INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Projet CAPS

Compilation, architectures parallèles et systèmes

Rennes

THÈME 1A

Table des matières

1 Composition de l'équipe **3**

2 Présentation et objectifs généraux **4**
2.1 Panorama 4
2.2 Architectures de processeurs 5
2.3 Environnements de développement pour architectures hautes performances . . . 6
2.4 Gestion de ressources au sein de calculateurs parallèles 7
2.5 Technologies pour le Metacomputing 8

3 Fondements scientifiques **9**
3.1 Panorama 9
3.2 L'exécution spéculative 9
3.3 Simulation de processeurs et collecte de traces 10
3.4 Compilation pour architectures hautes performances 12
3.5 Mémoire virtuellement partagée 15

4 Domaines d'applications **17**
4.1 Panorama 17
4.2 Couplage de codes de simulation 17

5 Logiciels **17**
5.1 Panorama 17
5.2 Salto : un environnement de transformations pour les langages d'assemblages (cf. 2.3) 17
5.3 Mome : une mémoire virtuelle partagée pour des langages parallèles (cf. 6.3) . . 18
5.4 PaCo : objet Corba parallèle (cf. 6.4) 19
5.5 Do! : Générateur automatique de code Java réparti 20

6 Résultats nouveaux **20**
6.1 Architectures de processeurs (cf. 2.2) 20
6.2 Environnement pour architectures hautes performances (cf. 2.3) 23
6.3 Gestion de ressources au sein de calculateurs parallèles (cf. 6.3) 26
6.4 Technologies pour le METACOMPUTING (cf. 2.5) 29

7 Contrats industriels (nationaux, européens et internationaux) **32**
7.1 Projet OMI SPEAR 2, convention $n°$ 197C8499931398005 (12/97-07/98) 32
7.2 Projet Esprit LTR Oceans (cf. 2.3, 6.2), convention $n°$ 196C5350031308006 (10/96-10/99) 32
7.3 Projet Esprit R & D Fits : Fortran Integrated Tool Set (cf. 2.3, 6.2), convention $n°$ 197C7720031308202 (6/97-6/99) 33
7.4 Projet Esprit R & D Pacha (cf. 2.5), convention $n°$ 96C318003130801 (5/96-12/98) 33
7.5 Projet Esprit TTN Pro HPC, convention $n°$ 97C4700031308005 (3/97-3/99) . . 33

7.6 Working Group Esprit EuroTools (WG 27141), convention n° 98C1790031308005 (04/98-04/00) . . . 34
7.7 Projet Esprit Jaco3, convention n° 98C3760031308005 (11/98-04/01) . . . 34
7.8 Contrat Aérospatiale Matra, convention n° 98C4460031308011 (02/99-07/99) . . . 35
7.9 Nec (cf. 2.3, 2.5, 6.2) . . . 35
7.10 Projet SoftIT, convention n° 198A4110000MPR012 (01/98-06/99) . . . 35
7.11 System level Methods and Tools Medea n° 199C9010031308011 (1999-2000) . . . 36
7.12 Etude relative à la parallèlisation de modèle dynamique océanique, Epshom n° 198C5760031308061 (1998-1999) . . . 36
7.13 modélisation d'architecture de microprocesseurs multithreadés pour le multimédia, CEA, (1999-2002) . . . 36

8 Actions régionales, nationales et internationales 37
8.1 Actions régionales . . . 37
8.2 Actions nationales . . . 37
8.3 Actions financées par la Commission Européenne . . . 37
8.4 Relations bilatérales internationales . . . 37

9 Diffusion de résultats 37
9.1 Animation de la communauté scientifique . . . 37
9.2 Enseignement universitaire . . . 38
9.3 Autres enseignements . . . 38
9.4 Participation à des colloques, séminaires, invitations . . . 38
9.5 Divers . . . 39

10 Bibliographie 39

1 Composition de l'équipe

Responsable scientifique

André Seznec [DR Inria, en disponibilité chez DEC-COMPAQ depuis le 1er mars 1999]

Assistante

Huguette Béchu [TR Inria]

Personnel Inria

Yvon Jégou [CR]
Christine Morin [CR]
Pierre Michaud [CR, depuis le 1er octobre]
Thierry Priol [DR]

Personnel Upresa 6074

François Bodin [professeur, université de Rennes 1]
Jacques Lenfant [professeur, université de Rennes 1]
Jean-Louis Pazat [maitre de conférence, Insa]
Pascale Launay [Ater, université de Rennes I]
Dan Truong [Ater, université de Rennes I]

Ingénieurs experts Inria

Stéphane Gouache
Stéphane Chauveau [jusqu'au 1er avril]
Yann Mével [usqu'au 1er mai]

Chercheurs doctorants

Stéphane Écolivet [bourse MENESR]
Thierry Lafage [bourse Inria-région]
Renaud Lottiaux [bourse MENESR]
David Mentré [bourse Inria]
Christophe René [bourse MENESR]
Antoine Monsifrot [bourse MENESR, depuis le 1er octobre]
Ronan Amicel [bourse Inria, depuis le 1er octobre]
Jonathan Perret [bourse Inria, depuis le 1er novembre]
Romain Dolbeau [AMN, depuis le 1er octobre]

Autres personnels

Tsunehiko Kamachi [chercheur Nec invité]

2 Présentation et objectifs généraux

2.1 Panorama

Résumé :

Le projet Caps a pour objectif d'étudier les concepts à la fois matériels et logiciels entrant dans la conception des calculateurs hautes performances.

Les performances théoriques des calculateurs croissent régulièrement. Cependant cet accroissement des performances de crête se poursuit au prix d'une complexité matérielle de plus en plus élevée. Ainsi, de nombreux niveaux de parallélisme sont présents sur le matériel, et l'obtention de performances élevées nécessite l'exploitation simultanée de tous ces niveaux par les applications. La mise au point des applications pour la performance devient de plus en plus une activité de haute technologie.

Les recherches menées au sein du projet Caps visent à exploiter de manière efficace les différents niveaux de parallélisme présents dans les applications et sur les architectures tout en masquant la complexité des matériels et systèmes à l'utilisateur.

Nos recherches en architecture de processeur s'appuient sur une activité de veille technologique diffusée depuis 1991. Ces recherches visent à améliorer le comportement de la hiérarchie mémoire et augmenter le parallélisme d'instructions présenté au matériel. Ainsi, de nouvelles structures matérielles d'antémémoires sont étudiées afin de réduire les pénalités engendrées par les accès à la mémoire principale. D'autre part, nous étudions de nouveaux mécanismes de prédictions de branchements afin d'augmenter le parallélisme d'instructions soumis au matériel par **un** *processus. Cependant, nous explorons aussi l'approche orthogonale, dite* **multiflot simultané** *où les instructions présentées aux unités d'exécution sont issues de* **plusieurs** *processus différents.*

L'obtention de performances sur un processeur passe aussi par une maitrise logicielle du parallélisme d'instructions et de la hiérarchie mémoire. C'est pourquoi, nous étudions des techniques logicielles d'optimisation de code visant à détecter et à exploiter la localité des accès à la mémoire. Des techniques de réordonnancement de codes (pipeline logiciel, déroulage de boucles,...) sont aussi développées afin de soumettre un parallélisme d'instructions important au matériel. Ces techniques sont appliquées aussi bien aux processeurs généraux qu'aux processeurs enfouis (multimédia par exemple).

Nos recherches en logiciel visent aussi à obtenir des performances sur les architectures multiprocesseurs aussi bien à mémoire partagée que physiquement distribuée. Nos travaux au niveau système visent à faciliter la programmation des systèmes parallèles et distribués (grappes de calculateurs) pour la simulation numérique. Nous étudions notamment le concept de mémoire virtuelle partagée qui offre la vision d'une mémoire globale à accès uniforme partagée. Une partie de plus en plus importante de nos travaux concernent les environnements de programmation pour la simulation numérique distribuée (metacomputing). L'objectif est de simpli-

fier la conception d'applications de simulation numérique qui nécessitent l'utilisation de plusieurs ressources géographiquement distribuées sur un réseau (intranet ou internet).

Afin de masquer à l'utilisateur la complexité logicielle de l'optimisation pour la performance, il convient de lui fournir des outils adaptés pour cette optimisation dans des environnements de développement. Une partie importante de notre activité est consacrée au développement de tels environnements.

2.2 Architectures de processeurs

Mots clés : microprocesseur, Risc, antémémoire, prédiction de branchement, multiflot simultané.

Résumé : *Les progrès technologiques permettent une plus grande densité d'intégration et une plus grande fréquence de fonctionnement des composants pour les processeurs. Ainsi, il est aujourd'hui possible d'intégrer sur un même composant une dizaine d'unités fonctionnelles et une grande antémémoire fonctionnant à une fréquence de l'ordre de 500* Mhz.

Cependant ces progrès ne se traduisent pas linéairement en un gain de performances. En effet, le temps de cycle des processeurs décroît plus rapidement que les temps d'accès à la mémoire principale, ce qui rend la performance effective du processeur de plus en plus dépendante du comportement de sa hiérarchie mémoire. De même, le parallélisme d'instructions limité des programmes (dépendences de données et contrôle) réduit les gains liés à l'exécution superscalaire.

Les actions de recherches que nous menons portent sur la structure et les optimisations matérielles et logicielles des hiérarchies mémoire, en particulier antémémoires, sur les mécanismes de lancement des instructions, en particulier prédiction de branchement ainsi que sur les structures de processeur multiflot simultané. Ces actions de recherche s'appuient sur une veille technologique sur les microprocesseurs menée depuis 1991.

L'évolution des microprocesseurs est extrêmement rapide. Depuis début 1991, nous menons une activité de veille technologique et de diffusion d'informations sur les microprocesseurs. Sept rapports détaillés comparant les architectures de processeurs ont été diffusés à ce jour. Cette activité de veille technologique permet d'orienter les recherches du projet en architecture de processeurs.

La différence entre temps d'accès à l'antémémoire sur le composant et temps d'accès à la mémoire principale tend à croître. Il est donc de plus en plus important d'optimiser le comportement des antémémoires. Le taux de succès lors des accès à une antémémoire dépend de nombreux facteurs liés à son organisation matérielle et à l'application. Nos recherches portent à la fois sur l'étude de structures d'antémémoires "skewed-associative" [6] ainsi que sur les techniques logicielles de détection et d'exploitation de la localité [7] et d'optimisation du placement de données.

L'allongement des pipelines et l'exécution superscalaire font que le délai entre le chargement d'une instruction et son exécution correspond aujourd'hui à l'exécution de plusieurs dizaines d'instructions. Or, toute instruction de branchement rompt le flot de contrôle et devrait donc en principe arrêter le séquencement. Afin d'éviter un tel arrêt, des mécanismes d'anticipation appelés prédicteurs de branchement sont mis en œuvre dans les processeurs d'aujourd'hui. D'autre part, avec l'avènement de l'exécution dans le désordre et de l'exécution spéculative très agressive, le chargement en parallèle d'un seul bloc de base (c'est-à-dire l'anticipation d'un seul branchement par cycle) apparaît comme trop limité. Il est maintenant nécessaire de charger plusieurs blocs de base par cycle. Nos travaux dans ce domaine visent à améliorer la précision de la prédiction de branchement ainsi qu'à augmenter le nombre d'instructions chargées par cycle [15].

Si jusqu'à présent, la recherche de la performance ultime sur un seul processus a guidé l'industrie du microprocesseur, l'énorme potentiel d'intégration aujourd'hui disponible permet d'envisager que, d'ici à quelques années, plusieurs processus s'exécutent en parallèle sur le même composant. Parmi les solutions exploitant ces nouvelles données technologiques, le *multiflot simultané* [TEL95], semble l'une des méthodes les plus prometteuses. Le *multiflot simultané* est basé sur l'exécution de plusieurs flots d'instructions indépendants ou issus d'une application parallèle sur un processeur superscalaire. Nous étudions les implications de l'utilisation du multiflot simultané dans le processeur.

2.3 Environnements de développement pour architectures hautes performances

Mots clés : Matlab, programmation parallèle, parallélisation automatique, portage d'applications, optimisation.

Résumé : *L'obtention de performances sur les architectures hautes performances nécessitent des outils logiciels adaptés qui cachent à l'utilisateur la complexité des matériels et des systèmes.*

Les actions de recherches que nous menons visent à fournir aux utilisateurs de calculateurs hautes performances des outils tels que compilateur, aide au portage, optimiseur pour permettre des développements et/ou portages d'applications hautes performances.

Ainsi, nous développons TSF*, un outil d'aide au portage de codes Fortran sur architectures hautes performances et Salto un environnement de manipulation de langage d'assemblage.*

Différentes approches sont utilisées par les utilisateurs pour développer les applications hautes performances. Nous développons différents environnements pour quelques-unes de ces approches.

[TEL95] D. Tullsen, S. Eggers, H. Levy, « Simultaneous multithreading : maximising on-chip parallelism », *in : 22nd Annual International Symposium on Computer Architecture*, p. 392–403, juin 1995.

L'arrivée des stations de travail multiprocesseur a banalisé l'usage des machines parallèles. De nombreux progiciels Fortran gagneraient à être portés sur ces architectures afin d'atteindre une vitesse élevée de calcul à un coût raisonnable. Le portage (parallélisation et amélioration de performance) d'applications sur machines hautes performances est une activité techniquement difficile faisant appel à beaucoup de savoir-faire. Aussi, dans l'outil de portage TSF que nous développons au dessus de l'environnement Foresys de Simulog, nous accélérons le portage grâce à l'utilisation conjointe de techniques issues de deux domaines : la parallélisation automatique et le raisonnement à partir de cas.

Les systèmes embarqués et à haute performance font un usage de plus en plus fréquent des technologies VLIW, DSP ou Risc. L'optimisation, l'étude de performance, l'analyse des codes nécessitent la mise en œuvre de méthodes adaptées aux spécificités de ces architectures, en particulier à l'exploitation du parallélisme d'instructions. Toutefois, l'écriture de tels outils requiert du programmeur un investissement significatif qui a peu de chance d'être réutilisable, compte tenu de la diminution de la durée de vie des composants. Le système Salto que nous avons développé propose un environnement de manipulation de programmes en langage assembleur. Il a pour objectif de permettre l'écriture rapide de tous les outils manipulant du langage assembleur : ordonnanceur, optimiseur de code, simulateur,...

2.4 Gestion de ressources au sein de calculateurs parallèles

Mots clés : mémoire virtuelle partagée, systèmes de gestion de fichiers parallèles.

Résumé : *La bonne gestion des ressources au sein d'une grappe de calculateurs est essentielle pour l'obtention de bonnes performances. Nous étudions les concepts qui permettent de gérer la mémoire (mémoire virtuelle partagée), les disques (système de gestion de fichiers parallèles) et les processeurs (migration de processus). Des études sont en cours sur différentes architectures: système parallèle (Nec Cenju-4) et grappes de* PC. *Nous nous intéressons notamment à l'intégration de ces différents concepts au sein d'un système distribué pour grappes de* PC.

Dans la programmation des calculateurs parallèles, l'obtention de bonnes performances est conditionnée par une utilisation judicieuse des ressources disponibles (processeur, mémoire, disque). L'objectif de nos recherches est d'étudier les différents concepts qui permettent de mieux gérer les ressources d'un système parallèle à mémoire distribuée : machine parallèle ou grappe de PC. Nous nous intéressons plus particulièrement à cette dernière catégorie. Les grappes de PC sont des calculateurs parallèles construits à partir de "composants sur étagères" (PC, réseaux haut-débit). Par rapport aux machines parallèles, leur intérêt est essentiellement économique. Cependant, ce type de machine s'avère plus complexe à utiliser car chaque PC fonctionne de façon indépendante. Nous pensons que le succès de ce type de machine sera fortement lié à la capacité d'en gérer globalement les ressources.

Notre objectif est donc d'étudier les différents concepts permettant de mieux gérer les ressources d'une grappe: mémoire virtuellement partagée (mémoire), systèmes de gestion de fichiers parallèles (disque) et migration de processus (processeur). L'originalité de notre approche est d'aborder de façon globale la gestion de ces ressources. Un mécanisme de mémoire

virtuelle partagée doit, par exemple, pouvoir être couplé de façon efficace avec un système de gestion de fichiers parallèles. Ceci afin d'autoriser la projection de fichiers en mémoire avec des performances "scalables". La migration de processus doit être également couplée avec la gestion mémoire afin de permettre la migration efficace des données du processus vers un processeur de la grappe.

Les contraintes que nous nous fixons sont bien entendues d'obtenir la haute-performance. Cependant, pour les grappes de PC, nous ajoutons également la haute disponibilité afin de tolérer la défaillance d'une ou plusieurs machines au sein de la grappe. A la différence des machines parallèles, les grappes de PC sont souvent moins robustes compte tenu de la qualité des composants et de l'aspect faiblement couplé de ce type d'architecture.

2.5 Technologies pour le Metacomputing

Mots clés : programmation parallèle, Now, Metacomputing, Corba.

Résumé : *L'accroissement rapide de la performance des calculateurs et des réseaux va permettre d'envisager des techniques de simulation numérique par couplage de codes. Il s'agit non plus de simuler un seul aspect physique d'un problème mais de plusieurs phénomènes physiques simultanément. Ainsi, par exemple, on pourra simuler le comportement d'un satellite dans l'espace en y intégrant la dynamique, la thermique, la déformation de structure et l'optique. L'enjeu est de réduire les temps de conception d'objets manufacturés. Pour permettre cette simulation multi-physiques, il sera nécessaire d'utiliser un ensemble de ressources de calcul disponibles sur un réseau afin de permettre l'exécution simultanée, et de façon coordonnée, de plusieurs codes de simulation. L'objectif de ce thème de recherche est de concevoir des technologies qui permettent la construction d'environnements logiciels qui permettent de supporter efficacement l'exécution d'applications de simulation numérique distribuée.*

Au cours des dix dernières années, les performances des calculateurs se sont fortement accrues grâce à l'évolution rapide des technologies des processeurs. Parallèlement, dans le domaine de la simulation numérique, les besoins en puissance de traitement ont également augmenté car les utilisateurs souhaitent simuler des phénomènes physiques de plus en plus complexes. Cette complexité est due à l'utilisation simultanée de plusieurs codes numériques prenant en compte l'interaction de plusieurs phénomènes physiques (interaction fluide-structure, thermique-optique, etc.). Pour obtenir des temps de calcul raisonnables, il est nécessaire d'utiliser simultanément plusieurs ressources de calcul ayant une forte hétérogénéité, ces ressources étant connectées par des réseaux d'interconnexion à très haut débit. Cette approche est connue sous le nom de «Metacomputing». L'hétérogénéité de ces ressources s'exprime par des modèles d'exécution, de communication et de programmation très différents. Il est donc nécessaire de concevoir des environnements de programmation permettant de masquer cette hétérogénéité à l'utilisateur. Les applications d'un tel environnement sont variées : travail coopératif pour la simulation, couplage de codes de simulation, etc.

L'objectif de nos recherches est de concevoir des environnements de programmation à la fois pour des réseaux de PC et des réseaux de ressources de calcul hétérogènes. L'originalité

de notre approche est de marier deux types de programmation: la programmation des calculateurs parallèles et la programmation des systèmes distribués. Notre approche s'appuie sur l'utilisation d'une technologie «middleware», comme Corba (Common Object Request Broker Architecture), qui permet la conception d'applications distribuées en utilisant une approche client/serveur. Notre objectif est d'étudier les différents mécanismes logiciels nécessaires à la conception d'un environnement de programmation par composants logiciels Corba pour le calcul de haute performance pour des réseaux de stations de travail et des plateformes de Metacomputing. Il s'agit notamment d'étendre des "middleware" existants (Corba) afin de supporter efficacement l'exécution parallèle de composants logiciels. Nous étudions également des protocoles de communication haute-performance au sein de courtier d'objets (ORB Corba).

3 Fondements scientifiques

3.1 Panorama

Résumé : *Les activités de recherche du projet Caps s'appuient sur des bases issues de plusieurs communautés scientifique : architecture, compilation et système. Nous avons choisi de présenter ici brièvement quelques fondements de nos recherches : les principes et défis liés à l'exécution spéculative, le problème de la simulation de processeurs et de la collecte de traces, un aperçu des techniques de transformation de programmes, et enfin les principes de la mémoire virtuelle partagée.*

3.2 L'exécution spéculative

Mots clés : prédiction de branchement, exécution spéculative.

Résumé : *Les pipelines d'exécution des processeurs superscalaires sont de plus en plus longs. Afin de limiter les ruptures de charge dans les pipelines dues aux instructions de branchement, des mécanismes de prédiction de branchement sont mis en œuvre dans les processeurs, et les instructions prédites sont exécutées spéculativement.*

Pour atteindre un niveau de performance élevé sur les processeurs superscalaires de large degré qui devraient apparaitre vers l'an 2000, il est nécessaire de charger des instructions non-contiguës en mémoire, mais aussi de rompre les chaînes de dépendances entre instructions par la prédiction de valeurs.

Les pipelines d'exécution des processeurs sont de plus en plus longs : 12 cycles sur l'Intel Pentium II. Les processeurs sont capables d'exécuter plusieurs instructions par cycle. Le séquencement des instructions devrait être interrompu à chaque instruction de branchement en attendant le calcul effectif de la condition et/ou de la cible : hors sur beaucoup d'applications, plus d'une instruction sur 5 ou 6 est un branchement.

Sur tous les processeurs superscalaires actuels, des mécanismes de prédictions de branchement sont mis en œuvre pour continuer le séquencement *spéculatif* des instructions après un

branchement sans attendre sa résolution : la cible et la direction du branchement sont prédites. En cas de mauvaise prédiction, les instructions séquencées (et parfois même déjà exécutées) doivent être annulées et le séquencement est repris sur le chemin réellement utilisé par l'application. Étant donné la très lourde pénalité payée en cas de mauvaise prédiction de branchement, la performance effective d'un processeur dépend de la précision de la prédiction. Des schémas de prédiction de plus en plus sophistiqués sont donc mis en œuvre dans les processeurs. Parmi les informations utilisées pour prédire un branchement, on peut citer l'adresse du branchement, l'historique des derniers branchements exécutés, l'historique des derniers passages dans ce branchement [Yeh93],... Cependant les recherches continuent dans plusieurs directions, parmi lesquelles on peut citer, la réduction des interférences sur les tables de prédictions de branchement [12] et la prédiction des branchements indirects [CHP97].

Les processeurs actuels exécutent les instructions de manière spéculative et dans le désordre. La génération actuelle de processeurs peut exécuter jusqu'à 4, parfois 6, instructions par cycle. Il est d'ores et déjà possible d'implémenter des processeurs pouvant lancer 10 voire 16 instructions par cycle. Cependant l'obtention de telles performances ne peut pas être envisagée en utilisant les mécanismes de séquencement actuels : seules des instructions consécutives sont chargées, alors que sur beaucoup d'applications, plus d'une instruction sur 5 ou 6 est un branchement. Pour permettre de réduire ce goulôt d'étranglement, il est nécessaire de prédire plusieurs branchements par cycle [15].

Une autre difficulté surgit avec la possibilité d'exécuter un grand nombre d'instructions indépendantes en parallèle. Souvent les applications n'exhibent pas ces instructions indépendantes : or l'exécution d'un programme doit respecter les dépendances entre les instructions. La prédiction de branchement est un premier accroc à ce respect des dépendances : toute instruction postérieure à un branchement est dépendante de ce branchement ; cette dépendance est «cassée» par la prédiction, mais les instructions sont validées dans l'ordre du programme. Récemment, il a été noté que le même principe pouvait être appliqué pour aussi «casser» les dépendances de données sur les programmes : on peut ainsi prédire le résultat d'une instruction ou d'un calcul d'adresse [LS96a,SVS96].

3.3 Simulation de processeurs et collecte de traces

Mots clés : collecte de traces, simulation.

Résumé : *La validation des nouvelles idées en architecture de processeur passe par la simulation la plus précise possible du microprocesseur et de tout son environnement. Cette simulation doit être faite cycle par cycle et doit tenir compte de*

[Yeh93] T. Yeh, *Two-level adaptive branch prediction and instruction fetch mechanisms for high performance superscalar processors*, thèse de doctorat, University of Michigan, 1993.

[CHP97] P. Chang, E. Hao, Y. Patt, « Target prediction for indirect jumps », *in : Proceedings of the 24th Annual International Symposium on Computer Architecture*, 1997.

[LS96a] M. Lipasti, J. Shen, « Exceeding the dataflow limit with value prediction », *in : Proceedings of the 29th International Symposium on Microarchitecture*, 1996.

[SVS96] Y. Sazeides, S. Vassiliadis, J. Smith, « The performance potential of data dependence speculation and Collapsing », *in : Proceedings of the 29th International Symposium on Microarchitecture*, 1996.

l'ensemble des interactions à l'intérieur du processeur. De plus cette simulation doit être faite sur des applications si possible représentatives de la charge d'un processeur dans son environnement potentiel d'utilisation.

Deux approches sont utilisées, la simulation dirigée par l'exécution et la simulation dirigée par les traces. Nous décrivons ici ces deux approches, leurs intérêts et limitations réciproques.

Afin de valider, au niveau performance, les architectures de processeurs, la simulation est le seul outil accepté aussi bien par l'industrie que par la communauté de recherche. Cette simulation doit être faite avant le début de la conception matérielle.

Cette simulation doit être la plus précise possible et tenir compte de l'ensemble des interactions à l'intérieur du processseur. Deux approches peuvent être utilisées : la simulation dirigée par les traces et la simulation dirigée par l'exécution.

La simulation d'architecture dirigée par les traces présente l'avantage de décorréler la simulation de l'architecture de la collecte de traces [UM97]. Ainsi on pourra simuler une architecture en lui fournissant la trace de l'exécution d'une application c'est-à-dire par exemple la liste des instructions exécutées et des adresses accédées en mémoire. Cette approche a été utilisée depuis très longtemps en architecture de processeur. Les traces peuvent être collectées soit par matériel, soit par logiciel.

La collecte de traces d'exécution par matériel (analyseur logique) a été utilisée tant que les données et instructions circulaient sur les pattes d'entrées/sorties des processeurs. Sur les processeurs actuels, la collecte de traces ne peut plus être faite de cette manière. Ce qui explique que la collecte de traces par instrumentation logicielle soit la plus utilisée par la recherche en architecture (et aussi par l'industrie). Des outils adaptés à chaque jeu d'instructions sont aujourd'hui disponibles (Pixie, Atom, spy, EEL,...). Ces outils présentent le défaut de ne pouvoir tracer qu'une seule application et ne permettent pas en général de tracer l'activité système du processeur. Enfin le ralentissement des applications tracées est considérable (facteur 10-100) et ne permet pas d'envisager le traçage réaliste d'applications de grande taille (plusieurs centaines de milliards d'instructions). Enfin, elle est inappropriée pour la simulation réaliste de processeurs permettant l'exécution spéculative (c'est-à-dire prédisant les branchements et exécutant dans le désordre) : l'exécution spéculative requiert l'accès (en lecture) aux instructions de la fausse branche ainsi qu'aux données en mémoire de l'application tracée.

La simulation dirigée par l'exécution nécessite *l'exécution* par le simulateur de l'application tracée elle-même. Cette approche permet contrairement à la simulation dirigée par les traces de simuler l'impact des instructions exécutées spéculativement. Cependant cette approche peut s'avérer extrèmement lourde puisqu'il faut être capable de simuler non seulement le code directement écrit par le développeur, mais aussi les appels à des bibliothèques dynamiques et les appels systèmes, c'est-à-dire toutes les opérations susceptibles de modifier le contenu de la mémoire associée à l'application tracée. Cette approche a été suivie dans le simulateur SimOS. SimOS [RHWG95] est le simulateur complet d'une station MIPS "bootant" le système

[UM97] R. Uhlig, T. Mudge, « Trace-Driven memory simulation: a survey », ACM *Computing Surveys*, 1997.

[RHWG95] M. Rosemblum, S. Herrod, E. Witchel, A. Gupta, « Complete computer system simulation : the SimOS approach », IEEE *Parallel and Distributed Technology n° 3*, 1995.

Irix. L'avantage de SimOS est ainsi de permettre de simuler un processeur avec l'ensemble de son système d'exploitation. Par contre, les performances de la simulation restent très limitées et ne permettent pas d'envisager la simulation des "grosses" applications (plusieurs centaines de milliards d'instructions).

Le constat global est que la majeure partie des études pour les architectures de *demain* sont faites sur des traces d'applications dont on a souvent réduit le volume pour permettre des temps de simulation acceptables. Ceci peut conduire à des erreurs majeures pour le dimensionnement de structures telles que prédicteurs de branchement, antémémoire ou TLBs.. Le challenge en recherche pour la simulation réaliste d'architectures de processeurs est en fait aujourd'hui de parvenir à simuler le comportement des applications en vraies grandeurs et dans leur environnement système.

3.4 Compilation pour architectures hautes performances

Mots clés : hautes performances, compilation, hiérarchie mémoire, optimisation, transformation de code.

Résumé : *L'efficacité de l'exécution d'une application tant sur une machine multiprocesseur que sur un* PC *ou une station de travail dépend très fortement de la structure des programmes. Cette structure est imposée par le programmeur mais comporte des degrés de liberté que des techniques logicielles, appelées optimisations de code, peuvent exploiter pour augmenter la performance des applications. Nous présentons un rapide aperçu des transformations de code disponibles pour implémenter un compilateur optimiseur. Ces transformations peuvent être mises en oeuvre tant au niveau du code source que du code machine.*

L'efficacité des mécanismes matériels pour l'exploitation de la localité des références mémoire et du parallélisme, tant au niveau multiprocesseurs qu'instruction ("Instruction Level Parallelism"), dépend très fortement de la structure des programmes. Cette structure est imposée par le programmeur mais comporte des degrés de liberté que des techniques logicielles, appelées optimisations de code, peuvent exploiter pour augmenter la performance des applications. Ces optimisations de code sont fondées sur des transformations de programmes, qui respectent la sémantique des codes, mais réorganisent les calculs pour une meilleure exploitation d'une architecture donnée.

Les transformations de code destinées à l'amélioration de performances peuvent intervenir à plusieurs étapes dans un processus de compilation. La figure 1 montre l'organisation générale d'un compilateur. Des transformations de code peuvent être effectuées aussi bien au niveau du code source qu'au niveau du code machine.

Les optimisations effectuées au niveau du code machine sont principalement les optimisations "Peephole", qui consistent à remplacer des séquences d'instructions par des séquences plus rapides, et surtout l'application des techniques d'ordonnancement de code. Cet ordonnancement doit prendre en compte les caractéristiques fines de l'architecture telles que le nombre de registres disponibles, l'usage des ressources des processeurs, etc.

Par exemple, pour l'exploitation du parallélisme d'instructions au niveau logiciel, les méthodes les plus simples se restreignent à l'exploitation du parallélisme entre les instructions

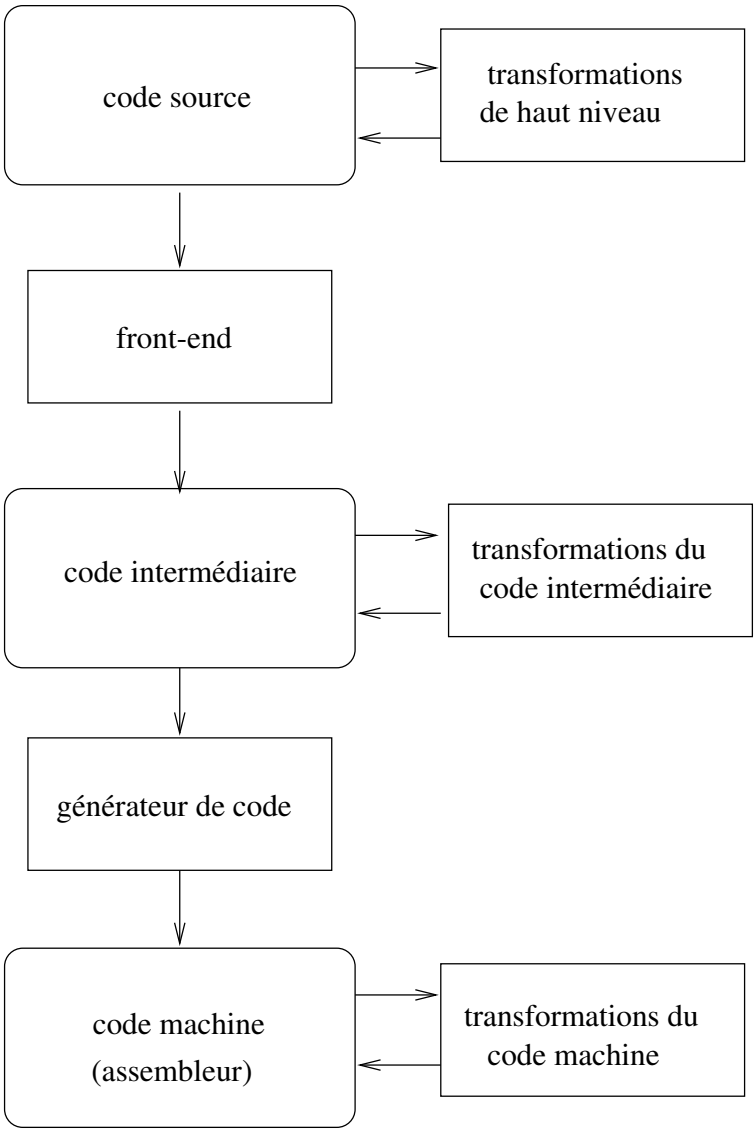


FIG. 1 – *Organisation d'un compilateur.*

d'un même bloc de base[1]. Cependant, le nombre limité d'instructions dans un bloc de base réduit l'efficacité de ce type de techniques. En pratique, surtout dans le cas des boucles, il faut extraire le parallélisme entre des instructions de plusieurs blocs de base, par exemple en utilisant la technique du pipeline logiciel. Cette technique, fondée sur l'exploitation du parallélisme disponible entre les instructions d'itérations différentes, consiste à segmenter le code des boucles d'une manière similaire à celle utilisée par les pipelines matériels.

Au niveau du code source, les transformations de programmes utilisent toutes les informations sémantiques disponibles tant au niveau du contrôle de flot que de l'usage des variables. A ce niveau, des réorganisations majeures du code peuvent être effectuées telles que par exemple le remplacement de l'appel d'une procédure par le corps de celle-ci ("inlining"). C'est aussi sur le code source que l'on peut appliquer les techniques de parallélisation automatique et les méthodes d'optimisation de la localité. Par exemple, la performance d'une hiérarchie mémoire dépend très fortement des caractéristiques de localité des accès aux données effectués par un programme. La prise en compte de la hiérarchie mémoire par un compilateur consiste à considérer les trois aspects fondamentaux suivants :

Détection et estimation de la localité : La détection de la localité est fortement liée au calcul des dépendances de données. En effet, si une dépendance existe, alors il y a réutilisation de données. Le deuxième aspect de cette question est de déterminer la proportion de références mémoire qui peuvent être évitées par l'exploitation effective de cette localité.

Exploitation de la localité : L'exploitation de la localité consiste essentiellement à déterminer le niveau de la hiérarchie mémoire qui tirera parti de la localité présente et à adapter la génération de code en conséquence.

Optimisation de la localité : Ces transformations de code ont pour but de restructurer les calculs pour permettre l'exploitation effective, par un niveau choisi de la hiérarchie, de la localité présente.

Il existe un nombre très important de transformations du code source pouvant être utilisées pour améliorer le comportement de la hiérarchie mémoire sur une application et/ou la paralléliser [BGS94]. La plupart de ces optimisations s'appliquent aux boucles. Parmi celles-ci on peut citer :

Blocage de boucles : Dans le cadre de l'optimisation de la localité, cette transformation permet de diviser l'espace d'itérations en pavés de telle sorte que les données réutilisées puissent être contenues dans un niveau de la hiérarchie mémoire.

Distribution de boucle : Les instructions d'une boucle sont réparties dans plusieurs boucles ayant le même espace d'itération que l'original. Cette transformation est utilisée pour diminuer la pression sur les registres ou extraire des calculs parallèles d'une boucle séquentielle.

Fusion de boucles : Les instructions de deux boucles sont fusionnées dans une seule boucle. Elle est par exemple utilisée pour améliorer les réutilisations de données.

1. Une séquence d'instructions comportant un seul point d'entrée (la première instruction) et un seul point de sortie (la dernière instruction).

[BGS94] D. Bacon, S. Graham, O. Sharp, « Compiler Transformations for High-Performance Computing », *ACM Computing Surveys 26*, 4, décembre 1994, p. 345–420.

Dépliage de boucle : Cette transformation consiste à répliquer le corps de la boucle. Cette transformation, toujours légale, permet de diminuer le coût de gestion de la boucle et augmente le parallélisme d'instructions potentiellement exploitable par les processeurs.

Strip-mining : Le "strip-mining" découpe l'espace d'itérations de boucle en blocs. Il permet d'ajuster la granularité des opérations dans le cas de la parallélisation ou de la vectorisation.

Ces transformations sont aujourd'hui relativement bien comprises individuellement. Le challenge est aujourd'hui de maîtriser l'interaction de toutes ces transformations et leurs impacts sur les performances.

3.5 Mémoire virtuellement partagée

Mots clés : MVP, gestion mémoire.

Résumé : *La programmation des architectures parallèles à mémoire distribuée est rendue difficile notamment par la présence de plusieurs espaces d'adressage disjoints. L'opération de distribution des données d'une application parmi ces différents espaces d'adressage est une tâche difficile. Le mécanisme de mémoire virtuelle partagée offre un seul espace d'adressage logique permettant d'éviter cette distribution explicite. Pour une implémentation efficace, le mécanisme de* MVP *s'appuie sur l'utilisation de caches dont la cohérence doit être assurée. Le choix d'un protocole de cohérence s'appuie sur les modèles de consistance mémoire qui en constituent les fondements scientifiques.*

La gestion des données dans une architecture parallèle à mémoire distribuée (APMD) est rendue complexe par la distribution physique des mémoires. Ce caractère distribué de la mémoire oblige l'utilisateur ou un compilateur "intelligent" à distribuer les données de l'algorithme devant s'exécuter en parallèle. Cette distribution des données est une opération complexe qui demande une très bonne connaissance de l'application à paralléliser. Cette distribution explicite peut être évitée grâce à des mécanismes de gestion de données permettant la migration de celles-ci en fonction des calculs effectués par chaque processeur. La mémoire virtuelle partagée (MVP) [Li86] est un exemple de mécanisme de gestion de données. Un tel concept offre un espace d'adressage global pour une architecture parallèle ayant un ensemble d'espaces d'adressage disjoints (APMD). L'implémentation d'un mécanisme de MVP s'appuie sur des mécanismes de gestion mémoire virtuelle au sein ou au dessus d'un système d'exploitation. L'espace d'adressage global est une région de mémoire virtuelle composée de pages migrant à la demande entre les processeurs selon les accès mémoire. Chaque mémoire locale agit comme un large cache, ou mémoire attractive, contenant les pages précédemment accédées. Comme tout dispositif fondé sur l'utilisation de caches, le problème de la cohérence de ces caches se pose.

Le concept de MVP fournit une vision globale de la mémoire dans laquelle les calculateurs peuvent lire ou écrire. Vis à vis de l'utilisateur, il offre également un modèle mémoire qui

[Li86] K. Li, *Shared Virtual Memory on Loosely Coupled Multiprocessors*, thèse de doctorat, Yale University, septembre 1986.

caractérise le comportement de la mémoire lorsque plusieurs calculateurs effectuent des accès simultanés. De façon intuitive, l'utilisateur souhaite que la mémoire fournisse toujours le dernier résultat qui a été écrit dans la mémoire. Cependant, dans un système parallèle, la notion de "dernier accès" est ambiguë. Il oblige à définir un ordre total sur tous les accès mémoire, ce qui n'est pas souvent nécessaire. Le modèle de cohérence séquentielle est un exemple de modèle mémoire dont les accès sont consistants avec un ordre total. Un système mémoire possède la propriété de cohérence séquentielle si tous les processus voient les accès mémoire comme si ils avaient été exécutés sur un calculateur séquentiel multiprogrammé. Du point de la vue de la mise en œuvre d'une MVP, un tel modèle impose de nombreuses communications (accès aux pages, invalidation, etc.). Plusieurs travaux ont été réalisés afin de concevoir de nouveaux modèles mémoire pouvant être implémentés plus efficacement sur des systèmes parallèles.

Parmi ceux-ci, le modèle de cohérence à la libération [GLL+90] a été un des plus étudiés. Le principe de ce modèle mémoire repose sur le constat que les accès aux données, effectués par un programme parallèle, sont souvent synchronisés. Le modèle mémoire à consistance à la libération est fondé sur l'utilisation de deux classes d'opérations sur la mémoire. La première classe regroupe les opérations classiques de lecture et d'écriture tandis que la deuxième classe contient les opérations de synchronisation : libération et acquisition. Le rôle de ces deux opérations est de propager les modifications qui ont été réalisées par les opérations d'écriture. Une opération de libération indique qu'un processeur a effectué des modifications et que celles-ci doivent être communiquées à tout processeur qui effectuera une opération d'acquisition. De même, une opération d'acquisition indique qu'un processeur va exécuter des opérations qui nécessitent la connaissance des modifications effectuées par les processeurs ayant exécuté une opération de libération. Deux formes de cohérence à la libération ont été proposées [KCZ92]. La première forme est appelée cohérence à la libération impatiente. Les modifications, réalisées depuis la dernière opération d'acquisition, sont propagées à tous les autres processeurs lors de la libération. La deuxième forme, appelée cohérence à la libération paresseuse, diffère de la précédente par le moment choisi pour diffuser les modifications. Plutôt que de le faire à la libération, les modifications sont propagées lors de l'opération d'acquisition. Lors de l'acquisition, le processeur détermine quelles sont les modifications valides dont il a besoin en fonction de la définition du modèle de cohérence à la libération. Cette approche permet ainsi de réduire fortement le nombre de messages Une première mise en œuvre de la cohérence à la libération paresseuse a été effectuée au sein de la MVP TreadMarks[KDCZ94]. Koan et Myoan implémentent une autre forme de cohérence permettant la modification simultanée par de multiples écrivains d'une même page [10].

Le modèle de cohérence relachée implémentée dans Mome permet à un processeur de de-

[GLL+90] K. Gharachorloo, D. Lenoski, J. Laudon, P. Gibbons, A. Gupta, J. Henessy, « Memory Consistency and event ordering in scalable shared memory multiprocessors », *in : 17th Annual International Symposium on Computer Architectures*, ACM, p. 15–26, mai 1990.

[KCZ92] P. Keleher, A. Cox, W. Zwaenepoel, « Lazy Release Consistency for Software Distributed Shared Memory », *in : 19th International Symposium on Computer Architecture*, p. 13–21, mai 1992.

[KDCZ94] P. Keleher, D. Dwarkadas, A. Cox, W. Zwaenepoel, « TreadMarks: Distributed Shared Memory on standard workstations and operating systems », *in : Proceedings of the 1994 Winter Usenix Conference*, p. 115–131, janvier 1994.

mander à voir toutes les modifications qui ont été apportées à une section de mémoire partagée avant un événement de synchronisation comme une barrière ou un verrou. La mise en oeuvre de ce modèle est basée sur l'utilisation d'une horloge globale à laquelle les requêtes de consistance font référence.

4 Domaines d'applications

4.1 Panorama

Résumé : *De par ses objectifs, le projet Caps travaille sur les technologies de base de l'informatique : architecture des processeurs (cf. 2.2), compilation et système orientés performance (cf. 2.3). Nos travaux induisent aussi le développement de prototypes logiciels (cf. 5.2,5.3,5.5, 5.4) dont les domaines d'applications sont essentiellement le calcul hautes performances : image, calcul scientifique,...*

4.2 Couplage de codes de simulation

Participants : Christophe René, Thierry Priol.

Mots clés : simulation numérique distribuée, metacomputing, couplage de codes.

Résumé : *L'étude du problème de couplage de codes de simulation est réalisée dans le cadre d'une collaboration avec la société Aérospatiale Matra. Nous étudions notamment les aspects informatique du couplage de codes de simulation dans le domaine de l'électromagnétisme. Le problème consiste à permettre la simulation des interactions électromagnétique entre deux objets phyiques. La simulation de chaque objet est effectuée par une instance d'un code parallèle de simulation conçu par Aérospatiale Matra. L'objectif est de permettre le couplage de ces deux instances du même code en y effectuant très peu de modifications avec des contraintes de haute performance. Pour atteindre cet objectif, nous utilisons le concept d'objet Corba parallèle qui permet d'encapsuler un code parallèle au sein d'un objet distribué.*

5 Logiciels

5.1 Panorama

Résumé : *Le projet Caps développe de nombreux prototypes logiciels de recherche : compilateurs, simulateurs, environnement de programmation,... Nous présentons ici* **Salto, Mome**, **PaCo** *et* **Do!**, *4 logiciels conséquents aujourd'hui disponibles développés au sein du projet.*

5.2 Salto : un environnement de transformations pour les langages d'assemblages (cf. 2.3)

Participants : François Bodin, Erven Rohou, André Seznec.

Mots clés : optimisation.

Contact: François Bodin
Statut : Déposé à l'APP sous le numéro IDDN.FR.001.070004.00.R.C.1998.000.10600, disponible sur demande.

Salto propose un environnement de manipulation de programmes en langage assembleur. Une abstraction des ressources matérielles exploitables permet de les dissocier de l'algorithme d'optimisation, ce qui a deux avantages :

- le même algorithme peut être appliqué à des programmes écrits pour différentes architectures avec très peu de modifications ;
- la manipulation du code assembleur est grandement simplifiée.

Salto est composé de quatre parties :

1. le noyau effectue toutes les tâches nécessaires, rébarbatives et souvent sources d'erreurs dont le programmeur a envie de se passer, notamment l'analyse lexicale et syntaxique du code, le calcul de la structure en blocs de base et du flot de contrôle, le calcul des dépendances entre instructions ;
2. la description de la machine est un fichier qui détaille le jeu d'instructions et l'ensemble des ressources matérielles de l'architecture cible qui sont susceptibles d'intervenir dans le processus d'optimisation. Elle peut être plus ou moins précise : une description simple peut s'intéresser simplement aux unités fonctionnelles tandis qu'une description plus fine peut faire intervenir les bus d'accès à la mémoire, les ports sur le fichier de registres, etc. ;
3. l'interface utilisateur orientée objets donne un moyen d'accès aux structures de données internes de Salto. Un certain nombre de classes correspondent aux types de données connus ;
4. un algorithme d'instrumentation ou d'optimisation fourni par l'utilisateur utilise l'interface pour accéder au code et éventuellement le modifier. Salto en lui-même n'a aucun effet sur le programme assembleur, il se contente de fournir des abstractions du code et des méthodes à même de faciliter l'implantation d'algorithmes. C'est à l'utilisateur de spécialiser Salto pour obtenir un outil correspondant à ses besoins.

Pour en savoir plus se référer à http://www.irisa.fr/caps/PROJECTS/Salto ou contacter François Bodin.

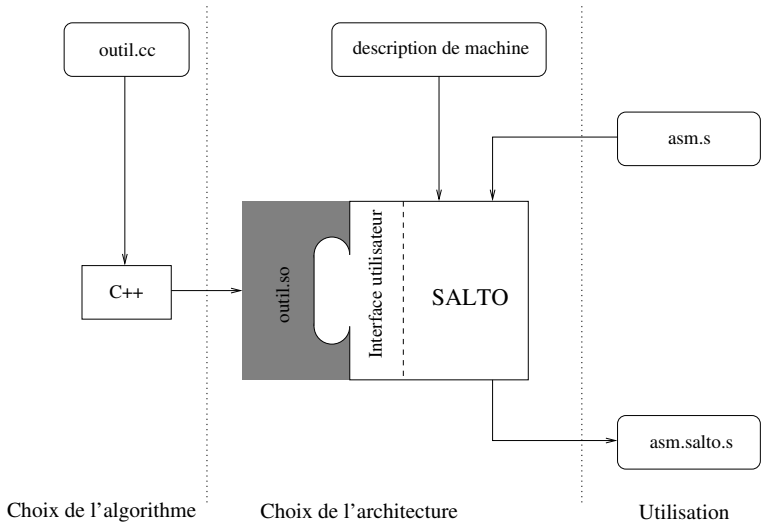
5.3 Mome : une mémoire virtuelle partagée pour des langages parallèles (cf. 6.3)

Participant : Yvon Jegou.

Mots clés : Mémoire virtuelle partagée.

Contact: Yvon Jégou
Statut : En cours de dépot à l'APP.

La MVP Mome permet la communication par partage de mémoire sur des architectures à mémoires distribuées. Mome met en oeuvre un modèle de consistance relachée avec écrivains multiples. La consistance d'une page peut être contrôlée par chaque processeur en faisant



référence à certains événements. Par exemple, il est possible de demander qu'un accès en lecture sur une section de mémoire partagée fournisse toutes les modifications apportées à cette section avant la dernière barrière de synchronisation ou avant le relachement d'un verrou. Mome met en oeuvre une horloge globale qui permet de dater les événements et qui sert de référence aux demandes de consistance.

Mome a été portée sur les calculateurs Cenju-3 et Cenju-4 de Nec sous micro-noyau Mach et sur des stations de travail sous Unix (Sparc Solaris ou PC sous Linux). Plusieurs couches de communications ont été développées : IPC et MPI pour les versions Mach, TCP/IP et SCI pour les versions Unix.

5.4 PaCo : objet Corba parallèle (cf. 6.4)

Participants : Thierry Priol, Christophe René.

Contact : Thierry Priol.
Statut : En cours de dépot à l'APP.

Paco est une mise en oeuvre du concept d'objet Corba parallèle au sein de Mico, une implémentation de Corba réalisée par l'université de Francfort. Un objet Corba parallèle se présente sous la forme d'une collection d'objets Corba identiques. Pour cacher totalement la nature parallèle de l'objet, nous avons proposé une extension du langage de spécification d'interface (IDL) afin d'y introduire des moyens d'expression du parallélisme. Ces extensions ont été ajoutées dans le compilateur IDL de Mico. Le processus de génération des sl talons et des *squelettes* a été modifié afin de permettre l'exécution simultanée d'une opération sur les objets appartenant à la collection et de distribuer les données entre les objets. La distribution de donnée est réalisée par la bibliothèque Dalib du GMD (T. Brandes). Cette bibliothèque de fonctions de distributions s'appuie sur MPICH, une implémentation de la bibliothèque de communication MPI. Paco est disponible sur des réseaux de machines Unix. Une implémentation sur le calculateur parallèle Nec Cenju-4 a été effectuée par T. Kamachi dans le cadre de la

collaboration Inria-Nec.

5.5 Do! : Générateur automatique de code Java réparti

Participants : Pascale Launay, Jean-Louis Pazat.

Mots clés : frameworks, objets, transformations de programmes.

Contact : Jean-Louis Pazat
Statut : Déposé à l'APP sous ne numéro IDDN.FR.001.270020.00.R.P.1998.000.10600, disponible sur le serveur Web du projet.

Le logiciel Do! réalise une génération automatique de code réparti à partir de code Java parallèle centralisé. Le modèle de programmation parallèle est exprimé par un framework, qui permet de limiter l'expression du parallélisme sans modification du langage Java. Le placement des tâches et des données sur les processeurs est dérivé d'indications du programmeur sur les caractéristiques de distribution de son application. Le code généré s'appuie sur le RMI Java et un exécutif (classes Java) permettant la création distante d'objets. Par rapport à l'approche HPF, nous prenons en compte dans un même cadre la génération de code réparti par distribution de contrôle et par distribution de données.

Le modèle de programmation parallèle de Do! est basé sur les notions d'*objets actifs* ("tâches") et de *collections* pouvant contenir tout type d'éléments (et en particulier des tâches). Il est structuré sous forme d'un *framework*, basé sur le *design pattern* des opérateurs.

Dans le modèle d'exécution de Do!, les collections sont distribuées (leur éléments, tâches ou données, sont répartis sur les processeurs). Le placement des tâches et des données est donc guidé par la distribution des collections.

La transformation d'un programme centralisé en programme réparti est réalisé automatiquement par Do! en changeant la bibliothèque des collections utilisées pour le parallélisme (utilisation des collections distribuées), et en transformant les composants définis par le programmeur, afin d'assurer une localisation transparente des objets du programme (placement et accès).

6 Résultats nouveaux

6.1 Architectures de processeurs (cf. 2.2)

Participants : Thierry Lafage, André Seznec, François Bodin, Dan Truong, Pierre Michaud.

Mots clés : microprocesseur, Risc, antémémoire, localité, hiérarchie mémoire, prédiction de branchement, multiflots.

Résumé : *Les actions de recherches du projet Caps en architecture de processeurs portent sur la structure et les optimisations matérielles et logicielles des*

hiérarchies mémoire, en particulier antémémoires, sur les mécanismes de lancement des instructions, en particulier prédiction de branchement ainsi que sur les structures de processeur multiflot simultané.

Les antémémoires

Participants : François Bodin, André Seznec, Dan Truong.

Optimisation de la localité des programmes L'analyse de la localité des programmes est essentielle pour obtenir de bonnes performances sur les machines dotées d'une hiérarchie mémoire. Il s'agit de détecter et de caractériser les parties de structures de données sujettes à des réutilisations. Ces informations sont ensuite utilisées pour transformer les programmes afin d'améliorer l'exploitation de la hiérarchie mémoire. Ce type de transformations est essentiel pour l'obtention de performances sur les processeurs actuels dont un des goulets d'étranglement principaux est constitué par les accès à la mémoire.

Deux types de données sont fréquemments utilisées pour stocker de gros volumes de données dans les programmes, les matrices et les structures. Il existe de nombreux travaux antérieurs pour améliorer la localité spatiale des matrices, mais peu de travaux ont été faits pour les applications non scientifiques.

Nous avons proposé des techniques pour améliorer la localité spatiale des structures de données hétérogènes pour les programmes écrits en C. Une librairie d'allocation dynamique performante a été conçue pour faciliter l'optimisation manuelle des programmes écrits en C. Les résultats obtenus sur une *Origin 2000* indiquent que ces optimisations permettent d'obtenir des accélérations significatives pour des programmes limités par la performance de la hiérarchie mémoire, ce indépendament du jeu de données en entrée. Notre librairie d'allocation est, de plus, plus rapide que les librairies d'allocation dynamiques standard, et permet d'obtenir aussi des gains de performance pour les applications qui font un usage intensif des routines d'allocation.

Étude des mécanismes de séquencement

Participants : Pierre Michaud, André Seznec.

Prédiction de branchement plusieurs blocs en avance Pour permettre de réduire le goulot d'étranglement noté sur les processeurs à nombreuses unités fonctionnelles, nous avons proposé en 1996 un mécanisme appelé «multiple-block ahead branch predictor», [15]. Ce mécanisme prédit efficacement les adresses de plusieurs blocs d'instructions en avance. Un mécanisme différent appelé «trace cache» a aussi été proposé en 1996 par l'équipe de Jim Smith (université du Wisconsin) [RBS96].

[RBS96] E. Rotenberg, S. Bennet, J. Smith, « Trace cache : a low latency approach to high bandwidth instruction fetching », *in : Proceedings of the 29th International Symposium on Microarchitecture*, 1996.

Une étude a été menée afin de mettre en relief les qualités et défauts respectifs des deux approches [11, 13]. Nous avons montré que le principal défaut du «trace cache» est qu'il a une moindre contenance qu'un cache d'instruction classique à taille physique équivalente : le «multiple-block ahead branch predictor», qui repose sur l'utilisation d'un cache d'instructions, sollicite moins le cache secondaire. La principale qualité du «trace cache» est qu'il entraine une pénalité de mauvaise prédiction de branchement plus faible : l'alignement des instructions, leur décodage, et une partie du travail de renommage a déjà été effectué au moment de la construction des traces. Cette étude nous a permis de constater que les performances n'augmentent pas linéairement avec le débit de chargement d'instructions.

Nous avons mené une étude afin de mieux comprendre les besoins en débit de chargement d'instructions dans les processeurs superscalaires à exécution non ordonnée [30]. Cette étude a montré que le besoin en débit de chargement d'instructions dépend de la fréquence des mauvaises prédictions de branchements et du parallélisme d'instructions des applications. En particulier, nous avons montré que le besoin en débit de chargement est proportionnel au degré de parallélisme d'instructions dans une fenêtre d'instructions de taille fixe, et qu'il varie également comme la racine carrée de la distance en instructions entre 2 branchements mal prédits successifs. Cette étude permet de mieux comprendre l'impact de nouvelles techniques architecturales comme la prédiction de valeur [LS96b] ou la prédication [MHB+94] sur le dimensionnement des mécanismes de chargement d'instructions.

Prédicteur de branchements biaisé Les performances des microprocesseurs actuels reposent de plus en plus sur les mécanismes de prédiction de branchements dynamiques. Les tables de prédiction des branchements conditionnels sont en général mises en œuvre sans utiliser d'étiquettes, ce qui entraîne un phénomène d'interférence appelé «aliasing». Comme l'introduction d'associativité dans ces tables nécessiterait la présence d'étiquettes coûteuses, une nouvelle approche pour résoudre le problème de l'«aliasing» de conflit sans utiliser d'étiquettes a été proposée. Le Skewed Branch Predictor est une structure à trois tables : chaque table est indexée avec une fonction différente et la prédiction est fournie par un vote à la majorité. Lorsqu'une prédiction est mauvaise, les trois bancs sont mis à jour. Cette redondance augmente l'aliasing de capacité, mais le compromis entre la réduction de l'aliasing de conflit et l'augmentation de l'aliasing de capacité s'avère bénéfique [13].

Multiflot simultané

Participant : André Seznec.

Parmi les solutions pour exploiter les possibilités d'intégration, le *multiflot simultané* [TEL95]

[LS96b] M. Lipasti, J. Shen, « Exceeding the dataflow limit with value prediction », *in : Proceedings of the 29th International Symposium on Microachitecture*, 1996.

[MHB+94] S. A. Mahlke, R. E. Hank, R. A. Bringmann, J. C. Gyllenhaal, D. M. Gallagher, W. mei W. Hwu, « Characterizing the impact of predicated execution on branch prediction », *in : Proceedings of the 27th Annual International Symposium on Microarchitecture*, 1994.

[TEL95] D. Tullsen, S. Eggers, H. Levy, « Simultaneous multithreading : maximising on-chip parallelism », *in : 22nd Annual International Symposium on Computer Architecture*, p. 392–403, juin 1995.

est l'une des solutions les plus prometteuses. Disposer de plusieurs flots exécutables simultanément sur une architecture *superscalaire* doit permettre de maximiser le taux d'utilisation du processeur et donc les performances. De nombreux paramètres entrent en jeu dans la conception d'une architecture *multiflot*: nombre de supports physiques de flots, types et partage des ressources, hiérarchie mémoire, etc.

Un simulateur a été développé afin d'explorer les choix possibles pour les architectures *multiflots*. Les travaux menés ont permis d'établir que l'utilisation d'une hiérarchie mémoire standard ne permettra pas d'utiliser plus de 4 ou 6 flots simultanément[8]. Ceci est dû à la saturation de la bande passante entre les premiers et seconds niveaux de caches. D'autre part, nous avons aussi montré que l'exécution dans l'ordre sur un processeur multiflot permettrait d'obtenir pratiquement le même niveau de performance qu'une exécution dans le désordre [25]. Enfin, une étude est en cours pour quantifier le comportement des architectures *multiflots simultanés* en présence de changements de contexte.

6.2 Environnement pour architectures hautes performances (cf. 2.3)

Mots clés : Matlab, programmation parallèle, parallélisation automatique, portage d'applications, optimisation.

Participants : François Bodin, Stéphane Chauveau, Thierry Lafage, Yann Mével, Erven Rohou, André Seznec, Paul Van der Mark.

Résumé : *L'obtention de performances sur les architectures hautes performances nécessitent des outils logiciels adaptés qui cachent à l'utilisateur la complexité des matériels et des systèmes.*

Les actions de recherches que nous menons visent à fournir aux utilisateurs de calculateurs hautes performances des outils tels que compilateur, aide au portage, optimiseur pour permettre des développements et/ou portages d'applications hautes performances.

Ainsi, nous développons TSF*, un outil d'aide au portage de codes Fortran sur architectures hautes performances et Salto un environnement de manipulation de langage d'assemblage.*

TSF : aide au portage sur les architectures hautes performances

Participants : François Bodin, Yann Mével.

Le portage (parallélisation et amélioration de performance) d'applications sur machines hautes performances est une activité techniquement difficile faisant appel à beaucoup de savoir-faire.

Le système TSF vise à accélérer cette activité grâce à l'utilisation conjointe de techniques issues de deux domaines : la parallélisation automatique et le raisonnement à partir de cas («Case-Based Reasoning»).

La parallélisation automatique fournit une bibliothèque de transformations de programmes à appliquer sous contrôle de l'utilisateur, dans le cas de TSF. Le raisonnement à partir de

cas permet de recueillir les fruits des expériences de portage précédentes et fournit un accès au savoir-faire de l'expert. De plus pour aider l'utilisateur, le système permet d'instrumenter les codes afin d'analyser leurs performances. L'intégration des deux méthodes a pour objectif d'aider au choix des transformations adaptées en s'appuyant sur des expériences de portage similaires répertoriées dans la base de cas. Avec la technique de raisonnement à partir de cas, ces choix sont guidés par des calculs de similarité de contexte.

Cette étude est réalisée en collaboration avec la société Simulog qui fournit toute l'infrastructure de base à l'analyse de programmes Fortran grâce à son outil Foresys et le projet Irisa Repco (R. Quiniou).

TSF fait l'objet d'une convention avec la société Simulog pour sa commercialisation. De plus TSF a été choisi comme technologie de base pour le projet Esprit Post.

Stratégie de compilation et interactions globales entre optimisations logicielles

Participants : François Bodin, Erven Rohou, André Seznec, Paul Van der Mark.

Au cours d'une chaîne de compilation classique, le type d'informations disponibles subit une profonde modification. Les premières phases ont une bonne connaissance de l'algorithme et des structures de données complexes utilisées par le langage de haut niveau. Les dernières phases ont une vision très précise de l'exploitation des ressources matérielles du processeur. Idéalement elles devraient connaître l'information que possédaient les premières transformations et disposer du maximum de connaissances pour appliquer efficacement des optimisations. Toutefois les compilateurs actuels ne propagent pas l'information et chaque étape doit régénérer une bonne partie des informations qu'elle utilise.

Il est important de conserver à chaque étape l'information facile à obtenir dans la mesure où celle-ci peut se révéler importante plus tard. Par exemple le générateur de code est souvent capable de déterminer si des accès à la mémoire sont dépendants par la connaissance qu'il a des variables locales et globales et des indices de tableaux. L'optimiseur bas-niveau a un besoin crucial de ce type d'information mais il ne peut que rarement l'obtenir sans faire appel à des mécanismes complexes comme la résolution d'équations diophantiennes.

Nous pensons aussi qu'il est important de briser la chaîne de compilation classique qui applique consciencieusement une liste de transformations dans un ordre immuable. L'information doit pouvoir remonter vers les optimisations de haut-niveau et celles-ci doivent être capables de défaire une partie de leur travail. Par exemple, il est parfois nécessaire de maîtriser les expansions de code résultant de nombreuses techniques de mise en œuvre du parallélisme à grain fin pour ne pas dégrader les performances du cache d'instructions [3].

Notre activité dans ce domaine se concentre sur l'interaction entre les optimisations au niveau du code source et celles effectuées au niveau du code machine.

Les réflexions sur la nécessité de faire communiquer les différentes phases d'un processus de compilation ainsi que de comparer les performances obtenues par différentes stratégies ont abouti au développement d'un premier prototype de compilateur. Les transformations de haut niveau et de bas niveau sont fortement couplées et s'échangent de l'information via un langage de communication simple appelé IL. Le principal objectif de ce langage est d'établir une carte qui mette en correspondance des portions de code haut-niveau et leur équivalent en assembleur.

Ce nommage permet par exemple au générateur de code de décrire des propriétés visibles à haut-niveau qui pourront être exploitées à bas niveau.

Le compilateur dispose d'un certain nombre de transformations. Chacune est capable de modifier un fragment de programme, de quantifier son action, éventuellement de l'annuler et de transmettre de l'information à l'extérieur. Il est alors possible d'établir une véritable stratégie de compilation à partir d'un ensemble de transformations, la stratégie s'exprimant comme un algorithme. De nombreux autres critères, tels que l'allocation des registres, sont pris en compte.

Une partie importante de ces travaux est effectuée dans le cadre du projet Esprit LTR Oceans [21].

Collecte de traces d'exécution

Participants : François Bodin, Thierry Lafage, Erven Rohou, André Seznec.

Dans le cadre de la simulation dirigée par la trace, les outils logiciels d'instrumentation actuellement disponibles (Pixie, Atom, ...) ont le défaut de ralentir de manière très significative l'exécution des applications (facteur 10-50), même si la trace générée n'est pas utilisée dans sa totalité. En effet, la trace d'une application est très volumineuse (plusieurs giga-octets pour de petites applications) et donc très difficile à utiliser en ce sens qu'une simulation sur une trace entière est très (trop) coûteuse en temps. Alors, des techniques telles que l'*échantillonnage de trace* sont utilisées pour réduire ce volume. D'autre part, tout programme comporte une phase d'initialisation qui n'est pas représentative de l'exécution totale. De ce fait, les premiers milliards d'éléments de trace qui correspondent à cette phase d'initialisation sont, dans la plupart des cas, ignorés.

Le ralentissement induit par l'instrumentation est trop important et pour cela ne permet pas, actuellement, de collecter ne serait-ce que quelques morceaux traces utilisables (i.e. après la phase d'initialisation) sur de réelles (grosses) applications exécutées dans leur totalité. Par réelles applications, nous entendons par exemple les applications de calcul intensif, les applications de type client-serveur (X, gestion de bases de données), mais aussi le système d'exploitation lui-même qui génère une activité rarement prise en compte.

Afin d'adresser ce problème, nous avons défini une nouvelle méthode de collecte de traces appelée « code cloning tracing » [27] qui permet de ne pas trop ralentir les parties du programme testé dont la trace sera rejetée. Le code original est dupliqué. Les deux copies (clones) sont instrumentées de manière légère. Cette légère instrumentation permet, à l'exécution, de passer dynamiquement d'un clone à l'autre. Un des clones, appelé *P_inst* est ensuite instrumenté de manière lourde afin de collecter les traces (adresses des données par exemple), alors que l'autre clone, appelé *P_exec* sera exécuté tel quel: il correspond au mode d'exécution rapide de l'application instrumentée, ou mode « sans trace ». Un premier prototype appelé **calvin** a été implémenté à l'aide de Salto [14].

Un des inconvénients majeurs de **calvin** est que la génération d'une trace différente nécessite de ré-instrumenter toutes les applications, ce qui est coûteux. Aussi, pour pouvoir tracer le code contenu dans les librairies partagées (libc, libm, ...), il est impératif d'instrumenter

chaque fichier source et de re-compiler le tout. Et pour chaque type de trace, l'opération est à renouveler.

Nous proposons alors une seconde solution qui est d'émuler le code (instruction par instruction) au lieu d'exécuter le clone *P_inst* en mode « trace ». De cette façon, le code n'est pas dupliqué, mais seulement instrumenté de manière légère pour qu'il puisse passer en mode « émulation » dynamiquement. La trace est aisément générée en mode « émulation » par appels de fonctions écrites en langage de haut niveau (C). La génération d'une trace différente nécessite uniquement de modifier l'émulateur et de refaire l'édition de lien des programmes à tracer. En mode « émulation », les librairies partagées sont automatiquement tracées (sans instrumentation) puisque leur code binaire est accessible à l'exécution.

calvin2 (une nouvelle version de **calvin**) et l'émulateur Dice constituent une implémentation de cette seconde solution. En mode « émulation-trace » les benchmarks SPEC95 subissent des ralentissements importants (de 92 à 191 pour les adresses des instructions exécutées et les adresses des données référencées), mais ce mode qui génère la trace est censé ne prendre le contrôle que rarement au cours de l'exécution complète des programmes à tracer. En mode « sans trace », les ralentissements vont de 1.09 à 1.69. Ainsi, les parties des programmes qui ne sont pas à tracer sont peu ralenties et nous permettent d'envisager le traçage de grosses applications.

6.3 Gestion de ressources au sein de calculateurs parallèles (cf. 6.3)

Participants : Yvon Jégou, Tsunehiko Kamachi, Renaud Lottiaux, David Mentré, Christine Morin, Thierry Priol.

Mots clés : SCI, réseau de stations de travail (Now), MVP.

Résumé : *La bonne gestion des ressources au sein d'une grappe de calculateurs est essentielle pour l'obtention de bonnes performances. Nous étudions les concepts qui permettent de gérer la mémoire (mémoire virtuelle partagée), les disques (système de gestion de fichiers parallèles) et les processeurs (migration de processus). Des études sont en cours sur différentes architectures: système parallèle (Nec Cenju-4) et grappes de* PC. *Nous nous intéressons notamment à l'intégration de ces différences concept au sein d'un système distribué pour grappes de* PC.

Gestion de ressources au sein de grappes de PC

Participants : Renaud Lottiaux, Christine Morin.

Du fait de l'augmentation continue de la puissance des microprocesseurs et de l'évolution de la technologie des réseaux d'interconnexion, les grappes de multiprocesseurs sont devenues des architectures attrayantes pour l'exécution d'applications de calcul et/ou d'accès aux données intensifs. Un problème clé des grappes de calculateurs est de combiner haute performance et haute disponibilité afin de pouvoir satisfaire les exigences des applications parallèles de longue durée.

Nous avons initié une activité de recherche portant sur la conception et la mise en oeuvre d'un système d'exploitation distribué, appelé Gobelins, pour le calcul haute performance sur des grappes de calculateurs. Pour garantir à la fois de hautes performances et la haute disponibilité, notre approche est de mettre en oeuvre une gestion globale et intégrée des ressources (mémoire, disque, processeur). Notre objectif est d'offrir la vision d'un système à image unique grâce à des mécanismes systèmes pour donner l'illusion d'un multiprocesseur extensible à mémoire partagée.

Cette année, nos efforts ont porté plus spécialement sur la conception d'un système couplant une mémoire virtuelle partagée et un système de gestion de fichiers parallèle ainsi que sur les mécanismes de tolérance aux fautes associés. Le système proposé est fondé sur une mémoire virtuelle partagée à pagination en mémoire distante pour la gestion globale de la ressource mémoire. La gestion globale de la ressource disque est assurée par un système de gestion de fichiers parallèle. Le choix de la projection de fichiers en mémoire virtuelle partagée comme interface du système de gestion de fichiers parallèle nous conduit à un système de stockage uniforme des données dans lequel les ressources mémoires et disques sont gérées de manière globale et intégrée. Un des principaux avantages de ce système est de décharger le programmeur de la gestion explicite des transferts de données entre les mémoires et les disques. En outre, les caches et les tampons d'écriture du système de gestion de fichiers sont complètement intégrés à la mémoire virtuelle partagée pour permettre une utilisation optimale des ressources en fonction de la charge du système. Des travaux sont en cours pour la conception de mécanismes de préchargement adaptés pour pallier les inconvénients liés à la petite taille des accès disque générés dans notre système par des défauts de pages et aux nombreux déplacements de la tête de lecture engendrés par les accès concurrents aux informations sur les disques. Un prototype de ce système a été réalisé sous la forme d'un module d'extension du noyau Linux et est en cours d'expérimentation sur une grappe de PCs bi-processeurs. La mise en oeuvre de ce prototype nous a amené à concevoir et réaliser un système de communication efficace adapté à l'implantation de services systèmes distribués au sein du système d'exploitation Linux. A la suite de cette expérience, des travaux sont en cours pour concevoir un système de communication efficace, fiable et portable (indépendant de la technologie du réseau d'interconnexion sous-jacent) pour des services systèmes distribués mis en oeuvre dans le noyau Linux.

Tolérer la défaillance d'un disque ainsi que le redémarrage ou la défaillance d'un noeud est essentiel dans le système Gobelins du fait de la gestion distribuée des ressources. Nous avons conçu des mécanismes de tolérance aux fautes pour tolérer ces types de défaillance dans le sous-système couplant la gestion de la mémoire et des disques. Ainsi, pour des raisons de simplicité et de performance, nous avons retenu une technique de mirroir (Raid-1) pour tolérer la défaillance d'un disque dans le système de gestion de fichiers parallèle de Gobelins. Pour tolérer la défaillance d'un noeud, Gobelins s'appuie sur une technique de recouvrement arrière. Il met en oeuvre un système de mémoire virtuelle partagée recouvrable (inspiré de Icare [9]) dans lequel les données de récupération sont créées en mémoire pour des raisons de performance. Un problème délicat que nous avons traité est la prise en compte par les mécanismes de tolérance aux fautes des transferts de données entre les mémoires et les disques. En outre, des mécanismes et des politiques de transfert des données de récupération sur disque ont été proposés pour d'une part faire face à l'encombrement de la mémoire par les données de récupération et d'autre part tolérer une panne de courant affectant l'ensemble de la grappe.

Ces mécanismes sont en cours de mise en oeuvre.

Mémoire virtuelle partagée comme support d'exécution de HPF

Participants : Yvon Jégou, Tsunehiko Kamachi.

La mémoire virtuelle partagée Mome en cours de développement intègre un protocole de cohérence relâchée avec multiples écrivains. Cette MVP cible tout particulièrement les codes issus d'un processus de compilation de type HPF. Ce type de processus de compilation considère les relations entre la distribution des itérations des boucles parallèles sur les processeurs et la distribution des accès dans les tableaux par analyse des fonctions d'indiçage. Il est généralement possible de prédire quels éléments des tableaux seront lus ou modifiés par chaque processeur avant d'exécuter une boucle parallèle. Mome met en œuvre un protocole de cohérence relâchée avec multiples écrivains. Plusieurs processeurs peuvent modifier des éléments différents d'une même page de mémoire sans entraîner d'invalidations mutuelles. Mome permet l'utilisation d'une consistance forte, par exemple dans les phases séquentielles, et de fournir une vision cohérente des pages sur les points de synchronisation. Ceci permet, par exemple, de garantir que les données accédées après une barrière de synchronisation intègrent toutes les modifications apportées avant cette barrière. L'utilisation des informations extraites au cours du processus de compilation permet de restreindre l'application de ces contraintes uniquement aux pages potentiellement utiles. De plus, Mome permet de déclencher spéculativement des accès non bloquants en lecture ou en écriture et donc de réduire les temps d'attente lorsqu'un processeur accède à la page pour la première fois.

MOME a été mise au point sur le calculateur Nec Cenju3 sous micro noyau Mach puis portée sur le calculateur Nec Cenju4. Une mise en oeuvre basée sur l'utilisation des protections de mémoires sous Unix a également été developpée et permet son utilisation sur des stations de travail Sparc-Solaris ou PC-Linux. Mome utilise Une couche de communication basée sur le réseau SCI pour la version PC-Linux.

Mome offre un certain nombre de fonctionnalités comme les barrières de synchronisation, les verrous, les opérations de réduction et de diffusion sur les données qui permettent son utilisation sans faire appel à des couches de communication complexes. Les derniers développements sur Mome visent à permettre la communication par partage de mémoire entre plusieurs applications.

L'intégration de la MVP Mome dans un exécutif du langage HPF est en cours d'étude, notamment dans le cadre de la collaboration Inria/Nec. Elle permettra son utilisation de manière transparente au travers d'un langage de haut niveau.

Mémoire virtuelle partagée générique

Participants : David Mentré, Thierry Priol.

Notre équipe s'intéresse depuis plusieurs années à la conception et à la réalisation de mémoires virtuelles partagées (MVP). Ces travaux ont montré la nécessité d'avoir une MVP générique, c'est à dire supportant plusieurs protocoles de cohérence. De plus, une MVP se devant d'être fiable, les protocoles utilisés doivent être vérifiés.

C'est pour répondre à ces contraintes de souplesse et de vérification que nous avons conçu un système de mémoire virtuelle partagée dont le protocole de cohérence peut être défini en utilisant un langage de haut-niveau [19, 29]. Pour la conception, les protocoles de cohérence sont décrits de manière abstraite dans un formalisme dérivé du formalisme Gamma Structuré. Cette description de haut niveau permet la vérification d'invariants indépendamment de toute instanciation particulière d'un protocole (nombre de nœuds,...). Ce travail est réalisé en étroite collaboration avec Daniel Le Métayer de l'équipe Lande. Pour la réalisation, cette description de haut niveau est dérivable en un automate compilable et exécutable au sein d'un environnement d'exécution. Cette dérivation est réalisée grâce à des choix d'implémentation fournit par le concepteur du protocole. L'incorporation de ces choix à la spécification initiale est réalisée grâce à une technique de « tissage » dérivée de la programmation par aspects (AOP : *Aspect-Oriented Programming*).

Nous réalisons actuellement un prototype, appelé Dupleix, validant cette approche. Ce prototype offre un programme de vérification d'invariants sur une spécification de haut niveau, un programme de dérivation d'une spécification en automate et un environnement d'exécution distribué de cet automate. Ce prototype fonctionne sur une grappe de PC fonctionnant avec le système d'exploitation Linux.

6.4 Technologies pour le METACOMPUTING (cf. 2.5)

Participants : Stéphane Écolivet, Stéphane Gouache, Tsunehiko Kamachi, Pascale Launay, Jean-Louis Pazat, Thierry Priol, Christophe René.

Mots clés : metacomputing, Corba, Java, SCI, réseau de stations de travail (Now), frameworks, objets, transformations de programmes, couplage de codes.

Résumé : *L'accroissement des performances des calculateurs et des réseaux permet d'envisager de nouvelles applications dans le domaine de la simulation. Il est ainsi possible de coupler plusieurs codes de calcul afin d'améliorer la qualité des résultats en prenant en compte un plus grand nombre de phénomènes physiques. L'utilisation de réseaux à hauts débits permet également d'envisager la visualisation des résultats produits par un supercalculateur quelquesoit la distance qui sépare le supercalculateur du système de visualisation. Cette activité a pour objectif de contribuer au développement de technologies qui permettent la conception d'environnement de "metacomputing". Nos travaux portent principalement sur des extensions au concept d'objets distribués en y intégrant le parallélisme, la génération automatique de code réparti, la conception de mécanismes de communication efficace entre objets distribués et la conception d'environnements logiciels pour la programmation par composants logiciels.*

Objet Corba parallèle

Participants : Thierry Priol, Christophe René.

Le but de ces travaux est d'introduire le parallélisme au sein de Corba. Nous avons conçu un nouveau type d'objets : les objets Corba parallèles [20, 34, 35]. Un objet Corba parallèle se présente sous la forme d'une collection d'objets Corba identiques. Pour l'utilisateur, un objet Corba parallèle se manipule comme un objet standard. La mise en oeuvre des objets Corba parallèles, doit permettre à un objet standard d'invoquer une méthode fournie par un objet parallèle. Elle doit également permettre à un objet parallèle d'invoquer des méthodes mises en oeuvre par d'autres objets (séquentiels ou parallèles). Nous avons étendu le langage de spécification d'interface (IDL) afin d'y introduire des moyens d'expression du parallélisme. Il s'agit notamment de spécifier la cardinalité de la collection d'objets ainsi que la distribution des données fournies en paramètre lors de l'appel d'une méthode. Lorsque l'utilisateur invoque une méthode d'un objet parallèle, celle-ci est exécuté simultanément par tous les objets de la collection.

Les invocations simultanées d'une même méthode sur tous les objets d'une collection sont masqués à l'utilisateur. Ces opérations sont effectués par un talon (*stub*) généré par le compilateur IDL que nous avons modifiés afin qu'il supporte nos extensions. Le talon est également responsable de la distribution des données sur chaque objet de la collection. La communication entre les talons (invocation à partir d'un objet Corba parallèle) est réalisée par échange de messages. La redistribution des données est gérée par la bibliothèque de fonctions Dalib conçue au GMD (Allemagne) par T. Brandes avec qui nous avons démarré une collaboration.

Pour pouvoir gérer plus facilement les objets Corba parallèles, nous avons étendu certains services Corba. Par exemple, nous avons modifiés le service de nommage pour manipuler une collection d'objet avec un nom unique. Nous avons également conçu un mécanisme pour activer à distance un objet Corba parallèle.

Le concept d'objet Corba parallèle fait l'objet d'une réponse de l'Inria [56] à un "Request For Information (RFI)" de l'OMG sur le thème "Aggregated Computing". Il s'agit d'une étape préliminaire à des extensions possibles de Corba pour exploiter des grappes de machines (support du parallélisme et équibrage des charges).

Le concept des objets Corba parallèles a été utilisé pour construire une plate-forme de simulation. Cette plate-forme a été réalisé en couplant plusieurs codes de calculs scientifiques fournis par Aérospatiale-Matra. Il sera également la base d'un environnement de Metacomputing en cours de définition dans le cadre du projet Esprit R & D Jaco3.

Transformation de programmes parallèles en programmes répartis

Participants : Pascale Launay, Jean-Louis Pazat.

Nous explorons depuis quelques années la conception et la mise en en œuvre d'outils pour la construction et l'exécution efficace de programmes répartis. A partir des résultats que nous avons obtenus dans le cadre du langage High Performance Fortran, nous étudions maintenant la généralisation de cette approche dans un contexte de programmation par objets.

Nous réalisons des transformations de programmes explicitement parallèles en des programmes parallèles et répartis. Grâce à l'utilisation d'un le langage à objets (Java), nous pouvons prendre en compte dans un même cadre la génération de code réparti par distribution de contrôle et par distribution de données.

L'approche repose sur l'utilisation de collections qui sont des objets génériques pouvant contenir tout type d'éléments (et en particulier des tâches). Dans le programme réparti que nous générons, les collections sont remplacées par des "collections distribuées" qui permettent d'accéder de manière transparente aux objets distribués. La transformation d'un programme centralisé en programme réparti se fait automatiquement en changeant la bibliothèque des collections utilisées et en transformant les composants définis par le programmeur.

La notion d'objet parallèle Corba est un cas particulier de collection distribuée. On peut donc espérer transposer la généralisation de l'expression de la distribution que nous avons faite dans le cadre Java pour l'adapter à Corba et faciliter ainsi la programmation d'applications réparties dans ce cadre.

Les résultats nouveaux que nous avons obtenus concernent :

- la définition et la mise en œuvre de transformations de programmes pour faciliter la définition de ces objets et de leurs communications.
- la définition et la mise en œuvre de supports de communication efficaces entre objets distribués,

Conception d'un ORB haute performance

Participants : Stéphane Écolivet, Jean-Louis Pazat, Thierry Priol.

Un ORB (serveur d'invocation de méthodes) constitue le cœur d'une implémentation de Corba. Il permet d'invoquer les méthodes d'un objet dont la localisation et le langage d'implémentation sont inconnus de l'appelant. La conception et la réalisation d'un ORB efficace (faible latence et fort débit) sont cruciales pour l'utilisation de la technologie Corba au sein d'environnements de metacomputing que nous souhaitons développer au sein du projet. En effet, il s'agit d'utiliser l'ORB pour faire communiquer des informations (données, contrôle) entre plusieurs codes de calcul scientifique ou bien de visualiser des informations à distance en utilisant d'autres technologies complémentaires telles que VRML et Java.

Les mises en œuvre actuellement disponibles ne permettent pas de tirer parti des performances des réseaux de stations de travail (Myrinet, SCI, etc.). Le souci des concepteurs est plutôt d'offrir l'interopérabilité entre ORB plutôt que la performance. Notre objectif est de conserver cette interopérabilité tout en offrant de nouveaux protocoles plus performants qui s'adaptent mieux aux réseaux tels que Myrinet, SCI ou tout autres réseaux rapides tels que ceux disponibles dans les machines parallèles. En vue de proposer de nouveaux protocoles, nous souhaitons tout d'abord étudier finement les performances d'un ORB existant. Nous travaillons actuellement à l'analyse fine des coûts des mécanismes mis en œuvre dans un ORB par l'instrumentation de chacune de ses fonctions. Ceci va nous permettre d'étudier sur quels point les optimisations doivent porter. Nous utilisons le système Salto développé dans le projet afin d'effectuer une instrumentation automatique du code. L'ORB étudié est celui de Mico, une implémentation de Corba sous licence GNU GPL.

Environnement de simulation distribué

Participants : Stéphane Gouache, Jean-Louis Pazat, Thierry Priol, Christophe René.

Mots clés : Simulation numérique distribuée, Corba.

Ce travail est réalisé dans le cadre du projet Esprit R & D Jaco3. Il s'agit de construire un environnement de metacomputing pour la simulation. Cet environnement, déployé sur un ensemble de supercalculateurs interconnectés par des réseaux, doit permettre le couplage de codes de calcul scientifique distribués. Des mécanismes de travail coopératif seront offerts pour permettre à des experts dans des domaines numériques particuliers de coopérer à l'analyse des résultats de la simulation. Le travail de l'Inria consiste plus particulierement à concevoir et mettre en œuvre l'architecture de l'environnement et à fournir une méthodologie ainsi qu'un ensemble d'outils destinés à faciliter l'intégration de codes de simulation numérique existants [44, 45, 46, 47]. L'architecture proposée est bâtie autour du bus logiciel Corba et s'appuie sur un ensemble de services spécifiques. Ces services permettent notamment la gestion des sessions des utilisateurs, le mouvement des données entre les codes de simulation, et le stockage des informations décrivant le couplage entre composants logiciels. Des solutions sont proposées pour faciliter l'intégration de codes de simulation existants. L'emploi du concept d'objet Corba parallèle est mis en avant pour ses performances et sa simplicité de mise en œuvre. Un outil est également développé pour permettre l'encapsulation de codes sans modifications.

7 Contrats industriels (nationaux, européens et internationaux)

7.1 Projet OMI SPEAR 2, convention n° 197C8499931398005 (12/97-07/98)

Participant : André Seznec.

Le projet OMI Spear 2 réunissant T-Square, Ace, Forth, et le projet Caps visait à fournir une première définition d'un microcontrôleur Sparc. Au cours de ce projet, Caps a fourni une expertise en architecture de processeurs et ordonnancement de codes.

6.2), convention n° 196C5350031308006 (10/96-10/99)

7.2 Projet Esprit LTR Oceans (cf. 2.3, 6.2), convention n° 196C5350031308006 (10/96-10/99)

Participants : François Bodin, Erven Rohou, André Seznec, Paul Van der Mark.

La plupart des transformations de haut niveau affectent sensiblement les possibilités d'ordonnancement de code et d'allocation de registres, sans que cela puisse être pris en compte au niveau du code source. Dans le cadre du projet LTR Oceans (en collaboration avec les universités de Leiden (contractant principal), Manchester et Versailles ainsi que la société Philips) nous explorons les schémas de compilation autorisant des rétroactions au niveau des transformations du code source afin de prendre en compte les contraintes d'ordonnancement et de registres. Le prototype que nous développons utilise le système Salto. Il est connecté aux systèmes Pilo et Lora (respectivement un ordonnanceur et un allocateur de registre pour le pipeline logiciel) développé par C. Eisenbeis (avant-projet A3 de l'Inria) pour mettre en œuvre l'ordonnancement de boucle pour architectures VLIW. Les études sont centrées sur le

processeur multimédia TriMedia-1 de la société Philips. La thèse de Erven Rohou porte sur ce sujet [14].

Fits : Fortran Integrated Tool Set (cf. 2.3, 6.2), convention n° 197C7720031308202 (6/97-6/99)

7.3 Projet Esprit R & D Fits : Fortran Integrated Tool Set (cf. 2.3, 6.2), convention n° 197C7720031308202 (6/97-6/99)

Participants : François Bodin, Stéphane Chauveau, Yann Mével.

Le projet Esprit R & D Fits est mené en collaboration avec Simulog (contractant principal), VCPC (Vienne) et les sociétés Batelle, QSW et Alenia Spazio. Il a pour objectif d'intégrer des outils d'analyse de code Fortran pour le portage de code sur architectures hautes performances. Il s'agit, entre autres, de coupler le système Foresys avec la bibliothèque de traçage pour MPI, Vampire de Pallas, ainsi que les outils de visualisation de graphe d'appel de fonctions Ida et Analyst du VCPC. Dans le cadre de ce projet nous effectuons un transfert de technologie, concernant les transformations de programme pour l'amélioration de performances. Ceci consiste, en pratique, à l'ajout de transformations de code dans le système Foresys pour l'amélioration de la localité. La technologie de base utilisée dans ce projet est celle de TSF.

(cf. 2.5), convention n° 96C318003130801 (5/96-12/98)

7.4 Projet Esprit R & D Pacha (cf. 2.5), convention n° 96C318003130801 (5/96-12/98)

Participant : Thierry Priol.

Le projet Esprit R & D Pacha, en collaboration avec Spacebel (Contractant principal), Dolphin Interconnect Solutions, Aerospatiale, Novartis, Intecs, Telmat et l'université de Bale et FhG a pour objectif la construction d'une plateforme de haute performance exploitant le concept client/serveur. Cette plateforme utilise la technologie d'interconnexion SCI développée par Dolphin Interconnect Solutions. Au sein de ce projet, un ORB (Object Request Broker) Corba est en cours de développement. Il exploite les capacités du réseau SCI pour améliorer de façon sensible les performances de l'ORB. Cette implémentation haute performance de Corba est utilisée par les autres partenaires du projet au sein d'applications et d'environnements de programmation pour le calcul haute performance. Le projet Caps est responsable du développement d'un exécutif (Cobra) pour l'exécution de composants logiciels parallèles. Le projet Caps développe un générateur de code Fortran, inspiré des travaux autour de Fortran- S [48, 50, 49]. T. Priol est responsable, à l'Irisa, du projet HPCN Pacha (convention n° 96C318003130801).

7.5 Projet Esprit TTN Pro HPC, convention n° 97C4700031308005 (3/97-3/99)

Participant : Thierry Priol.

Le projet Esprit TTN Pro HPC est un nœud de transfert technologique dans le domaine du calcul et des réseaux de haute performance (HPCN). Il vise à animer des activités de trans-

fert technologique à destination principalement des petites et moyennes entreprises. Il s'agit de montrer en quoi l'apport de la technologie HPCN permet de résoudre des problèmes que l'informatique classique ne peut pas traiter dans des délais raisonnables. Les partenaires de ce nœud de transfert technologique sont l' ENS-Lyon/Lip (contractant principal), Matra et Simulog. Le rôle de l'Inria est, comme les autres partenaires, de rechercher des activités de transfert technologique. Des activités dans le domaine des télécommunications sont en cours de montage. Il assure également les activités de dissémination (serveur Web, campagne de presse, plaquette, posters,...). Le TTN Pro HPC gère actuellement dix activités de transfert technologique dans des domaines très variés. L'Inria est associé directement à deux de ces activités (domaine de la finance/assurance et simulation de tête de forage). H. Leroy, de l'Atelier, participe activement à l'animation de Pro HPC. T. Priol est le directeur-adjoint de ce nœud de transfert technologique.

7.6 Working Group Esprit EuroTools (WG 27141), convention n° 98C1790031308005 (04/98-04/00)

Participant : Jean-Louis Pazat.

Jean-Louis Pazat est le coordinateur du groupe de travail Européen EuroTools sur les outils pour le calcul hautes performances parallèle et distribué. Ce *Working Group* est financé par la communauté Européenne depuis avril 1998. Il comprend des industriels, des grands centres de calcul Européens et des universitaires spécialistes des outils pour le calcul parallèle et distribué. Le but de ce groupe est d'une part de promouvoir au niveau international auprès des utilisateurs les travaux réalisés dans le domaine des outils pour le parallélisme en Europe et d'autre part d'évaluer l'impact et les évolutions de ces technologies. Nous organisons des conférences, des démonstrations et séances de "travaux pratiques". Nous avons notamment organisé des Workshops lors des conférences HPCN, Ecoop et EuroPar en 1998 ; nous avons participé et soutenu les workshops des groupes d'utilisateurs Européens d'une part de PVM et MPI et de HPF d'autre part. Un site Web a été mis en place pour le recensement des outils existants développés en Europe.

7.7 Projet Esprit Jaco3, convention n° 98C3760031308005 (11/98-04/01)

Participants : Stéphane Gouache, Jean-Louis Pazat, Thierry Priol, Christophe René.

Le projet Esprit Jaco3, en collaboration avec Aerospatiale-Matra (Contractant principal), Alcatel, KTH, ESB, Intecs, Allgon a pour objectif la construction d'un environnement de metacomputing pour la simulation. Cet environnement permet le couplage de codes de calcul scientifique dans un environnement constitué de supercalculateurs interconnectés par des réseaux. Cet environnement offrira des mécanismes de travail coopératif permettant à des experts dans des domaines numériques particuliers de coopérer à l'analyse des résultats de la simulation. Cet environnement est fondé sur le concept de composants logiciels. L'implémentation s'appuiera sur les technologies Corba et Java. L'Inria est responsable de la conception de l'architecture de l'environnement de metacomputing et de sa mise en oeuvre. Plus particulièrement, l'architecture developpée par l'Inria s'appuie sur un ensemble de services permettant

la gestion des sessions des utilisateurs, le mouvement des données entre les codes de calcul, ainsi que la gestion d'un repertoire central contenant la description des applications de simulation. Il s'agira notamment d'intégrer des codes de calcul scientifique existants, ainsi que d'exploiter le concept d'objet Corba parallèle au sein de cet environnement permettant ainsi un transfert technologique. T. Priol est responsable, à l'Irisa, du projet HPCN Jaco3 (convention n° 98C3760031308005).

7.8 Contrat Aérospatiale Matra, convention n° 98C4460031308011 (02/99-07/99)

Participants : Thierry Priol, Christophe René.

Depuis plusieurs années, nous avons démarré une collaboration avec le centre de recherche de l'Aérospatiale Matra à Suresnes. Ce centre développe notamment des codes numériques pour des architectures parallèles. La collaboration avec l'Aérospatiale Matra a commencé dans le cadre du projet Esprit R&D Pacha (voir section 7.4) et continue dans le cadre du projet Esprit Jaco3 (voir section 7.7). Un accord de collaboration a été signé avec cette société dans le domaine du couplage de codes numériques. Il s'agit ici d'utiliser le concept d'objet CORBA parallèle pour coupler deux codes numériques parallèles sur une grappe de calculateurs homogènes (réseau de PCs) [51]. Ce contrat a deux intérêts: tout d'abord "valider" la technologie objet Corba parallèle sur un exemple concret et susciter de nouvelles idées de recherche dans le domaine des outils pour le couplage de codes.

7.9 Nec (cf. 2.3, 2.5, 6.2)

Participants : Yvon Jégou, Jean-Louis Pazat, Thierry Priol.

T. Priol est responsable d'un contrat de recherche d'une durée de deux ans (avril 1999/mars 2001) avec la société Nec impliquant plusieurs projets de l'Inria : projet Caps à Rennes et projet Gamma à Rocquencourt. La machine parallèle Cenju3 mise à la disposition des chercheurs de l'Inria par la société Nec a été remplacée par une machine Cenju4 plus performance au printemps 1999. Cette machine est dotée de 16 processeurs R10000. Elle offre des mécanismes de communication très variées permettant de nombreuses expérimentations (échange de message, adressage à distance, adressage global). Le rôle du projet Caps est d'expérimenter une mémoire virtuelle partagée comme support d'exécution pour des compilateurs HPF et d'expérimenter le concept d'objet Corba parallèle sur ce type d'atchitecture. La mémoire virtuelle partagée Mome a été développée sur Cenju3 puis portée sur la machine Cenju4. Les premiers tests de performance confirment que l'approche mémoire virtuelle partagée proposée est réaliste même sur des codes réguliers. La société Nec a mis un chercheur, Tsunehiko Kamachi, à la disposition de l'Inria, pour une durée d'un an, pour participer à cette collaboration.

7.10 Projet SoftIT, convention n° 198A4110000MPR012 (01/98-06/99)

Participant : Thierry Priol.

Le projet Soft-IT fait suite à l'appel d'offre de la Commission Européenne (No III/97/31) pour fournir un support scientifique et technique dans le domaine des environnement de simulation. Ce projet est coordonné par Simulog avec comme autre partenaire Thomson-CSF. Ce projet a pour but de concevoir un environnement de simulation permettant l'interopérabilité des outils HPCN. Le rôle de l'Inria consiste à effectuer un survey des environnements de simulation existants qui ont été réalisés dans le cadre de projets Européens. Ce travail a été effectué en collaboration avec Stéphane Lantéri du projet Sinus [54]. Il consiste également à apporter son expertise dans la technologie Corba pour la réalisation d'un démonstrateur sous la maîtrise d'oeuvre de Thomson-CSF[53, 52].

7.11 System level Methods and Tools Medea n° 199C9010031308011 (1999-2000)

Participants : François Bodin, Pierre Michaud, André Seznec.

Dans ce projet, en collaboration avec le projet A3 de l'Inria Rocquencourt, nous concevons un module reciblable de recherche des séquences d'instructions - ou d'expressions - susceptibles d'être remplacées par des instructions multimédia vectorisées. C'est actuellement un point bloquant pour une exploitation efficace des nouvelles générations de DSP hautes performances.

7.12 Etude relative à la parallèlisation de modèle dynamique océanique, Epshom n° 198C5760031308061 (1998-1999)

Participants : François Bodin, Stéphane Chauveau.

Cette étude, avec l'Epshom, a porté sur la parallélisation d'un code Adjoint du code de modélisation de l'Atlantique nord Micom. Cette étude a servi, entre autres, à la validation du système TSF.

microprocesseurs multithreadés pour le multimédia, CEA, (1999-2002)

7.13 modélisation d'architecture de microprocesseurs multithreadés pour le multimédia, CEA, (1999-2002)

Participants : Pierre Michaud, André Seznec.

La convention vise à explorer les concepts architecturaux des microprocesseurs multithreadés pour le domaine des applications multimédia, ainsi que les mécanismes de compilation et d'optimisation de code qui peuvent leur être destinés. La première étape consiste en l'analyse, la conception et la réalisation d'un simulateur d'architectures multithreadées adaptable en fonction des évolutions de la conception architecturale. La seconde étape consiste en l'analyse des variantes architecturales et de leur contrepoint logiciel, dans un souci d'optimisation de la performance orientée sur le domaine des applications multimédia (incluant les aspects traitement des images, traitement du son, bases de données, distribution vidéo, etc.).

8 Actions régionales, nationales et internationales

8.1 Actions régionales

Les thèses de Thierry Lafage et Jonathan Perret sont partiellement financées par la région Bretagne.

8.2 Actions nationales

T. Priol est le coordinateur de l'action coopérative Rescapa. Cette action soutenue par la direction scientifique de l'Inria vise à expérimenter et à évaluer la technologie d'interconnexion SCI pour la construction de serveurs à hautes performances. Plusieurs équipes sont partenaires de cette action : quatre projets Inria (Apache, Caps, ReMaP, Sirac), et l'équipe Espace du LIFL (Lille). L'objectif de l'action est de développer des services systèmes et des environnements de programmation en vue de l'exploitation des mécanismes d'accès direct à une mémoire distante fournis par des réseaux à capacité d'adressage (SCI). L'approche proposée est fondée sur une conception conjointe (co-design) des mécanismes liés au système et de ceux liés à l'environnement de programmation.

8.3 Actions financées par la Commission Européenne

Yvon Jégou est responsable du projet Intas *Compilers and Programming Environments for Parallel Computers* auquel participent l'Inria en France, le RAL (en Grande Bretagne), l'Institute for System Programming ainsi que le Keldysh Institute of Applied Mathematics à Moscou. Contrat Intas 95–369.

T. Priol est co-responsable, au sein d'Ercim, d'une initiative visant à mettre en relation des industriels européens et chinois dans le domaine du calcul et des réseaux haute performance (HPCN).

8.4 Relations bilatérales internationales

Yvon Jégou est responsable, avec Victor Ivannikov de l'Institute for System Programming à Moscou, du projet *Boîte à outils pour la mise au point de programmes Java à parallélisme de données sur des multicalculateurs parallèles* de l'institut franco-russe Liapunov. Dans ce cadre, nous avons accueilli Mikhail V. Domratchev de l'Institut Liapunov de Moscou au sein de l'équipe Caps pour une durée de deux semaines. Durant cette période, nous avons développé des adaptateurs parallèles à la bibliothèque DPJ (Data Parallel Java) développée au sein de l'institut Liapunov et analysé cette approche. Ceci a permis la parallélisation de programmes Java utilisant la bibliothèque de *containers* et remplaçant cette dernière par la DPJ.

9 Diffusion de résultats

9.1 Animation de la communauté scientifique

F. Bodin a été membre du comité de programme de la conférence Europar'99.

Y. Jégou est membre du comité de rédaction de la revue TSI.

J-L. Pazat anime le groupe de travail français "Grappes" du GdR ARP dont les activités sont centrées sur l'utilisation des réseaux de stations de travail pour le calcul hautes performances.

T. Priol est membre des comités de lecture des revues Calcultateurs Parallèles et International Journal of High Performance Computer Graphics Multimedia and Visualisation. Il a été également membre du comité de programme du "Fourth International Workshop on High-Level Parallel Programming Models and Supportive Environments (HIPS'99)", Parallel Visualisation and Graphics (PVG'99).

9.2 Enseignement universitaire

F. Bodin et P. Michaud interviennent dans les cours d'architectures et compilation du DEA informatique et du DIIC de l'université de Rennes I.

F. Bodin intervient dans un cours sur les architectures de processeurs à l' ENST de Bretagne.

Christine Morin a donné un cours sur les systèmes distribués dans le cadre du DEA calcul intensif de l'université libanaise à Beyrouth au Liban en janvier 1999.

J.-L. Pazat est responsable du cours de méthodes de programmation parallèle à l'Insa de Rennes.

Nous avons organisé et dispensé à Thomson multimédia 50 heures de cours sur les architectures des microprocesseurs et les techniques d'optimisations de code associées (F. Bodin, P. Michaud et A-M. Kermarrec).

9.3 Autres enseignements

Nous avons organisé et dispensé à Thomson multimédia 50 heures de cours sur les architectures des microprocesseurs et les techniques d'optimisations de code associées (F. Bodin, P. Michaud et A-M. Kermarrec).

9.4 Participation à des colloques, séminaires, invitations

Outre les conférences et workshops donnant lieu à publication des actes listés dans la bibliographie, les membres du projet Caps ont présenté leurs travaux dans les séminaires ou workshops suivants :

- Pierre Michaud a présenté le tutoriel sur la prédiction de branchements au 5ème Symposium en Architectures Nouvelles de Machines à Rennes le 8 juin 1999.
- Pierre Michaud a présenté "Exploring Instruction-Fetch Bandwidth Requirement in Wide-Issue Superscalar Proocessors" au groupe Compaq/VSSAD à Shrewsbury MA (18 octobre 99).
- Christine Morin a donné un séminaire intitulé "An Efficient and Scalable Approach for Implementing Fault Tolerant DSM Architectures" à Rice University (USA) en février 99 et à Rudgers University (USA) en décembre 1999.
- Thierry Priol a été invité à faire une présentation lors de la conférence "Problem-Solving Environments: Infrastructure and Prototypes" (European Research Conference) en juin 1999 et à la conférence PDC, organisée par le KTH (Suède), en décembre 1999.

– Du 15 au 21 mars F. Bodin a été invité a faire des présentations dans les universités de Tsing-Hua, NSYSU et Zone-Chen à Taiwan.

9.5 Divers

Thierry Priol participe aux travaux de préparation de standardisation de l'OMG (RFI "Aggregated Computing").

A. Seznec a participé à un audit R & D de STMicroelectronics dans le cadre d'un contrat pluriannuel entre le Ministère de l'Industrie et STMicroelectronics.

A. Seznec est membre du jury du concours SGI-Cray 1998 et du jury du prix de la meilleure thèse Specif.

Y. Jégou est membre du conseil scientifique de l'institut franco-russe Liapunov.

10 Bibliographie

Ouvrages et articles de référence de l'équipe

[1] F. ANDRÉ, M. L. FUR, Y. MAHÉO, J.-L. PAZAT, « The Pandore Data Parallel Compiler and its Portable Runtime », *in : High-Performance Computing and Networking*, LNCS 919, Springer Verlag, p. 176–183, Milan, Italy, mai 1995.

[2] F. BODIN, P. BECKMAN, D. GANNON, J. SRINIVAS, « Sage++ : a class library for building Fortran and C++ restructuring tools », *Proceedings of the Second Object-Oriented Numerics Conference*, avril 1994.

[3] F. BODIN, Z. CHAMSKI, C. EISENBEIS, E. ROHOU, A. SEZNEC, « GCDS: a compiler strategy for trading code size against performance in embedded applications », *rapport de recherche n° 3346*, Inria, janvier 1998.

[4] F. BODIN, W. JALBY, C. EISENBEIS, D. WINDHEISER, « Window-based register allocation », *Code Generation - Concepts, Tools, Techniques, Proceedings of the International Workshop on Code Generation*, 1991, p. 119–145.

[5] F. BODIN, L. KERVELLA, T. PRIOL, « Fortran-S : a fortran interface for shared virtual memory architectures », *in : Proceedings of Supercomputing*, IEEE Computer Society Press (éditeur), p. 274–283, novembre 1993.

[6] F. BODIN, A. SEZNEC, « Skewed associativity improves performance and en hances predictability », IEEE *Transactions on Computers*, mai 1997.

[7] C. EISENBEIS, W. JALBY, D. WINDHEISER, F. BODIN, « A strategy for array management in local memory », *Journal of Mathematical Programming*, 63, 1994, p. 331–370.

[8] S. HILY, A. SEZNEC, « Standard memory hierarchy does not fit simultaneous multithreading », *in : Proceedings of the Workshop on Multithreaded Execution, Architecture and Compilation (MTEAC' 98)*, Las Vegas, février 1998.

[9] A.-M. KERMARREC, C. MORIN, M. BANÂTRE, « Design, Implementation and Evaluation of ICARE », *Software Practice and Experience*, 1998.

[10] Z. LAHJOMRI, T. PRIOL, « KOAN: A Shared Virtual Memory for iPSC/2 Hypercube », *in : Proc. of the 2nd Joint Int'l Conf. on Vector and Parallel Processing (CONPAR'92)*, p. 441–452, septembre 1992.

[11] P. MICHAUD, A. SEZNEC, S. JOURDAN, P. SAINRAT, « Alternative schemes for high-bandwidth instruction fetching », *rapport de recherche n° 1180*, Irisa, mars 1998.

[12] P. Michaud, A. Seznec, R. Uhlig, « Trading conflict and capacity aliasing in condit ional branch predictors », *in : Proceedings of the 24th International Symposium on Computer Architecture*, ieee-acm (éditeur), Denver, juin 1997.

[13] P. Michaud, *Chargement des instructions sur les processeurs superscalaires*, Thèse de doctorat, université de Rennes I, novembre 1998.

[14] E. Rohou, *Infrastructures et stratégies de compilation pour parallélisme à grain fin*, Thèse de doctorat, université de Rennes I, novembre 1998.

[15] A. Seznec, S. Jourdan, P. Sainrat, P. Michaud, « Multiple-block ahead branch predictors », *in : Proceedings of the 7th conference on Architectural Support for Programming Languages and Operating Systems*, octobre 1996.

Thèses et habilitations à diriger des recherches

[16] Y. Mével, *Environnement pour le portage de codes orienté performance sur machines parallèles et monoprocesseurs*, Thèse de doctorat, université de Rennes I, mars 1999.

Articles et chapitres de livre

[17] A. H. J. Bigham (éditeur), *Software Agents for Future Communications Systems*, Springer verlag, 1999, ch. Mobile Agents for Managing Networks: the Magenta perspective.

[18] S. Chauveau, F. Bodin, « Menhir: An environment for high performance Matlab », *Scientific Computing*, 1999.

[19] D. Mentré, D. Le Métayer, T. Priol, « Conception de protocoles de cohérence de MVP par traduction d'une spécification Gamma à l'aide d'aspects », *Calculateurs parallèles, réseaux et systèmes*, 1999.

[20] T. Priol, C. René, G. Alléon, « SCI-based Cluster Computing », *Springer LNCS State of the Art Surveys*, Springler Verlag, 1999, ch. Programming SCI Clusters using Parallel CORBA Objects, p. To appear, `http://www.irisa.fr/paris/biblio/Fichiers-PS/Thierry-Priol/lncs99.ps.gz`.

Communications à des congrès, colloques, etc.

[21] M. Barreteau, F. Bodin, al., « OCEANS: Optimising Compilers for Embedded Applications », *in : Euro-Par'99*, Toulouse, août 1999.

[22] F. Bodin, Y. Mevel, S. Chauveau, E. Rohou, « Parallelizing an Ocean code using TSF », *in : Proceedings of the Twelfth International Workshop on Languages and Compilers for Parallel Computing (LCPC'99)*, The University of California, San Diego, La Jolla, CA USA, août 1999.

[23] B. Chapman, F. Bodin, L. Hill, J. Merlin, G. Viland, F. Wollenweber, « FITS - A Light-Weight Integrated Programming Environment », *in : Euro-Par'99*, Toulouse, août 1999.

[24] B. Chapman, J. Merlin, D. Pritchard, F. Bodin, Y. Mevel, T. Sørevik, L. Hill, « Tools for Development of Programs for a Cluster of Shared Memory Multiprocessors », *in : International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'99)*, 1999.

[25] S. Hily, A. Seznec, « Out-Of-Order Execution May Not Be Cost-Effective on Processors Featuring Simultaneous Multithreading », *in : proceedings of HPCA-5*, Orlando, USA, 1999.

[26] T. Kisuki, P. Knijnenburg, M. O'Boyle, F.Bodin, H. Wijshoff, « A Feasibility Study in Iterative Compilation », *in : International Symposium on High Performance Computing*, Kyoto, Japon, mai 1999.

[27] T. Lafage, A. Seznec, E. Rohou, F. Bodin, « Code Cloning Tracing: A "Pay per Trace" Approach », *in : Euro-Par'99*, Toulouse, août 1999.

[28] R. Lottiaux, C. Morin, T. Priol, « File Mapping in Shared Virtual Memory using a parallel file system », *in : Proc. of ParCo99*, août 1999.

[29] D. Mentré, D. Le Métayer, T. Priol, « Towards designing SVM coherence protocols using high-level specifications and aspect-oriented translations », *in : Proceedings of the 1999 Workshop on Software Distributed Shared Memory, held in conjunction with 1999 International Conference in Supercomputing*, p. 101–107, Rhodos, Greece, 1999.

[30] P. Michaud, A. Seznec, S. Jourdan, « Exploring Instruction-Fetch Bandwidth Requirement in Wide-Issue Superscalar Processors », *in : International Conference on Parallel Architectures and Compilation Techniques*, 1999.

[31] C. Morin, R. Lottiaux, « Global Resource Management for High Availability and Performance in a DSM-based cluster », *in : Proc. of 1st workshop on Software Distributed Shared memory*, juin 1999.

[32] C. Morin, « Global and Integrated Processor, Memory, and Disk management in a Cluster of SMP's », *in : Proc. of workshop on cluster based computing*, ACM, juin 1999.

[33] T. Priol, « High Performance Computing for Image Synthesis (Invited Talk) », *in : Lecture Notes in Computer Science, 1573*, p. 563–578, 1999.

[34] C. René, T. Priol, « Concept d'objet CORBA parallèle », *in : Renpar'11*, juin 1999.

[35] C. René, T. Priol, « MPI Code Encapsulation using Parallel CORBA Object », *in : Proceedings of the Eighth IEEE International Symposium on High Performance Distributed Computing*, IEEE, p. 3–10, aout 1999.

[36] A. Sahai, C. Morin, « Mobile Agents for Location Independent Computing », *in : Proceedings of ACM Symposium on Applied Computing*, San Antonio, USA, mars 1999.

Rapports de recherche et publications internes

[37] D. Mentré, D. L. Métayer, T. Priol, « Towards designing SVM coherence protocols using high-level specifications and aspect-oriented translations », *rapport de recherche n° 1262*, Irisa, 1999.

[38] C. Morin, R. Lottiaux, « Global Resource Management for High Availability and Performance in a DSM-based cluster », *Rapport de recherche n° 3694*, INRIA, juin 1999.

[39] C. Morin, R. Lottiaux, « Global Resource Management for High Availability and Performance in a DSM-based Cluster », *publication interne n° 1251*, IRISA, juin 1999.

[40] T. Priol, C. René, G. Alléon, « Programming SCI Clusters using Parallel CORBA Objects », *rapport de recherche n° 1239*, Irisa, 1999.

[41] T. Priol, C. René, « Efficient Support of MPI-based Parallel Codes within a CORBA-based Software Infrastructure », *rapport de recherche n° 1258*, Irisa, 1999.

[42] C. René, T. Priol, « MPI Code Encapsulation using Parallel CORBA Object », *rapport de recherche n° 1238*, Irisa, 1999.

[43] A. Seznec, P. Michaud, « Dealiased Hybrid Branch Predictors », *rapport de recherche n° 1229*, Irisa, 1999.

Divers

[44] « Deliverable D1.1 - Application requirements », JACO3, 1999.

[45] « Deliverable D1.2 - JACO3 General Architecture », JACO3, 1999.

[46] « Deliverable D2.1 - Application Development Guidelines », JACO3, 1999.

[47] « Deliverable D2.3 - Distributed resource manager specification », JACO3, 1999.

[48] « Deliverable D5180 - Adaptation and porting of Runtime to platform D », PACHA, 1999.

[49] « Deliverable D6130 - Performance evaluation of FORTRAN pre-compiler », PACHA, 1999.

[50] « Delivererable D6140 - Coupling Fortran-S with the Intecs IDL2F90 compiler », PACHA, 1999.

[51] « Rapport final de contrat: techniques de couplage de codes en utilisant CORBA », Aerospatiale Matra, 1999.

[52] « Recommendations », SOFT-IT, 1999.

[53] « SOFT-IT Demonstrator », SOFT-IT, 1999.

[54] « Software interoperability and platform independence: the next generation of simulation environments », SOFT-IT, 1999.

[55] R. L. C. Morin, « File Mapping as a New Parallel File System Interface », 8th Workshop on Scalable Shared Memory Multiprocessors, avril 1999, abstract.

[56] T. Priol, « Efficient support of MPI-based parallel codes within a CORBA-based software infrastructure », *in: Response to the Aggregated Computing RFI from the OMG, Document orbos/99-07-10*, juillet 1999.