

Utilisations des compteurs matériels dans les multiprocesseurs : estimation de l'accélération et contrôle-commande d'exécution

Mauricio Pillon
Doctorant CNPq - Bresil,
Laboratoire ID-IMAG (UMR 5132), Projet APACHE (CNRS/INPG/INRIA/UJF)

INPG antenne de Monbonnot - ZIRST, 51 avenue Jean Kuntzmann
38330 Monbonnot - France
Mauricio.Pillon@imag.fr

Résumé

Cet article présente des résultats sur l'utilisation des compteurs matériels présents sur les processeurs et pour les machines multiprocesseurs à faible coût. La première utilisation consiste à déduire l'accélération obtenue sur les applications parallèles de l'observation du débit de certains événements. Cette estimation est ensuite utilisée pour piloter un mécanisme de contrôle-commande afin d'augmenter le rendement d'exécution des multiprocesseurs.

Mots-clés : compteurs matériels, multiprocesseurs à faible coût, hiérarchie mémoire, évaluation de performances, contrôle-commande d'exécution.

1. Introduction

La décennie précédente a vu l'apparition de compteurs d'événements matériels sur l'ensemble des nouveaux processeurs, notamment pour la famille x86. L'ensemble des événements permettent de tracer précisément l'activité des machines. On peut ainsi compter : le nombre d'instructions exécutées (par type), les opérations de lecture et écriture, de succès et d'échecs pour chaque cache, d'accès/échecs au TLB...

Principalement utilisés pour l'analyse de performance d'application, ces compteurs ont comme principales propriétés : la finesse de mesure (à l'événement près) et un coût de mesure (comptage) nul puisque intégré dans le processeur.

Du fait de la faible intrusion de la gestion des compteurs il est possible d'envisager le développement de nouveaux outils pour l'optimisation des performances, opérant lors de l'exploitation des machines. Les travaux qui sont présentés dans cet article ont cette finalité.

Après une présentation des travaux autour de l'utilisation des compteurs matériels, nous faisons état de certaines limitations intrinsèques des hiérarchies mémoires des biprocesseurs actuels. La section suivante aborde le problème de l'estimation de l'accélération d'applications parallèles en mémoire partagée. Puis nous proposons une approche originale d'utilisation de cette estimation qui vise à l'augmentation du rendement global des multiprocesseurs. Finalement nous concluons cet article en présentant les travaux futurs ainsi que les problèmes soulevés.

2. Utilisation des compteurs matériels de performances

L'ensemble des grands constructeurs de machines hautes performances ont intégré dans leurs logiciels de *profiling* [15, 6, 1, 14, 4] l'accès aux compteurs d'événements matériels. Ceci permet de détecter les éventuels problèmes de performance et de cibler l'endroit des programmes où porter les efforts d'optimisation. Parmi les cas usuels on peut citer : les problèmes sur la hiérarchie mémoire (exemple : taux élevés d'échecs sur les caches), un faible nombre d'opérations de calculs flottants ou entiers et la présence d'un taux élevé de transactions liées au protocole de cohérence de caches dans les multiprocesseurs.

Les outils de première génération offraient un ensemble de données brutes du contenu des compteurs à l'utilisateur. Des outils récents [6] [7] proposent des traitements de plus haut niveau : ensemble plus complet de statistiques, instrumentation automatique, support du multi-flot, et outils pour la visualisation.

Un problème important limitant l'utilisation des compteurs est leur hétérogénéité selon les différents processeurs. En effet généralement leurs nombres diffèrent ainsi que les types d'événements accessibles et les modes d'accès aux valeurs. Par exemple la série de processeurs P6 d'Intel possède 2 compteurs de 40 bits alors que la famille Pentium 4 en comporte 18 de même largeur.

Deux projets PCL [2] et PAPI [8, 12] adressent ce problème et proposent une interface de programmation (API) unique pour un ensemble de processeurs. A l'heure actuelle, le projet PAPI permet d'accéder aux comptages d'un ensemble d'événements communs ainsi qu'aux événements spécifiques de chaque processeur. De plus comme dans [15] une fonction de multiplexage permet de simuler plus de compteurs qu'il n'est, au prix de l'introduction d'une certaine imprécision dans les mesures.

Parmi les autres utilisations des compteurs matériels on trouve des travaux autour d'outils pour la compilation qui opèrent sur les choix d'ordonnancement des instructions [11].

3. Évaluation des hiérarchies mémoires de biprocesseurs à faible coût

Les performances des machines multiprocesseurs comme les machines monoprocesseurs dépendent de l'équilibre entre les principales caractéristiques des composants constituant. La hiérarchie mémoire en est généralement considérée comme la clé de voûte. Réduire la pression sur cet ensemble fait l'objet de nombreuses études dans le domaine de l'architecture des machines. Critique pour les machines monoprocesseurs il l'est d'autant plus pour les machines multiprocesseurs.

La hiérarchie mémoire se compose de l'ensemble des caches L1 et L2 généralement situés dans le processeur, voire d'un cache de 3ème niveau, du bus d'accès à l'extérieur appelé bus système, du chipset qui est un ensemble de circuits qui arbitrent les différents accès à la mémoire, au bus d'entrées / sorties et d'un bus video. Et enfin de la mémoire principale qui peut être organisée par bancs entrelacés pour en augmenter le débit. La qualité de la hiérarchie mémoire dépend donc d'un grand nombre de facteurs.

La figure 1 présente les architectures simplifiées des machines biprocesseurs à faible coût des constructeurs Intel (figure 1-a) et AMD (figure 1-b). On note deux approches différentes, la première autour d'un bus système partagé par les processeurs (Intel), et la seconde avec une liaison point-à-point pour chaque processeur vers le chipset qui doit alors se comporter comme un commutateur pour l'accès à la mémoire ou aux bus E/S. Si la première approche suggère clairement une sensibilité possible au problème de contention sur le bus système partagé, il faut souligner que la deuxième approche impose une complexité supérieure pour le chipset (commutateur) et reporte le problème de la contention sur le couple bus mémoire/mémoire principale.

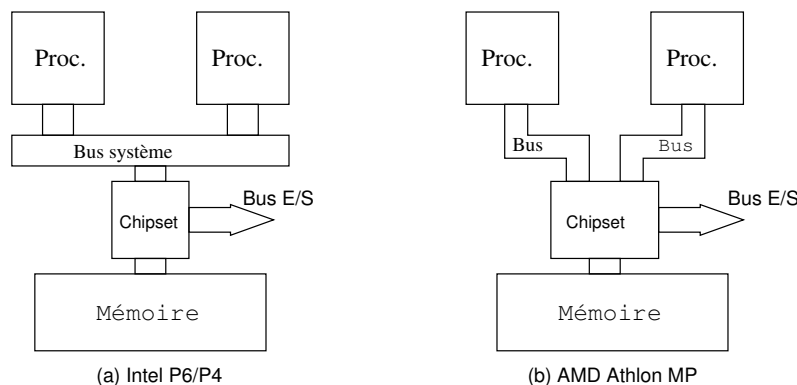


FIG. 1 – Les architectures générales et simplifiées des machines biprocesseurs à faible coût des constructeurs Intel (a) et AMD (b).

Nous nous contentons ici de simples mesures pour montrer les limitations intrinsèques des machines bi-processeurs à faible coût. Pour se faire nous utilisons le programme test STREAM [13] qui consiste à parcourir 2 ou 3 tableaux de nombres flottants de 8 octets en effectuant des opérations arithmétiques simples (copie, addition, multiplication, addition-multiplication). La taille de l'ensemble de travail (*working set*) évolue linéairement avec la taille des tableaux considérés. Nous avons aussi évalué un sous-ensemble des programmes tests NAS empruntés à l'étude [10]. Le paradigme de parallélisation en mémoire partagée utilisé dans cette étude est le standard OpenMP [5] basé sur l'insertion de directives de compilation.

Les tests ont été effectués sur 3 multiprocesseurs différents : PentiumPro 200Mhz, Pentium II 450Mhz et Athlon MP 1.2Ghz. Ces machines appartiennent à trois générations différentes. Seul le multiprocesseur PentiumPro possède 4 processeurs. Le tableau 1 présente les principales caractéristique de la hiérarchie mémoire de ces machines.

	Fréq.	L1 instr taille	L1 données taille	L2 Mémoire taille/freq.	Bus système Débit	Mémoire Débit
PentiumPro	200Mhz	8Ko	8Ko	512Ko/100Mhz	528Mo/s	528Mo/s
Pentium II	450Mhz	16Ko	16Ko	512Ko/225Mhz	800Mo/s	800Mo/s
Athlon MP	1.2Ghz	64Ko	64Ko	256Ko/1.2Ghz	2,1Go/s	2,1Go/s

TAB. 1 – Caractéristiques principales de la hiérarchie mémoire des multiprocesseurs utilisés.

Le tableau 2 présente les accélérations obtenues pour chaque machine. Les tableaux pour le programme test STREAM, possèdent environ 16 millions d'éléments soit 384 Mo de mémoire, ce qui est de plusieurs ordres de grandeurs supérieur à la capacité des caches de seconds niveaux. Pour les programmes NAS la taille des données traitées correspond à la version A. Pour l'ensemble des programmes, nous avons utilisé le compilateur *pgi v3.2-4* de Portland Group avec les directives de compilation *-O2 -fast -mp*.

	PentiumPro 200Mhz (2 proc.)	PentiumPro 200Mhz (4 proc.)	Pentium II 450Mhz (2 proc.)	Athlon MP 1.2Ghz (2 proc.)
Copy (a=b)	1,05	1,00	1,20	1,18
Scale (a=Cst*b)	1,20	1,13	1,20	1,16
Add (a=b+c)	1,00	0,99	1,24	1,34
Triad (a= a+b*c)	1,01	0,95	1,24	1,34
NAS-SP	1,15	1,17	1,35	1,30
NAS-CG	1,27	1,20	1,60	1,79
NAS-EP	1,98	3,94	1,97	1,98

TAB. 2 – Accélérations obtenues sur 3 générations de machines multiprocesseurs avec les programmes tests STREAM et un sous-ensemble de programmes NAS parallélisés suivant le paradigme en mémoire partagée OpenMP [10].

Les résultats sur le programme STREAM montrent que les performances en terme d'accélération si elles ont nettement évolué entre la première et la seconde génération (20 à 34 %), elles n'ont pas ou peu évolué sur les 2 dernières générations. De plus les résultats sont assez loin de l'optimal. On notera les piètres performances de la première génération.

Les résultats sur les programmes NAS montrent un éventail important de performances : proche de l'optimum avec EP et des performances assez faible avec SP. Il a été montré dans [9] avec une parallélisation quelque peu différente, que la limite est dûe à la capacité de la hiérarchie mémoire.

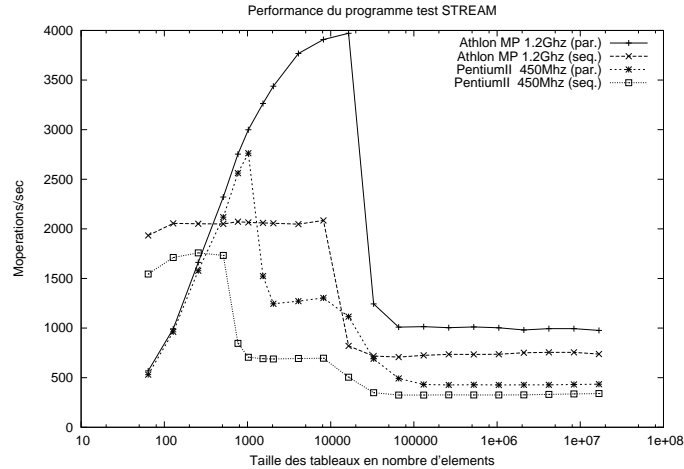


FIG. 2 – Évolution des performances, en séquentiel et parallèle, du nombre d’opérations/sec du programme STREAM ($a[i] = b[i] + cst * c[i]$) suivant la taille des tableaux pour différentes architectures.

La figure 2 présente l’évolution des performances séquentielles et parallèles, du nombre d’opérations/sec du programme STREAM ($a[i]=b[i]+cst*c[i]$) suivant la taille des tableaux. Les courbes montrent d’une part les performances brutes obtenues par les différentes configurations et d’autre part l’intérêt de disposer de caches L2 de tailles importantes.

Les faibles performances obtenues en parallèle au début des courbes (droites ascendantes) proviennent des rapports entre les coûts des appels aux fonctions OpenMP et la complexité des calculs pour les tailles de tableaux considérées.

Hormis une différence de performances absolues, on distingue clairement une différence d’allure entre les deux architectures. Pour la machine à base de Pentium II sur la courbe séquentielle on observe trois paliers dont les deux transitions sont le résultat du dépassement de capacité des caches L1 puis L2. S’il n’y a que 2 paliers pour la courbe analogue sur la machine à base d’Athlon, cela est dû à la fréquence de fonctionnement du cache L2 qui est la même que celle du cache L1 (cf. tableau 1).

Finalement on perçoit le décalage horizontal entre les courbes parallèles et séquentielles qui est la conséquence du doublement de taille de cache accumulé en parallèle.

Ces résultats dans leur ensemble rappellent pourquoi les constructeurs proposent des processeurs pour les architectures multiprocesseurs avec des caches L2 de tailles importantes (1Mo ou 2Mo), malheureusement pour un coût généralement important.

En conclusion de cette section, on peut retenir que malgré les avancées technologiques aussi bien sur les processeurs, les chipsets, les cartes mères et la mémoire, les limitations restent présentes pour certains programmes. De plus l’évolution de la vitesse des microprocesseurs comparativement à celle de la mémoire et des bus systèmes fait que la pression sur la hiérarchie mémoire reste critique.

4. Estimation de l’accélération

Nous cherchons à étendre ici un résultat présenté dans [3], plus précisément nous montrons une limitation qui ne s’est pas présentée dans cette précédente étude. Cette étude porte sur l’évaluation de 2 paradigmes de programmation pour l’exploitation de grappes de biprocesseurs PC. Il est montré dans une analyse fine de la répartition des temps d’exécutions la corrélation entre les accélérations obtenues sur les sections d’exécutions parallèles en mémoire partagée et le débit observé sur le bus système (biprocesseurs Pentium II 400Mhz).

L’hypothèse de travail prise dans [3] suppose que l’accélération obtenue est une image du débit du bus système. De plus il est considéré que les applications ont un comportement symétrique, c’est-à-dire que chaque processeur effectue la même somme de travail et génère donc le même nombre d’accès

mémoires. Il a aussi été vérifié pendant les mesures qu'il n'y a pas de transactions autre que celles générées par des lectures ou des écritures (comme par exemple des transactions lié au protocole de cohérence de cache).

Pour estimer le débit sur le bus système, l'un des compteurs d'un des 2 processeurs comptait toutes les transactions observées sur le bus divisé par le temps mis pour exécuter le calcul parallèle. Un programme synthétique équivalent à la copie dans STREAM (cf paragraphe 3) avec une boucle d'attente paramétrable permettait de commander la charge sur la hiérarchie mémoire, et ainsi d'obtenir une courbe d'accélération en fonction du débit observé.

Nous avons repris ce principe en effectuant des calculs supplémentaires (addition, multiplication, addition-multiplication). La figure 3 présente l'accélération en fonction du débit observé sur le bus système obtenu avec la machine biprocesseur Pentium II 450Mhz. La première constatation et le caractère abrupt de la courbe : un angle droit et la chute de l'accélération de 2 vers 1 avec une évolution minimale du débit observé. Ceci souligne la présence d'un phénomène de saturation indiquant que le bus système est le goulot d'étranglement et atteint sa limite aux alentours de $2,2 \times 10^7$ tr/sec. Ce résultat est très différent de celui obtenu dans [3] où la décroissance de l'accélération évolue linéairement avec le débit observé.

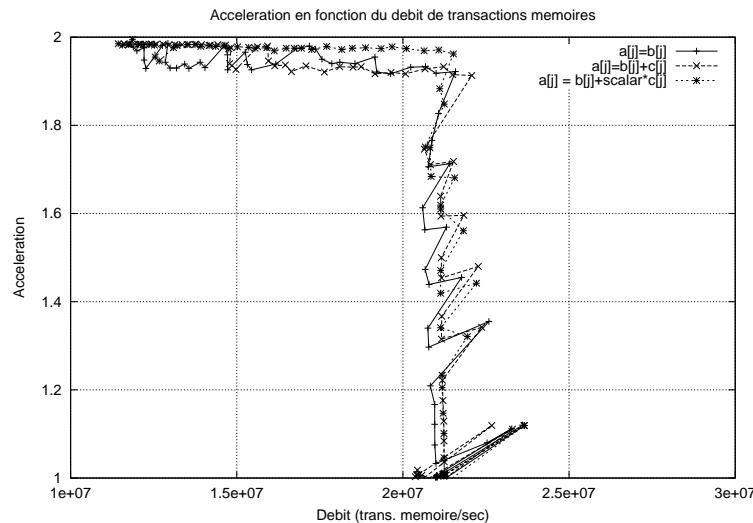


FIG. 3 – Courbe de l'accélération en fonction du débit d'événements observés (transactions mémoires/sec.) pour différents calculs.

Le comportement est globalement stable entre les différents types de calculs effectués bien que localement les résultats soient relativement discontinus. Nous pensons qu'à cette finesse d'observation cela est dû à la sensibilité des différents mécanismes architecturaux mis en jeu (par exemple : mécanisme d'exécution dans le désordre). La variance de mesure est faible et a été omise dans les figures.

La conséquence de ce phénomène est que la mesure du débit sur le bus système ne permet pas, à elle seule, l'estimation de l'accélération. Nous avons identifié un autre événement plus proche du processeur qui nous permet d'obtenir une estimation de l'accélération en fonction du débit d'événements. Il s'agit de l'événement qui a pour numéro 22H, qui correspond au nombre de cycles pendant lesquels le bus de données du cache L2 était occupé. La figure 4 présente la courbe obtenue.

Avec ce nouveau résultat, nous pouvons estimer l'accélération en fonction du débit d'événements puisque la décroissance est une fonction linéaire. Malheureusement, nous n'avons pas d'explication satisfaisante sur la différence de comportement entre les types d'événements.

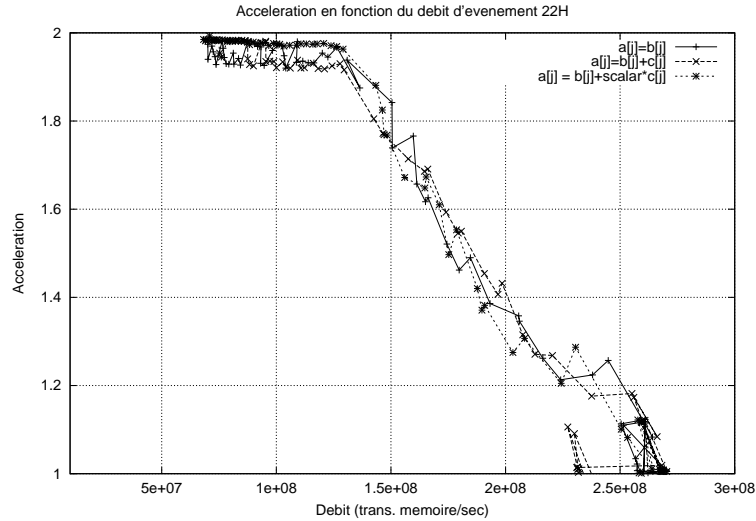


FIG. 4 – Courbe de l’accélération en fonction du débit observé (nombre de cycles où le bus de donnée du cache L2 est occupé/ sec.) pour différents calculs.

5. Un prototype de contrôle-commande d’exécution

5.1. Principe

La faible intrusion induite par l’utilisation des compteurs permet d’envisager des observations lors du déroulement des applications. Ceci rend possible l’estimation de l’accélération lors de l’exécution et de déduire l’efficacité ou le rendement courant de la machine.

En prenant pour analogie le domaine de l’automatisme, nous proposons l’élaboration d’un mécanisme de contrôle-commande (régulation) d’exécution qui du point de vue de l’administrateur permettra d’ordonner les exécutions afin d’obtenir le meilleur rendement possible. En effet en contrôlant le rendement, on peut décider qu’une application parallèle n’atteint pas une accélération suffisante (débit d’événements trop élevé), et donc décider de lui retirer un processeur au profit d’une application moins exigeante en terme d’accès à la mémoire. Lorsque la première application diminue sa pression sur la hiérarchie mémoire la ressource processeur peut de nouveau lui être attribuée au détriment de la seconde application. Le mécanisme se déclenchant en fonction du débit, on obtient une boucle de rétroaction qui essaie de maximiser le rendement de la machine.

Une condition importante pour le fonctionnement d’un tel mécanisme est la disposition d’un ensemble d’applications prêtes à être exécutées avec la contrainte forte d’être très peu exigeante pour la hiérarchie mémoire. Nous pensons que dans un cadre de production il est possible d’identifier de telles applications.

La figure 5 présente le diagramme de commande et le déroulement d’un scénario simplifié d’une exécution avec et sans mécanisme. Grâce à sa forme en hystérésis le diagramme de commande que nous avons retenu permet une certaine résistance aux perturbations. En effet, dans ce type de schéma les seuils de déclenchement d’un état de commande à un autre sont largement différent suivant le sens de transition.

5.2. Détails d’implantation du prototype

Pour ce premier prototype, nous avons choisi une approche simple qui consiste à instrumenter les applications parallélisées en mémoire partagée avec OpenMP. L’environnement OpenMP utilisé est celui fourni avec le compilateur *pgi v3.2-4*. Notre instrumentation prend place au niveau des appels aux fonctions OpenMP liées à la parallélisation de boucles.

Elle se présente sous la forme d’une bibliothèque et nécessite donc une édition de liens spécifique. L’observation et la prise de décision s’effectue à chaque appel de la boucle la plus externe du nid de boucles parallélisées. Les prises de décision sont effectuées sur une moyenne de 5 observations. Le contrôle de la seconde application en attente s’effectue par l’intermédiaire d’envois de signaux. Le retrait de la res-

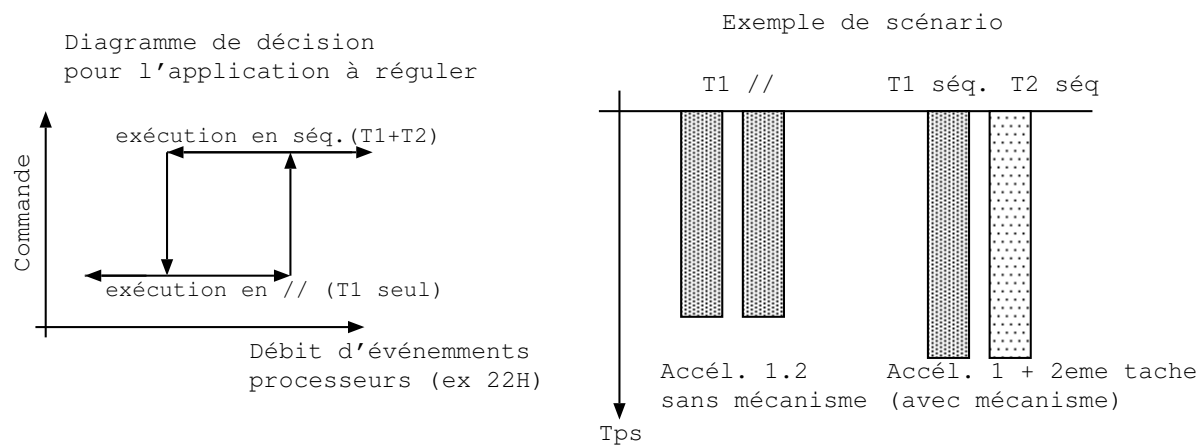


FIG. 5 – Diagramme de commande en hystérésis pour la prise de décision entre l’exécution parallèle de T1 plus l’arrêt de la seconde application séq. T2 et l’exécution séquentielle de T1 plus l’exécution de la seconde application séq. T2. A droite, exemple de scénario de contrôle-commande d’exécution pour une machine multiprocesseur (T1 application parallèle régulée et T2 application séq. supplémentaire)

source processeur pour l’application régulée consiste à effectuer les prochaines itérations en séquentiel.

5.3. Premiers résultats

Nous avons utilisé pour nos premiers tests des programmes qui génèrent des charges artificielles. L’application parallèle à réguler comporte deux sections parallèles qui sont alternativement exécutées 5 fois chacune. La première section effectue une boucle de calcul de type multiplication-addition identique à celle utilisée au paragraphe 3, et possède donc une mauvaise accélération. La deuxième section parallèle déroule plusieurs fois une boucle sur un tableau d’un million d’éléments en effectuant un calcul coûteux. Cette section engendre un très faible besoin en débit mémoire. La deuxième application en attente d’exécution, exécute en séquentielle un calcul coûteux sur un petit tableau et génère une très faible charge sur la hiérarchie mémoire.

Le tableau 3 présente nos premiers résultats. La première colonne indique les temps mis pour l’exécution de l’application à réguler. On notera que le temps en exécution parallèle et le temps avec le mécanisme de régulation est presque identique. La deuxième colonne, à considérer seulement avec le mécanisme de régulation, indique le temps qui a été récupéré pour l’application séquentielle. Ceci montre que notre mécanisme est capable de récupérer la ressource processeur sur les phases où l’accélération est très dégradée.

	Application parallèle	Application séquentielle en attente d’exécution
Temps exéc. parallèle	16,7 sec.	-
Temps exéc. séquentielle	23 sec.	-
Temps avec la régulation	17 sec.	5 sec.

TAB. 3 – Résultats du prototype montrant l’intérêt du mécanisme de régulation d’exécution.

5.4. Limitations et discussions

Ce premier prototype avait pour but de tester la faisabilité du mécanisme. Des développements sont encore nécessaires pour compléter nos tests. En effet pour valider pleinement notre mécanisme nous devons étendre nos tests avec des programmes plus réalistes, comme les programmes NAS. Une phase de calibration est nécessaire pour calculer les seuils utilisés par le processus de prises de décision, elle est réalisée manuellement et devrait être automatisée.

Un certain nombre de questions se posent par rapport à ce mécanisme, comme le fait que nous avons seulement considéré des applications parallèles avec un comportement symétrique. De plus, ces applications ne doivent pas non plus engendrer de transactions liées au protocole de cohérence de cache. Des tests seront aussi nécessaires pour évaluer la robustesse de notre mécanisme.

Dans une vision un peu restrictive nous avons seulement considéré le cas d'applications parallèles à réguler, on pourrait aussi s'intéresser aux cas de la régulation d'ensembles de processus indépendants avec des besoins en débits mémoires variés. Et enfin il serait intéressant de considérer le cas de multiprocesseurs avec un plus grand nombre de processeurs.

6. Conclusion

Dans cet article, nous avons rappelé l'importance de soulager la pression sur la hiérarchie mémoire pour les multiprocesseurs à faible coût avec quelques programmes tests classiques. Puis nous avons étendu le résultat d'une étude portant sur la corrélation entre le débit des transactions observées sur le bus système et l'accélération obtenue pour une application parallèle en mémoire partagée. Nous avons montré que lorsque le bus système était saturé, il était nécessaire d'observer le débit d'événements lié à l'occupation du cache de second niveau pour établir une nouvelle corrélation. Des travaux futurs devraient nous permettre de mieux comprendre la pertinence de ce choix. Nous avons ensuite proposé une utilisation originale de cette corrélation, qui consiste, par un principe de boucle à rétroaction, à contrôler le rendement d'exécution sur une machine multiprocesseur. Les premiers résultats obtenus avec notre prototype sont encourageants.

Bibliographie

1. L. Berc, S. Ghemawat, M. Henzinger, S. Leung, M. Lichtenberg, D. Sites, M. Vandevoorde, C. Waldspurger, and B. Wehl. Digital continuous profiling infrastructure, 1996.
2. R. Berrendorf and H. Ziegler. Pcl – the performance counter library : A common interface to access hardware performance counters on microprocessors, 1998.
3. Franck Cappello, Olivier Richard, and Daniel Etiemble. Investigating the performance of two programming models for clusters of SMP PCs. In *HPCA*, pages 349–359, 2000.
4. Intel. Corporation. Vtune performance analyzer. <http://developer.intel.com/vtune/>.
5. Leonardo Dagum and Ramesh Menon. OpenMP : An industry-standard API for shared-memory programming. *IEEE Computational Science & Engineering*, 5(1) :46–55, January/March 1998.
6. L. DeRose. The hardware performance monitor toolkit. In *Proceedings of EuroPar*, August 2001.
7. L. DeRose and A. Reed. Svpablo : A multi-language architecture independent performance analysis system, 1999.
8. Shirley Moore Philip Mucci Jack Dongarra, Kevin London and Daniel Terpstra. Using papi for hardware performance monitoring on linux systems. *Linux Clusters : The HPC Revolution*, July 2001.
9. H. Jin, M. Frumkin, and J. Yan. The openmp implementation of nas parallel benchmarks. Technical Report NAS-99-011, NASA Ames Research Center, 1999.
10. Kazuhiro Kusano, Shigehisa Satoh, and Mitsuhiro Sato. Performance evaluation of the omni openmp compiler. In *ISHPC*, pages 403–414, 2000.
11. Gotz Lindenmaier, Kathryn S. McKinley, and Olivier Temam. Load scheduling with profile information. In *European Conference on Parallel Processing*, pages 223–233, 2000.
12. Dongarra J. Moore S. Mucci P. Seymour K. and Spencer T. London, K. End-user tools for application performance analysis, using hardware counters. *International Conference on Parallel and Distributed Computing Systems*, August 2001.
13. J. McCalpin.
14. W. Wehl. Continuous profiling : Where have all the cycles gone. In *In 16th ACM Symposium of Operating System Design*, pages 357–390, 1997.
15. Marco Zgha, Brond Larson, Steve Turner, and Marty Itzkowitz. Performance analysis using the MIPS R10000 performance counters. In *Supercomputing*, 1996.