–12, 2000.

[CRU02] Cozette (Olivier), Randriamaro (Cyril) et Utard (Gil). – Improving cluster io performance with remote efficient access to distant device. *In : Proceedings of the Workshop on High-Speed local Netwoks to be held in conjunction with the 27th Annual IEEE Conference on Local Computer Networks (LCN).* – Embassy Suites Hotel, Tampa, Florida, novembermai 2002.

[DLM+01] Dongarra (Jack), London (Kevin), Moore (Shirley), Mucci (Phil) et Terpstra (Dan). – Using PAPI for hardware performance monitoring on Linux systems. *In : Linux Clusters: The HPC Revolution, June 25–27, 2001, National Center for Supercomputing Applications (NCSA), University of Illinois, Urbana, IL*, éd. par???? pp.??–?? –????, 2001. Submitted.

[DNS91] DeWitt (David J.), Naughton (Jeffrey F.) et Schneider (Donovan A.). – Parallel sorting on a shared-nothing architecture using probabilistic splitting. *In : Proceedings of the First International Conference on Parallel and Distributed Information Systems*, pp. 280–291.

[Dol00] Dolev (Shlomi). – *Self-Stabilization.* – Cambridge, MA, MIT Press, 2000, 208p. 6 x 938 illus., cloth, Ben-Gurion University of the Negev, Israel.

[Fan93] Fan (Xiaoming). – Analysis of multithreaded architectures: a case study. *In : Proceedings of the 1992 EUROMICRO. Symposium on Microprocessing and Microprogramming.*, pp. 87–90. – Published in Microprocessing & Microprogramming, 37,1993.

[FG99] Ferragina (Paolo) et Grossi (Roberto). – The string B-tree: a new data structure for string search in external memory and its applications. *Journal of the ACM*, vol. 46, n° 2, mars 1999, pp. 236–280.

[FK97] Foster (I.) et Kesselman (C.). – Globus: A metacomputing infrastructure toolkit. *The International Journal of Supercomputer Applications and High Performance Computing*, vol. 11, n° 2, Summer 1997, pp. 115–128.

[FLPR99] Frigo (M.), Leiserson (C. E.), Prokop (H.) et Ramachandran (S.). – Cache-oblivious algorithms. *In : 40th Annual Symposium on Foundations of Computer Science: October 17–19, 1999, New York City, New York,*, éd. par IEEE. pp. 285–297. – 1109 Spring Street, Suite 300, Silver Spring, MD 20910, USA, 1999.

[Fra98] Fraigniaud (Pierre). – Hierarchical broadcast networks. *Information Processing Letters*, vol. 68, n° 6, décembre 1998, pp. 303–305.

[FS99] Ferreira (Afonso) et Schabanel (Nicolas). – A randomized bsp/cgm algorithm for the maximal independent set problem. *Parallel Processing Letters*, vol. 9, n° 3, 1999, pp. 411–422.

[GC99] Gordon (Andrew D.) et Cardelli (Luca). – Equational properties of mobile ambients. *In : Proceedings of the Second International Conference on Foundations of Software Science and Computation Structures (FoSSaCS '99), Held as Part of the Joint European Conferences on Theory and Practice of Software (ETAPS'99), (Amsterdam, The Netherlands, April 1999)*, éd. par Thomas (Wolfgang). pp. 212–226. – Springer.

[GG99] Gannon (Dennis) et Grimshaw (Andrew). – Object-based approaches. *In : The Grid: Blueprint for a New Computing Infrastructure*, éd. par Foster (Ian) et Kesselman (Carl), pp. 205–236. – San Francisco, CA, Morgan Kaufmann, 1999.

[GGB93] Gao (Guang), Gaudiot (Jean-Luc) et Bic (Lubomir). – Dataflow and multithreaded architectures: Guest Editors' introduction. *Journal of Parallel and Distributed Computing*, vol. 18, n° 3, juillet 1993, pp. 271–297.

[Gon81] Gonnet (Gaston H.). – Expected length of the longest probe sequence in hash code searching. *Journal of the Association for Computing Machinery*, vol. 28, n° 2, avril 1981, pp. 289–304.

[Gri00] Grimshaw (Andrew). – Legion — an applications perspective. *In : SC2000: High Performance Networking and Computing. Dallas Convention Center, Dallas, TX, USA, November 4–10, 2000*, éd. par ACM. pp. 99–99. – New York, NY 10036, USA and 1109 Spring Street, Suite 300, Silver Spring, MD 20910, USA, 2000.

[GRS98] Guha (Sudipto), Rastogi (Rajeev) et Shim (Kyuseok). – CURE: an efficient clustering algorithm for large databases. *In : ACM SIGMOD International Conference on Management of Data*, pp. 73–84.

[GRS99] Garofalakis (Minos N.), Rastogi (Rajeev) et Shim (Kyuseok). – SPIRIT: Sequential pattern mining with regular expression constraints. *In : The VLDB Journal*, pp. 223–234.

[GS99] Gerbessiotis (Alexandros V.) et Siniolakis (Constantinos J.). – Efficient deterministic sorting on the bsp model. *Parallel Processing Letters*, vol. 9, n° 1, 1999, pp. 69–79.

[Gut84] Guttman (Antonin). – R-trees: a dynamic index structure for spatial searching. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, vol. 14, n° 2, 1984, pp. 47–57.

[GWF+94] Grimshaw (Andrew S.), Wulf (William A.), French (James C.), Weaver (Alfred C.) et Reynolds, Jr. (Paul F.). – *A Synopsis of the Legion Project.* – Rapport technique n° CS-94-20, Department of Computer Science, University of Virginia, juin 08 1994. Mon, 28 Aug 1995 21:06:39 GMT.

[Hac89] Hac (A.). – A distributed algorithm for performance improvement through file replication, file migration, and process migration. *IEEE Transactions on Software Engineering*, vol. 15, n° 11, novembre 1989, pp. 1459–1470.

[HC83] Huang (J. S.) et Chow (Y. C.). – Parallel Sorting and Data Partitioning by Sampling. *In : IEEE Computer Society's Seventh International Computer Software & Applications Conference (COMPSAC'83)*, pp. 627–631.

[HJ97] Helman (David R.) et JáJá (Joseph). – *Sorting on Clusters of SMPs.* – Technical Report n° CS-TR-3833, University of Maryland, College Park, novembre 1997.

[HJ99] Helman et JáJá. – Sorting on clusters of SMPs. *Informatica: An International Journal of Computing and Informatics*, vol. 23, 1999.

[HJB96] Helman (David R.), JáJá (Joseph) et Bader (David A.). – *A New Deterministic Parallel Sorting Algorithm With an Experimental Evaluation.* – Technical Report n° CS-TR-3670 and UMIACS-TR-96-54, College Park, MD, Institute for Advanced Computer Studies, University of Maryland, août 1996.

[HKP97] Hellerstein (Joseph M.), Koutsoupias (Elias) et Papadimitriou (Christos H.). – On the analysis of indexing schemes. *In : PODS '97. Proceedings of the Sixteenth ACM SIGSIGMOD-SIGART Symposium on Principles of Database Systems, May 12–14, 1997, Tucson, Arizona*, éd. par ACM. pp. 249–256. – New York, NY 10036, USA, 1997.

[HP02] Hennessy (J.L.) et Patterson (D.). – *Computer Architecture, A Quantitative Approach (third edition)*. – Morgan Kauffmann, 2002.

[I+94] Iannucci (Robert A.) et al. (édité par). – *Multithreaded computer architecture: a summary of the state of the art*. – Dordrecht, The Netherlands; Boston, MA, USA, Kluwer Academic Publishers, 1994, *The Kluwer international series in engineering and computer science*, volume SECS 0281, xvi + 400p.

[Jáj92] Jájá (Joseph). – *Introduction to Parallel Algorithms*. – Addison Wesley, 1992.

[JFMS02] Jean-Frédéric Myoupo (David Semé) et Stojmenovic (Ivan). – Optimal bsr solutions to several convex polygon problems. *Journal of Supercomputing*, vol. 21, 2002, pp. 77–90.

[JSPM01] James S. Planck, Alessandro Bassi (Micah Beck D.Martin Swany Rich Wolski) et Moore (Terence). – Managing data storage in the network. *IEEE Internet Magazine*, 2001.

[KAS02] Karl Aberer, Manfred Hauswirth (Magdalena Punceva) et Schmidt (Roman). – Improving data access in p2p systems. *IEEE Internet Computing*, 2002.

[Kim86] Kim (Michelle Y.). – Synchronized disk interleaving. *IEEE Transactions on Computers, Vol.C-35*, no11, novembre 1986.

[Knu98] Knuth (Donald E.). – *Sorting and Searching*. – Reading, MA, USA, Addison-Wesley, 1998, second édition, *The Art of Computer Programming*, volume 3, xiv + 780p.

[KRVV93] Kanellakis (Paris C.), Ramaswamy (Sridhar), Vengroff (Darren E.) et Vitter (Jeffrey S.). – Indexing for data models with constraints and classes (extended abstract). *In : PODS '93. Proceedings of the Twelfth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems: May 25–28, 1993, Washington, DC*, éd. par ACM. pp. 233–243. – New York, NY 10036, USA, 1993.

[KS91] Kistler (James J.) et Satyanarayanan (M.). – Disconnected operation in the coda file system. *SOSP*, vol. 8, n° 2, October 1991, pp. 213–2251.

[Lam78] Lamport (Leslie). – Time, clocks and the ordering of events in a distributed system. *Communications of the ACM*, vol. 21, 7, 1978, pp. 558–565.

[Lee00] Lee (Yui-Wah (Clement)). – *Operation-based Update Propagation in a Mobile File System*. – Thèse de PhD, Department of Computer Science and Engineering, The Chinese University of Hong Kong, January 2000.

[Lei84] Leighton (T.). – Tight bounds on the complexity of parallel sorting. *In : ACM Symposium on Theory of Computing (STOC '84)*. pp. 71–80. – Baltimore, USA, avril 1984.

[Lei92] Leighton (Thomson F.). – *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes*. – Morgan Kauffman Publishers, 1992.

[LL99] LaMarca (Anthony) et Ladner (Richard E.). – The influence of caches on the performance of sorting. *Journal of Algorithms*, vol. 31, n° 1, avril 1999, pp. 66–104.

[LLS+93] Li (X.), Lu (P.), Schaeffer (J.), Shillington (J.), Wong (P. S.) et Shi (H.). – On the versatility of parallel sorting by regular sampling. *Parallel Computing*, vol. 19, octobre 1993, pp. 1079–1103.

[LLS99] Lee (Yui-Wah), Leung (Kwong-Sak) et Satyanarayanan (Mahadev). – Operation-based update propagation in a mobile file system. *In : Usenix Annual Technical Conference*, pp. 43–56.

[LNSS93] Lipton (Richard J.), Naughton (Jeffrey F.), Schneider (Donovan A.) et Seshadri (S.). – Efficient sampling strategies for relational database operations. *Theoretical Computer Science*, vol. 116, n° 1, août 1993, pp. 195–226.

[LPJN97] Larriba-Pey (J.L), Jimenez (D.) et Navarro (J.J). – An analysis of superscalar sorting algorithms on an r8000 processor. *In : Proceedings of the 17th International Conference of*

the Chilean Computer Science Society (SCCC '97), November 12-14, Valpariso, IEEE Computer Society.

[LS94] Li (Hui) et Sevcik (Kenneth C.). – Parallel sorting by overpartitioning. *In : Proceedings of the 6th Annual Symposium on Parallel Algorithms and Architectures.* pp. 46–56. – New York, NY, USA, juin 1994.

[Mac00] Macedonia (Michael). – Entertainment computing: Distributed file sharing: Barbarians at the gates? *Computer*, vol. 33, n° 8, août 2000, pp. 99–101.

[May00] May (John M.). – *Parallel I/O for High Performance Computing.* – Los Altos, CA 94022, USA, Morgan Kaufmann Publishers, 2000, ca. 325p.

[MC93] Michel Cosnard (Denis Trystram). – *Algorithmes et architectures parallèles.* – InterEditions, 1993.

[MDP+00] Milojicic, Douglis, Paindaveine, Wheeler et Zhou. – Process migration. *CSURV: Computing Surveys*, vol. 32, 2000.

[MDW99] Milojicic (Dejan), Douglis (Frederick) et Wheeler (Richard) (édité par). – *Mobility: processes, computers, and agents.* – Reading, MA, USA, Addison-Wesley, 1999, xii + 682p.

[Mil99] Milojičić (Dejan). – Trend wars: Mobile agent applications. *IEEE Concurrency*, vol. 7, n° 3, juillet/septembre 1999, pp. 80–90.

[MJH98] Milvang-Jensen (K.) et Hu (A. J.). – BDDNOW: A parallel BDD package. *Lecture Notes in Computer Science*, vol. 1522, 1998, pp. 501–509.

[MR94] Miller (R.) et Reed (J.L.). – *The Oxford BSP Library : User's Guide.* – Rapport technique, Oxford University Computing Laboratory, 1994.

[MSct] Myoupo (J.F.) et Semé (D.). – Efficients parallel algorithms for the lis and lcs problems using the bsr model with a bounded number of selections. *In : Intern. Symposium, of High Performance Computing, North Carolina.*

[MS99] Myoupo (Jean-Frédéric) et Semé (David). – Time-efficient algorithms for the longest common subsequence and related problems. *Journal of Parallel and Distributed Computing*, vol. 57, 1999, pp. 212–223.

[MTV97] Mannila (Heikki), Toivonen (Hannu) et Verkamo (A. Inkeri). – Discovery of frequent episodes in event sequences. *Data Mining and Knowledge Discovery*, vol. 1, n° 3, 1997, pp. 259–289.

[NBC+94] Nyberg (C.), Barclay (T.), Cvetanovic (Z.), Gray (J.) et Lomet (D.). – AlphaSort: A RISC machine sort. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, vol. 23, n° 2, juin 1994, pp. 233–242.

[NG95] Nemawarkar (Shashank S.) et Gao (Guang R.). – *Latency Tolerance: A Metric for Performance Analysis of Multithreaded Architectures.* – Rapport technique n° ACAPS Memo-85, McGill University, School of computer science, 1995.

[Nil00] Nilsson (Stefan). – The fastest sorting algorithm? *Dr. Dobb's Journal of Software Tools*, vol. 25, n° 4, avril 2000, pp. 38, 40, 42, 44–45.

[NS82] Nassimi (D.) et Sahni (S.). – Parallel permutation and sorting algorithms and a new generalized connection network. *J. ACM*, no29, 1982, pp. 642–667.

[NV95] Nodine (Mark H.) et Vitter (Jeffrey Scott). – Greed sort: Optimal deterministic sorting on parallel disks. *Journal of the ACM*, vol. 42, n° 4, juillet 1995, pp. 919–933.

[OH97] Omondi (Amos R.) et Horne (Michael). – Performance of a context cache for a multithreaded pipeline. *In : Australasian Computer Architecture Conference, Sydney, Australia.* pp. 129–146. – Springer-Verlag, Singapore.

[pat91] *Computer Algorithms: Key search Strategies.* – IEEE Computer Society, technology series, 1991.

[PBKL95] Plank (James S.), Beck (Micah), Kingsley (Gerry) et Li (Kai). – Libckpt: Transparent Checkpointing under Unix. *In : USENIX Technical Conference Proceedings.*

[Pea99] Pearson (Matthew D.). – *Fast Out-of-Core Sorting on Parallel Disk Systems.* – Rapport technique n° PCS-TR99-351, Hanover, NH, Dept. of Computer Science, Dartmouth College, juin 1999.

[PHP+01] Pinto (Helen), Han (Jiawei), Pei (Jian), Wang (Ke), Chen (Qiming) et Dayal (Umeshwar). – Multi-dimensional sequential pattern mining. *In : CIKM*, pp. 81–88.

[Pla89] Plaxton (C. G.). – *Efficient Computation on Sparse Interconnection Networks.* – Rapport technique n° STAN-CS-89-1283, Department of Computer Science, Stanford University, septembre 1989.

[PRR99] Plaxton, Rajaraman et Richa. – Accessing nearby copies of replicated objects in a distributed environment. *MST: Mathematical Systems Theory*, vol. 32, 1999.

[Qui89] Quinn (M.J.). – Analysis and benchmarking of two parallel sorting algorithms: Hyperquicksort and quickmerge. *BIT*, vol. 29, n° 2, 1989, pp. 239–250.

[Raj98] Rajasekaran. – A framework for simple sorting algorithms on parallel disk systems (extended abstract). *In : SPAA: Annual ACM Symposium on Parallel Algorithms and Architectures.*

[R.E86] R.E. Bryant. – Graph-Based Algorithms for Boolean Function Manipulation. *IEEE Transactions on Computers*, vol. C-35, n° 8, août 1986, pp. 677–691.

[RKU00] Ranade (Abhiram), Kothari (Sonal) et Udupa (Raghavendra). – Register efficient mergesorting. *In : HiPC'2000. 7th International Conference on High Performance Computing. Bangalore, India.* – Springer-Verlag.

[RR00] Rahman (N.) et Raman (R.). – *Adapting Radix Sort to the Memory Hierarchy.* – Rapport technique n° TR-00-02, Department of Computer Science, King's College London, October 2000. A preliminary version of the paper appeared in the Proceedings of the 2nd Workshop on Algorithm Engineering and Experiments (ALENEX'00).

[RV83] Reif (John H.) et Valiant (Leslie G.). – A logarithmic time sort for linear size networks. *In : Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing*, pp. 10–16. – Boston, Massachusetts, 25–27 avril 1983.

[RV87] Reif (J. H.) et Valiant (L. G.). – A Logarithmic time Sort for Linear Size Networks. *Journal of the ACM*, vol. 34, n° 1, janvier 1987, pp. 60–76.

[SA94] Stojmenovic (Ivan) et Akl (S.G.). – Multiple critera bsr: An implementation and applications to computational geometry problems. *In : Proc. 27th Hawaii Int'l Conf. System Sciences*, pp. 159–168. – Maui, Hawaii, Janvier 1994.

[SA96] Srikant (Ramakrishnan) et Agrawal (Rakesh). – Mining sequential patterns: Generalizations and performance improvements. *In : Proc. 5th Int. Conf. Extending Database Technology, EDBT*, éd. par Apers (Peter M. G.), Bouzeghoub (Mokrane) et Gardarin (Georges). pp. 3–17. – Springer-Verlag.

[SAG91] Selim Akl (Lorraine Fava-Lindon) et Guenther (Grant R.). – Broadcasting with selective reduction on an optimal pram circuit. *Technique et Science Informatiques*, 1991.

[S.B78] S.B. Akers. – Binary Decision Diagrams. *IEEE Transactions on Computers*, vol. C-27, n° 6, juin 1978.

[SBC91] Saavedra-Barrera (Rafael H.) et Culler (David E.). – *An analytical solution for a Markov chain modeling multithreaded execution.* – Report n° UCB/CSD 91/623, Berkeley, CA, USA, University of California, Berkeley, Computer Science Division, avril 1991.

[SBCvE90] Saavedra-Barrera (Rafael H.), Culler (David E.) et von Eicken (Thorsten). – Analysis of multithreaded architectures for parallel computing. *In : SPAA '90. 2nd Annual ACM Symposium on Parallel Algorithms and Architectures.*, pp. 169–178. – Also as Tech report UCB-CSD-90-569, University of California Berkeley, Department of Computer Science.

[S.C00] S.Chatterjee (S.Sen). – Towards a theory of cache efficient algorithms. *In : Proceedings of the Eleventh Annual ACM/SIAM Symposium on Discrete Algorithms (SODA'00)*, pp. 829 – 838. – San Francisco, California, United States, 2000.

[SE94] Srivastava (A.) et Eustace (A.). – Atom: a system for building customized program analysis tools. *In : In Proceedings of the SIGPLAN '94 Conference on Programming Language Design and Implementation, 1994.*, pp. 196–205.

[Sen96] Sentovich (E. M.). – A brief study of BDD package performance. *Lecture Notes in Computer Science*, vol. 1166, 1996, pp. 389–??

[SGM86] Salem (Kenneth) et Garcia-Molina (Hector). – Disk Striping. *In : Proceedings of the 2nd International Conference on Data Engineering.* ACM, pp. 336–342.

[Sha50] Shannon (C. E.). – Memory requirements in a telephone exchange. *Bell Syst. Tech. J.*, 1950.

[SK96] Schikuta (Erich) et Kirkovits (Peter). – Analysis and evaluation of sorting for parallel database systems. *In : In Proc. Euromicro 96, Workshop on Parallel and Distributed Processing, Braga, Portugal, IEEE Computer Society Press*, pp. 258–265.

[SK97] Sibeyn (J.F.) et Kaufmann (M.). – *BSP-Like External-Memory Computation.* – Technical Report n° MPI-I-97-1001, Max-Planck Institut für Informatik, Saarbrücken, Germany, 1997.

[SKM+93] Satyanarayanan (M.), Kistler (James J.), Mummert (Lily B.), Ebling (Maria R.), Kumar (Puneet) et Lu (Qi). – Experience with disconnected operation in a mobile environment. *In : Proceedings of the USENIX Mobile and Location-Independent Computing Symposium: August 2–3, 1993, Cambridge, Massachusetts, USA*, éd. par USENIX Association. pp. 11–28. – Berkeley, CA, USA, août 1993.

[SMH94] Saavedra (R. H.), Mao (Weihua H.) et Hwang (Kai). – Performance and optimization of data prefetching strategies in scalable multiprocessors. *Journal of Parallel and Distributed Computing*, vol. 22, n° 3, septembre 1994, pp. 427–448.

[SN92] Seshadri (S.) et Naughton (Jeffrey F.). – Sampling issues in parallel database systems. *In : Proceedings of Advances in Database Technology (EDBT '92)*, éd. par Pirotte (Alain), Delobel (Claude) et Gottlob (Georg). pp. 328–343. – Berlin, Germany, mars 1992.

[SNM+00] Suzumura (T.), Nakagawa (T.), Matsuoka (S.), Nakada (H.) et Sekiguchi (S.). – Are global computing systems useful? comparison of client-server global computing systems ninf, NetSolve versus CORBA. *In : Proceedings of the 14th International Conference on Parallel and Distributed Processing Symposium (IPDPS-00).* pp. 547–558. – Los Alamitos, mai 1–5 2000.

[SNS+97] Sato (M.), Nakada (H.), Sekiguchi (S.), Matsuoka (S.), Nagashima (U.) et Takagi (H.). – Ninf: A network based information library for global world-wide computing infrastructure. *In : High-Performance Computing and Networking (HPCN'97 Europe).* – Vienna, Springer Verlag, avril 1997.

[Squ94] Squillante (Mark S.). – *Analytic modeling of processor utilization in multithreaded processor architectures.* – Research report n° RC 19543 (84999), Yorktown Heights, NY, USA, IBM T. J. Watson Research Center, avril 1994.

[SRK01] Sean Rhea, Chris Wells (Patrick Eaton Denis Geels Ben Zhao Hakim Weatherspoon) et Kubiatowicz (Johan). – Maintenance-free global data storage. *IEEE Internet Magazine*, 2001.

[SS92] Shi (Hanmao) et Schaeffer (Jonathan). – Parallel sorting by regular sampling. *Journal of Parallel and Distributed Computing*, vol. 14, n° 4, 1992, pp. 361–372.

[Sto96] Stojmenovic (Ivan). – Constant time bsr solutions to parenthesis matching, tree decoding, and tree reconstruction from its traversals. *IEEE Transaction of Parallel and Distributed Systems*, vol. 7, n° 2, February 1996.

[TB93] Tridgell (A.) et Brent (R.P.). – *An Implementation of a General-purpose Parallel Sorting Algorithm.* – Rapport technique n° TR-CS-93-01, Computer Science Laboratory, Australian National University, février 1993.

[UW02] Ullman (Jeffrey D.) et Widom (Jennifer D.). – *First Course in Database Systems, A, 2/e.* – Prentice Hall, 2002.

[Val90a] Valiant (L.G.). – A bridging model for parallel computation. *Communications of the ACM*, no33, August 1990, pp. 103–111.

[Val90b] Valiant (L.G.). – A bridging model for parallel computation. *Communiations of the ACM*, vol. 33, n° 8, août 1990, pp. 103–111.

[Vis93] Vishkin (Uzi). – *Advanced Parallel Prefix Sums, List Ranking and Connectivity, in Synthesis of Parallel Algorithms.* – Edited by John H. Reif, Morgan Kaufmann Publishers, 1993.

[VS94a] Vitter (Jeffrey Scott) et Shriver (Elizabeth A. M.). – Algorithms for parallel memory I: Two-level memories. *Algorithmica*, vol. 12, n° 2/3, août and septembre 1994, pp. 110–147.

[VS94b] Vitter (Jeffrey Scott) et Shriver (Elizabeth A. M.). – Algorithms for parallel memory II: Hierarchical multilevel memories. *Algorithmica*, vol. 12, n° 2/3, août and septembre 1994, pp. 148–169.

[VVK96] Vladimir Vlassov (Lars-Erik Thorelli) et Kraynikov (Alexander). – *A Queuing Model of Multithreading: A Case Study.* – Research Report n° TRITA-IT R 96:18, Dept. of Teleinformatics, Royal Inst. of Technology, Stockholm, 1996.

[Wa00] White et al. – *An Integrated Experimental Environment for Distributed Systems and Networks (full report).* – Rapport technique, University of Utah, may 2000. Revised version to appear at OSDI 2002, December 2002.

[WC00a] Welsh (Matt) et Culler (David). – Jaguar: enabling efficient communication and I/O in Java. *Concurrency: Practice and Experience*, vol. 12, n° 7, mai 2000, pp. 519–538.

[WC00b] Welsh (Matt) et Culler (David E.). – Achieving robust, scalable cluster i/o in java. *In: Proceedings of 5th International Workshop, LCR 2000 (Languages, Compilers, and Run-Time Systems for Scalable Computer)*. pp. 16–31. – Springer Verlag, LNCS 1915.

[WGAP01] Wu (Jiesheng), Gulati (Abhishek), Abali (Bulent) et Panda (Dhabaleswar K.). – *Design of An InfiniBand Emulator over Myrinet: Challenges, Implementation and Performance Evaluation.* – Research report n° OSU-CISRC-2/01-TR03, Ohio State University, février 2001.

[WT98] Watts (J.) et Taylor (S.). – A practical approach to dynamic load balancing. *IEEE Transaction on Parallel and Distributed Systems*, vol. 9, n° 3, 1998, pp. 235–248.

[Zak01] Zaki (Mohammed Javeed). – SPADE: An efficient algorithm for mining frequent sequences. *Machine Learning*, vol. 42, n° 1/2, 2001, pp. 31–60.

[Zak25] Zaki (Mohamed J.). – Parallel and distributed association mining: A survey. *IEEE Concurrency*, vol. Vol. 7, No. 4, October-December 1999, pp. 14-25.

[ZZWD93] Zhou (Songnian), Zheng (Xiaohu), Wang (Jingwen) et Delisle (Pierre). – Utopia: a load sharing facility for large, heterogeneous distributed computer systems. *Software Practice and Experience*, vol. 23, n° 12, décembre 1993, pp. 1305–1336.