

Les processeurs Itanium

**Programmation
et optimisation**

S m a i l N i a r

J a m e l T a y e b

© Groupe Eyrolles, 2005,

ISBN : 2-212-11536-9

EYROLLES



Introduction

Pourquoi l'architecture EPIC ?

Tout le monde connaît la loi de Gordon Moore énoncée en 1965 : *le nombre de transistors par circuit de même taille doublera tous les 18 à 24 mois*. Ceci s'entend pour le processeur précurseur (*lead microprocessor*) qui intègre pour la première fois une nouvelle micro-architecture. Nous ne prenons donc pas en considération ici les améliorations induites par les nouvelles techniques de lithographie permettant l'obtention de processeurs plus compacts (*compaction microprocessor*). En effet, une finesse de gravure accrue permet d'augmenter la densité des transistors pour un microprocesseur donné mais n'augmente pas pour autant leur nombre. Ainsi, la surface de la puce des processeurs précurseurs augmente d'environ 14 % tous les deux ans. En dix ans, la technologie de gravure est passée quant à elle de 1 μm à 0,18 μm (Intel produit aujourd'hui certaines de ses puces en 90 nm et prévoit des circuits en 32 nm d'ici 2009 et 22 nm en 2011). La fréquence d'horloge a été multipliée pour sa part par cinquante. La performance des processeurs a ainsi subi un accroissement d'environ $\times 75$, dont malheureusement seulement environ $\times 6$ sont dus à l'exploitation par les micro-architectures du budget de transistors sans cesse plus élevé. La question que nous pouvons dès lors nous poser légitimement est la suivante : quelle est la signification de la loi de Moore en termes de performance ? En effet, en dix ans, les avancées micro-architecturales telles que les pipelines, le super-scalaire, les cœurs à exécution désordonnée – *Out Of Order Execution Engine* (OOO) – ou encore l'exécution spéculative des instructions ont multiplié les performances d'un « petit » facteur six seulement !

Pour définir l'augmentation de performance, nous pouvons recourir à la formulation de Fred Pollack (*Intel Fellow*, directeur du *Microprocessor Research Labs*). Ce dernier a observé que pour le même procédé de fabrication, le processeur précurseur offre en moyenne un gain de performance d'environ $\times 1,4$ pour un doublement de la surface par rapport au processeur de génération précédente. Pollack constate ainsi que le gain de performance décline avec les générations successives et évolue comme la racine carrée de l'accroissement de la surface. En d'autres termes, bien que les physiciens permettent d'augmenter le nombre de transistors selon la prévision de Moore, il faut impérativement introduire des modifications radicales dans les micro-architectures pour augmenter les gains de performance. Autrement dit encore, l'industrie du microprocesseur est dans une impasse et ne peut plus se reposer sur les recettes du passé, à savoir augmenter les

fréquences d'horloge indéfiniment. En outre, cette course dont le rythme augmente sans cesse a un coût en termes de budget thermique, et, si rien n'est fait, celui-ci deviendra très rapidement insupportable pour les industriels.

La solution retenue par Intel pour résoudre le problème crucial de la performance peut se résumer en un seul mot : parallélisme ! Parallélisme de données tout d'abord avec le calcul SIMD (Single Instruction Multiple Data) introduit par les technologies MMX, SSE, SSE2 puis SSE3, mais aussi parallélisme au sens littéral du terme avec des technologies telles que l'*Hyper-Threading* ou SMT (Simultaneous Multi Threading), la virtualisation, le *multi-core*, ou plus fondamentalement le MIMD (Multiple Instruction Multiple Data) d'EPIC. Développée conjointement par Hewlett-Packard et Intel, l'architecture EPIC que nous détaillons au cours du chapitre suivant a donc pour mission délicate d'offrir une nouvelle approche architecturale qui doit être à même de relever le défi de la performance.

L'architecture EPIC se retrouve aujourd'hui au cœur des processeurs de la famille Itanium d'Intel. Elle offre de nombreux aménagements matériels inédits qui, utilisés conjointement avec des logiciels correctement conçus, permettent de passer outre les barrières de performances jusqu'alors infranchissables par les architectures actuelles (RISC, Reduced Instruction Set Computer et CISC, Complex Instruction Set Computer confondues). Comme nous le verrons au fil des chapitres de cet ouvrage, la composante logicielle de l'architecture EPIC est essentielle. Elle l'est à tel point que le compilateur fait intégralement partie de l'architecture. Ce n'est plus un simple outil voué à simplifier l'existence des programmeurs d'applications : il forme un tout avec EPIC. Avec son support, EPIC permet essentiellement d'augmenter le degré de parallélisme au niveau des instructions (ILP ou Instruction Level Parallelism), jusque-là apanage exclusif des cœurs OOO. En outre, il permet de masquer les latences d'accès aux données et/ou d'exécution des instructions, problème qui tourmente les architectes depuis l'avènement du MANIAC (Mathematical Analyser, Numerator, Integrator And Computer) de John Louis Von Neumann. Une fois de plus, le travail conjoint du matériel et du logiciel permet la levée de ce goulet d'étranglement fondamental que sont les accès en mémoire. Les technologies mises en œuvre s'appellent, comme nous le verrons prochainement, *software pipeline*, et spéculation de données et de contrôle.

La roadmap Itanium d'Intel

Si pour l'heure, seuls les processeurs de la famille Itanium d'Intel implémentent l'architecture EPIC, rien n'empêche en théorie de la retrouver un jour au sein de puces fondées par d'autres entreprises, ou proposant des implémentations variables d'EPIC. En effet, EPIC est une spécification qui laisse suffisamment d'espace libre pour autoriser des amendements, des implémentations variables, voire des interprétations, pour répondre à des besoins spécifiques, à l'instar de l'architecture x86. Seuls des considérations financières – certes essentielles pour l'industrie de l'électronique – peuvent limiter l'adoption d'EPIC par d'autres fondeurs (licences, coûts de développement, etc.). En effet, si Hewlett-Packard a mis fin au futur des processeurs Alpha et PA-Risc, c'est que l'entreprise

ne souhaitait plus investir des sommes littéralement colossales dans le développement de ses architectures propriétaires, et préférait confier cette tâche au spécialiste du secteur qu'est Intel.

L'architecture EPIC n'est pas figée pour autant puisque Intel lui-même propose régulièrement des évolutions de la famille Itanium. Le fondateur de Santa Clara prévoit même un rythme élevé de mise à jour avec la mise en vente d'une version majeure de l'Itanium tous les ans. Six nouvelles implémentations sont d'ailleurs en cours de développement dans les laboratoires d'Intel en Oregon (USA) et déjà sept processeurs Itanium sont utilisés par soixante-dix OEM (Original Equipment Manufacturer). Les tableaux I-1 et I-2 récapitulent la chronologie d'introduction des processeurs Itanium et, à titre indicatif, la liste des modèles et des prix disponibles à ce jour.

Tableau I-1 Les processeurs de la famille Itanium

Processeurs	Noms de code	Introduction	Spécificités
Itanium	Merced	1999	Processeur de développement
Itanium 2, LV Itanium 2	McKinley, Deerfield	2002	Implémentation complète d'EPIC, version basse tension
Itanium 2 6 Mo	Madison 6 Mo, Fanwood	2003	6 Mo de cache L3, version basse tension
Itanium 2 9 Mo	Madison 9 Mo, Fanwood	2004	9 Mo de cache L3, version basse tension
Inconnu	Montecito, Millington	Prévu pour 2005	Double cœur avec Hyper-Threading, version basse tension
Inconnu	Tukwila	Prévu pour 2006	Plus de deux cœurs, topologie indéfinie

Tableau I-2 Extrait de la liste des prix officiels d'Intel, valable au 17 octobre 2004

Processeurs Itanium 2 Fréquence	Taille de la mémoire cache L3	Prix par lot de 1000 processeurs*
1,50 GHz	6 Mo	\$4 227
1,40 GHz	4 Mo	\$1 980
1,30 GHz	3 Mo	\$910
1,60 GHz	3 Mo	\$2 408
1,40 GHz	3 Mo	\$851
1,40 GHz	1,5 Mo	\$851
1,0 GHz	1,5 Mo	\$530

Remarque

La liste des prix officiels d'Intel est mise en ligne à l'adresse <http://www.intel.com/intel/finance/pricelist/index.htm>. Tous ces processeurs sont gravés en 130 nm et ont une fréquence de bus frontal de 400 MHz.

Un processeur seul, aussi novateur ou performant soit-il, ne sert malheureusement à rien. Il doit impérativement disposer d'un écosystème digne de ce nom, surtout s'il a pour vocation commerciale – comme l'Itanium – de modifier la donne dans l'univers des processeurs RISC en transformant un modèle économique vertical en un modèle dit horizontal.

Modèle vertical/modèle horizontal

Un modèle informatique est dit vertical, lorsque la même entreprise fournit le processeur, la plate-forme, les périphériques, le système d'exploitation et quelquefois les applicatifs et les services. Par opposition, un système dit horizontal, permet à plusieurs entreprises de fournir les processeurs, les plates-formes, les systèmes d'exploitation, les logiciels et les services. Au final, la concurrence qui s'installe entre les différents acteurs mène inexorablement à une forte standardisation et à une baisse des coûts à l'image de l'industrie du PC.

À ce jour, trois grandes familles de systèmes d'exploitation (HP-UX, Linux et Windows et bientôt OpenVMS) et plus de 70 OEM proposent des plates-formes à base d'Itanium (du biprocesseur jusqu'aux systèmes SMP à 128 CPU). Le tableau I-3 reprend le positionnement des différentes versions du processeur Itanium 2 au sein de plates-formes typiques. Au total, Intel aura vendu près de 100 000 processeurs Itanium en 2003. Enfin, plus de 2 700 applications ont été portées et optimisées pour l'architecture EPIC.

Tableau I-3 Positionnement des différentes moutures du processeur Itanium 2

Performances maximales			
Itanium 2 (Madison 6 Mo, 1,5 GHz)	Itanium 2 (Madison 9 Mo, 1,6 GHz)	Montecito (2 cœurs, 24 Mo)	Tukwila (2+ cœurs)
Meilleur rapport €/FLOP			
Itanium 2 (3 Mo, 1,4 GHz, DP)	Itanium 2 (Fanwood, 3 Mo, 1,6 GHz)	Millington (DP fondé sur Montecito)	Dimona (DP fondé sur Tukwila)
Basse consommation			
LV Itanium 2 (1,5 Mo, 1,0 GHz, DP)	LV Itanium 2 (LV Fanwood, 3 Mo, 1,2 GHz, DP)	LV Millington (LV fondé sur Montecito, DP)	LV Dimona (LV fondé sur Tukwila, DP)
1999 – 2003	2004	Prévu en 2005	Prévu en 2006

Cette roadmap ou feuille de route, fait apparaître un détail qui pourrait passer inaperçu. En effet, le processeur Itanium 2 et ses successeurs s'y voient systématiquement déclinés en une version basse tension (LV, Low Voltage). Il faut savoir que la mouture 9M MP (Multi Processor) de l'Itanium 2 ne consomme pas moins de 130 watts (budget maximal). La version 3M DP (Dual Processor) pourra consommer pour sa part jusqu'à 99 watts, là où les versions LV consomment au maximum 62 watts (voir tableau I-4). Ces processeurs sont avant tout destinés à équiper les systèmes à haute densité, généralement conçus sous la forme de serveurs lame. Certains équipements réseau – pour le chiffrement SSL notamment – peuvent aussi tirer avantageusement parti de ces déclinaisons.

Enfin, certaines fermes de calcul (*clusters*) trouveront dans ces puces un bon rapport performance/prix, et surtout performance/consommation électrique.

Tableau I-4 Enveloppe de consommation des processeurs Itanium 2 en fonction de la fréquence

Fréquence	Thermal Design Power
900 MHz	90 W
1,0 GHz	100 W
1,30 GHz	97 W
1,40 GHz	91 W
1,50 GHz	107 W
1,60 GHz	122 W

Ces processeurs sont les annonciateurs d'un phénomène qui est en train de gagner le monde des serveurs. En effet, jusque-là réservée aux systèmes portables, la gestion fine de la consommation électrique devient une priorité et un critère de choix pour les centres de calcul. Le coût de l'électricité consommée par les processeurs – mais aussi par le système de climatisation – n'est pas négligeable. En outre, il faut savoir qu'à l'heure actuelle, près du 1/8^e de la puissance consommée par un processeur l'est en pure perte à cause des courants de fuite au niveau des transistors. Si rien n'est fait, les projections pronostiquent que la moitié de la puissance sera ainsi perdue avec un *process* de fabrication en 65 nm et l'enveloppe de puissance atteindrait 1000 watts ! Dès lors, en plus de ces versions LV, les processeurs Itanium (mais également les autres puces serveur) se verront affublés de circuits offrant des fonctionnalités sinon équivalentes, du moins proches de celles des plates-formes mobiles. Ainsi, le positionnement du voltage et de la fréquence sera utilisé pour adapter la puissance consommée à la charge de travail du CPU. Il sera possible, au niveau d'un *data center*, de fixer un seuil maximal de consommation électrique et de le faire varier dans le temps si le besoin venait à s'en faire sentir. Ainsi, dans une salle de marché utilisant les prochaines générations de processeurs Itanium par exemple, la consommation électrique maximale serait autorisée pendant les ouvertures des grandes places boursières, puis réduite considérablement pour le restant de la journée.

Les applications devront se préparer à répondre aux événements système qui leur indiqueront l'état électrique de la plate-forme. Cela se fera à l'image des applications amicales vis-à-vis des batteries (*battery friendly*) qui sont activement développées, notamment avec la montée en puissance du marché des ordinateurs portables. En fonction de cet état électrique, les performances des applications changeront (car la fréquence évoluera également). Nous pouvons alors imaginer que des technologies d'optimisation de consommation électrique voient le jour prochainement et soient intégrées dans les compilateurs. Parmi les pistes sur lesquelles travaillent les chercheurs, l'une d'elles serait d'autoriser le compilateur à créer différentes versions de certaines procédures. Et, en fonction de l'état électrique du serveur, le processeur irait exécuter préférentiellement un

chemin de code qui serait plus ou moins consommateur d'énergie. Ces technologies de régulation et de positionnement apparaîtront courant 2005 avec l'introduction du processeur Montecito Mais les implications du côté logiciel pourraient être bien plus nombreuses que nous ne pourrions l'imaginer. En effet, rien n'exclut que les multiples cœurs d'un même processeur physique n'aient pas la même fréquence, donc la même consommation électrique, à un instant donné.

Remarque

Les processeurs logiques peuvent, s'ils sont bien employés, augmenter considérablement la consommation électrique d'un processeur. Il est alors vrai que la durée de traitement en sera également réduite et que le système dans son ensemble pourra basculer en veille plus rapidement. Il s'agit comme toujours de trouver le juste équilibre.

Pourquoi faut-il optimiser ses applications pour EPIC ?

Comme vous l'apprendrez dans cet ouvrage, pour utiliser pleinement les capacités de l'architecture EPIC, les logiciels doivent impérativement coopérer au sens littéral avec le processeur. Si cette interaction est souhaitable pour n'importe quelle architecture afin d'obtenir le meilleur niveau de performance possible, cela est encore plus vrai pour l'EPIC. En effet, prendre le code source d'un logiciel, le porter pour gérer un adressage sur 64 bits si nécessaire (problématique abordée au chapitre 6), et le recompiler n'offre pas toujours le meilleur résultat possible. Cela est d'autant plus avéré que le compilateur utilisé n'intègre qu'un nombre limité de technologies d'optimisation. En effet, il faut toujours conserver à l'esprit que le compilateur est un composant fondamental de l'architecture EPIC.

Comme tout outil, aussi évolué soit-il, le compilateur n'est hélas pas capable de réaliser des miracles sur tous les codes, et plus particulièrement sur les plus mal conçus ! Puisque le processeur ne prend pas en charge la recherche dynamique du meilleur chemin d'exécution dans le code, les erreurs de conception algorithmique et d'implémentation apparaissent de façon patente. Ces erreurs sont exacerbées par une absence totale d'initiative du processeur. Dans EPIC, il y a *Explicit*, par conséquent le processeur exécutera très précisément ce que le programmeur aura écrit et il le fera dans l'ordre dans lequel il l'a écrit !

Dès lors, il apparaît clairement qu'il est indispensable d'optimiser ses applications pour l'architecture EPIC. S'il était auparavant souhaitable de connaître les principes fondamentaux de l'architecture EPIC (voir chapitre 2), cette étape n'est plus une nécessité absolue aujourd'hui. En effet, la première phase d'optimisation pour les processeurs Itanium peut être aussi simple que d'utiliser un bon compilateur et de l'aider au mieux de vos connaissances de l'application, via des options de compilation judicieusement choisies et des directives (*pragmas*) adaptées (voir chapitres 3 et 4). Enfin, les bibliothèques spécifiques et optimisées peuvent être d'un grand secours. Cette approche permet généralement d'obtenir des codes très performants sur les plates-formes à base de processeurs

Itanium sans modifier fondamentalement le code source de l'application. C'est une requête sans cesse formulée par les éditeurs de logiciels.

Cependant, malgré tous les efforts des compilateurs et des programmeurs, certains codes peuvent mettre en défaut les technologies d'optimisation logicielle. Ce sera alors au seul programmeur – ou au spécialiste des questions de performances – de prendre le relais. Mais encore faut-il être en mesure de détecter ces cas atypiques. Au chapitre 4, nous vous présentons à cet effet une méthodologie qui vous permet de contrôler la qualité du code généré par le compilateur, et de le rectifier par des retouches souvent mineures.

Et si cela ne suffisait toujours pas ? Dans ce cas, il faudra recourir à des outils d'analyse de performances dédiés (à l'image de VTune d'Intel). Ces outils ouvrent une fenêtre sur ce qui se passe au cœur du processeur pendant l'exécution de vos logiciels. En ce sens, ils sont extrêmement utiles, et donnent une vision très précise à l'échelle architecturale des événements impliqués dans votre problème de performance. Il faut savoir que les processeurs Itanium exposent aujourd'hui plus de 500 compteurs micro-architecturaux qui peuvent se révéler être autant d'indices d'une sous-utilisation des ressources du CPU. Et ce chiffre ne manquera pas de croître avec l'apparition des nouvelles implémentations d'EPIC.

Conscients du fait que les programmeurs ne sont pas des spécialistes de l'architecture des processeurs, et qu'ils ne souhaitent généralement pas le devenir, cette réponse serait une non-réponse. En outre, l'adage qui veut que « trop d'information tue l'information » se vérifie ici à plus d'un titre. Avec un choix de plus de 500 compteurs architecturaux à sa disposition, il est impossible, même pour un spécialiste ou un programmeur chevronné, d'effectuer un travail productif avec ces informations et d'améliorer ainsi les performances de son code dans un temps raisonnablement court. C'est pour cela que nous vous proposons au chapitre 5 une méthodologie quasi-automatique, « l'analyse micro-architecturale », qui permet d'identifier avec précision où vous perdez de la performance dans votre code. Elle vous explique également exactement pourquoi vous n'arrivez pas à exploiter l'architecture EPIC à son maximum ! Grâce à cette méthode, l'analyse ne requerra en général l'utilisation que d'une vingtaine de compteurs architecturaux. Une fois le problème cerné avec précision, le programmeur pourra agir chirurgicalement sur son code en employant nos conseils.

Notre stratégie d'optimisation

Il ne s'agit pas bien entendu de foncer tête baissée et d'appliquer nos recommandations aléatoirement. Nous vous proposons une stratégie d'optimisation depuis longtemps éprouvée chez Intel et ses partenaires technologiques. Elle consiste à réduire les interventions sur le code tout en utilisant le plus possible d'outils spécialisés, tels que les compilateurs optimiseurs, les bibliothèques optimisées ainsi que les outils d'analyse de performance. La stratégie exige également de dresser un profil de l'application et d'identifier la ou les routines qui consomment le plus de temps CPU à l'exécution. Suivant la nature des codes et des algorithmes, 10 % à 20 % du code seront ainsi éligibles pour un travail

d'optimisation (également appelé le « code chaud »). Les 80 % restants ne seront pas touchés. En revanche, pour les codes chauds de l'application (là où vous passez le plus clair du temps d'exécution), nous appliquerons notre processus itératif d'optimisation et utiliserons la panoplie de méthodes, d'outils et de solutions que nous présenterons tout au long de ce livre.

Nous commencerons par fixer un objectif à l'optimisation. Il s'agira généralement d'atteindre un niveau de performance qui sera mesuré par un ou plusieurs *benchmarks* ou bancs de tests et un ou plusieurs jeux de données représentatifs. Cette discussion – essentielle à la méthodologie – sera détaillée au chapitre 6, où nous présenterons des études de cas d'applications réelles. Une fois le banc de tests défini (il devra être reproductible, représentatif et honnête), il s'agira d'appliquer le processus d'optimisation itératif résumé ci-après. Certaines de ces étapes peuvent s'avérer optionnelles et vous remarquerez que la méthodologie relève du bon sens. En revanche, ce qui sera spécifique à l'architecture EPIC et aux processeurs Itanium, ce seront les méthodologies de second niveau, les outils et les techniques de mesure, ainsi que les optimisations mises en œuvre. Voici un résumé de notre méthodologie :

1. Fixer les objectifs de l'optimisation.
2. Mettre au point un banc de tests irréprochable et mesurer les performances de référence.
3. Identifier les 10 % à 20 % du code qui doivent être optimisés, rechercher la cause du problème et appliquer au choix les :
 - les optimisations de haut niveau ;
 - les optimisations via le compilateur et les bibliothèques ;
 - les optimisations via l'analyse micro-architecturale.
4. Mesurer les gains de performance (si les objectifs ne sont pas atteints, alors reprendre le processus itératif à l'étape 2). Si les objectifs sont atteints, alors mettre fin à l'optimisation.

Le futur d'Itanium

Avant de clore notre introduction, nous allons vous donner un aperçu des deux évolutions majeures que la famille des processeurs Itanium va subir prochainement. Nous savons à présent qu'Intel compte améliorer les performances de tous ses processeurs, non plus seulement en augmentant la fréquence d'horloge, mais aussi et surtout en élevant le degré de parallélisme offert par ses architectures. La première transformation des processeurs Itanium sera donc fort logiquement l'intégration en leur sein de la technologie SMT (Simultaneous Multi Threading) connue aussi sous le sigle HT (Hyper-Threading). La seconde révolution sera l'adoption de la technologie du *multi-core* ou cœur multiple.

Enfin, nous allons rapidement décrire ces amendements majeurs du design des processeurs Itanium, et préciser ce que cela signifie pour l'architecture EPIC. Dans les deux cas

de figure, il s'agit d'un message fort que le fondateur mais aussi ses concurrents qui annoncent des technologies similaires, adressent aux éditeurs de logiciels. Il leur sera en effet impératif de concevoir des applications parallèles qui pourront tirer profit de ces nouvelles architectures. Cet effort sera conséquent et bien plus long et coûteux que d'optimiser son code de façon spécifique pour une architecture. Néanmoins, il leur sera impossible de faire l'impasse, au risque d'augmenter considérablement le ratio prix/performance des plates-formes qui sont le vecteur principal du succès de leurs logiciels.

La technologie HT

La technologie HT, initialement proposée sur les processeurs Xeon et Pentium 4, offre le support matériel nécessaire à l'exécution concurrente de plusieurs threads – deux en l'occurrence pour ces premières moutures.

Remarque

L'*Hyper-Threading* est la première mise en œuvre industrielle de la technologie SMT décrite la première fois en 1995 (université de Washington).

Nous parlons bien d'exécution simultanée des threads et non pas du mécanisme de temps partagé du système d'exploitation. Celui-ci, pour peu qu'il prenne la technologie en compte, voit deux processeurs là où il n'y en a qu'un en réalité. Il s'agit de deux processeurs logiques par opposition à un seul processeur physique. Une plate-forme équipée de deux processeurs HT sera donc vue comme un système quadri-processeurs par le système d'exploitation et les applications.

Évolutions futures

Windows XP et Server gèrent parfaitement HT. Les noyaux 2.6 x ont connaissance de HT mais pourraient mieux faire. Le cas de Linux devrait prochainement évoluer, mais des oppositions féroces de Linus Thorvald ralentissent les amendements pourtant nécessaires au succès de ce remarquable système d'exploitation.

Pour comprendre la philosophie qui sous-tend l'HT, il faut savoir qu'un processeur moderne (superscalaire et OOO) est sous-exploité par la grande majorité des applications. Plus exactement, ses unités d'exécution sont oisives pendant environ 60 % du temps d'exécution. Ceci n'est pas scandaleux pour autant puisque cette limitation est imposée par le manque de parallélisme inhérent à tous les programmes informatiques (*Instruction Level Parallelism* inextensible). C'est en partant de ce constat que les ingénieurs d'Intel ont décidé d'implémenter la technologie du SMT au détriment d'un accroissement de la surface de la puce d'environ 5 %. En plaçant ainsi en concurrence deux threads, les ressources du processeur sont mieux exploitées (jusqu'à un gain de 30 %). Toutefois, dans le cadre de l'architecture EPIC, cet aspect n'est pas aussi important que pour un processeur OOO ayant un pipeline extrêmement long (comme le cœur Prescott des processeurs Pentium 4 et Xeon et ses 31 niveaux de pipeline). En revanche,

les états offerts par l'HT sont très bénéfiques pour les processeurs Itanium. « État » s'entend ici comme un état d'un automate. En effet, les processeurs HT définissent trois états fondamentaux : ST0, ST1 et HT. Ils correspondent respectivement aux trois cas suivants :

- Un seul thread s'exécute sur le processeur logique 0 du processeur physique.
- Un seul thread s'exécute sur le processeur logique 1 du processeur physique.
- Deux threads s'exécutent sur les deux processeurs logiques du processeur physique.

Par exemple, la configuration ST1 s'observe quand un processeur HT bascule de l'état HT à l'état ST1 après que le thread exécuté par le processeur logique 0 du processeur physique a été préempté par le système d'exploitation (suite à une exception ou l'attente d'une ressource par exemple). Dans ce cas, les ressources scindées sont entièrement réallouées au thread actif.

Remarques

Avec un surcoût de seulement 5 % en surface d'un processeur HT versus un modèle standard, il est évident que les aménagements sont mineurs. Si les registres et certains composants comme l'ACPI sont dupliqués, les autres ressources (caches, tampons, queue, etc.) sont partagées ou scindées.

La réallocation des ressources internes du processeur est gérée différemment par les ordonnanceurs (*schedulers*) des systèmes d'exploitation. Sachez simplement qu'elle ne s'opère pas immédiatement. En pratique, le basculement suit l'exécution d'une instruction `halt`. Pour mémoire, l'instruction `halt` de privilège 0 interrompt l'exécution des instructions et place le processeur en état interne HALT. Pour le sortir de sa léthargie, une interruption autorisée (NMI) ou une commande `reset` sont nécessaires pour le forcer à reprendre l'exécution du code.

Du point de vue du programmeur, quelques règles seront à respecter pour bien tirer profit de l'HT.

Optimisation algorithmique

Il nous est malheureusement impossible de donner le détail des optimisations algorithmiques qu'il est important d'appliquer pour bien exploiter HT. En effet, un livre entier n'y suffirait pas. Nous nous efforçons donc de donner ici les lignes directrices. En outre, le processeur Montecito n'étant encore qu'au stade alpha, de nombreux points de détails peuvent changer avant l'implémentation finale.

La première règle impose que votre code soit « multi-threadé ». Rappelons que toute application exécutée sous un système d'exploitation contemporain dispose d'au moins un thread : le thread primaire. Dans la mesure où il est créé par le système à l'exécution de votre programme, vous n'avez jamais eu à vous en soucier. Mais cela ne suffit bien évidemment pas – sauf si l'on considère que le thread primaire d'une autre application (une tâche de fond à faible priorité, par exemple) est dirigé par le système d'exploitation sur l'un des processeurs logiques oisifs du processeur physique. Votre application devra donc créer au moins un thread supplémentaire (via les API en vigueur : Win32, pThread,

OpenMP, etc.). Vous devrez ensuite éviter les *Spin-wait loops* ou boucle d'attente entre vos threads et recourir aux seuls services de synchronisation du système d'exploitation.

Une bonne gestion du temps d'attente

Lorsque vous savez que le temps d'attente d'un thread sur une ressource (un *lock* par exemple) sera inférieur au temps de gestion de la synchronisation de thread offert par le système d'exploitation, il est tentant d'effectuer une attente active au niveau du thread et de venir boucler très rapidement jusqu'à ce que l'acquisition de la ressource via une instruction atomique réussisse. Utilisez alors l'instruction pause dans la boucle.

La seconde règle consistera à faire une bonne décomposition fonctionnelle du programme et à privilégier la répartition des charges par rapport à la persistance de cache. Commençons par le problème de la décomposition fonctionnelle en revenant à l'essence même de l'HT. En effet, l'HT permet le partage des ressources du *back-end* entre deux threads mis en compétition. Si vos deux threads stressent les mêmes ressources (unités d'exécutions, tampons, etc.), alors vous ne gagnerez pas en performance puisque les instructions des deux concurrents seront systématiquement mises en attente (STALL). En revanche, si les deux threads exploitent des ressources différentes (calculs en virgule flottante – FP ou Floating Point – pour l'un et calculs sur les entiers pour l'autre, par exemple) alors vous gagnerez en performance. Abordons ensuite le problème de la répartition des charges. Si, malgré tous vos efforts, vos threads sont contraints d'effectuer des tâches similaires, alors vous devez réaliser une décomposition par domaine : le premier thread appliquera le traitement à une première moitié des données et le second se chargera de l'autre moitié. C'est typiquement le cas des applications scientifiques parallélisées via OpenMP ou des applications graphiques (calcul de lancé de rayon, encodage vidéo, etc.).

Commençons par examiner le cas d'un système biprocesseur classique. Chaque thread s'exécute donc sur son processeur physique attribué par le système d'exploitation. Les performances sont généralement bonnes car les données sont partagées et bénéficient pleinement de la fonction des mémoires cache. Dans ce cas, l'ordonnancement des threads peut être statique : les données sont partagées – en deux blocs généralement de même taille – et confiées aux deux threads. Dans le cas général, la persistance des caches pallie d'éventuelles disproportions de charge entre les threads. Ce déséquilibre apparaît lorsque le temps nécessaire au traitement des blocs de données est inégal. Notez que de nombreux cas particuliers échappent à cette simplification.

Voyons à présent ce qui se passe avec un système équipé de processeurs HT. L'allocation des blocs de données (plus nombreux et plus petits que dans le cas statique) se fait de façon dynamique au premier thread libre. Ainsi, lorsqu'un des threads hérite de données faciles à traiter (zone de l'image à calculer qui présente peu de détails, par exemple), il se verra attribuer un bloc de données supplémentaire pendant que l'autre thread traite encore son bloc de données plus complexe. Nous réduisons ainsi légèrement la persistance des caches, mais assurons un flux plus constant des instructions (et accessoirement, nous évitons un possible changement de mode HT vers STx malgré tout assez coûteux en termes de performance). Dès lors, la durée maximale d'attente entre les deux threads est

au plus égale au temps requis par le traitement d'un bloc de données... s'ils sont de taille uniforme bien entendu !

Pour la troisième règle, rappelons que vous pouvez/devez profiter de la fonction de *prefetch* ou préchargement induite par les threads sur un système HT. Il s'agit de mettre à profit un défaut qui devient de facto une qualité. Prenons deux threads qui se partagent le même processeur physique. Ce partage s'étend bien évidemment aux caches du processeur physique (aux caches L2 et L3 de l'un des cœurs du processeur Montecito). Les threads disposent d'environ une moitié des caches pour leurs données, ce qui cause irrémédiablement une baisse de performance puisque le gestionnaire de cache doit servir les défauts de cache dont le nombre croît (il s'agit des évictions de cache). Cela est bien sûr pénalisant lorsque les threads opèrent sur des données différentes. Supposons à présent que les threads travaillent sur les mêmes données en lecture (ou du moins proche sur le plan spatial et temporel). Dans ce cas, le premier thread qui manipule une donnée absente du cache L3 engendre un défaut de cache et force le gestionnaire à charger la donnée demandée (une ligne de cache dans une voie). Le second thread entre alors en jeu et demande la même donnée (ou une donnée dans la même ligne de cache – *Shared Cache Line*). Le gestionnaire de cache identifie la donnée comme présente et le thread est immédiatement servi. Le premier thread a rempli le rôle de *prefetcher* ou préchargeur pour le second ! C'est le mécanisme de la prochaine modification architecturale à venir sur les processeurs Itanium : le VMT (Virtual Multi Threading).

En guise de quatrième et dernière règle, évitez l'utilisation des données faussement partagées par les threads et stockées dans la même ligne de cache. La principale différence entre ce cas de figure et le cas présenté dans le paragraphe précédent est qu'ici nous nous intéressons aux accès en lecture/écriture et écriture/écriture, contre lecture/lecture précédemment. Supposons que nos deux threads disposent d'un bloc de paramètres (méthode traditionnelle pour passer des données à un thread lors de sa création via un pointeur `void *`). Supposons également que la taille du bloc de paramètres soit inférieure à 128 octets. Le bloc d'un des deux threads n'occupera donc pas la totalité de sa ligne de cache L2 par exemple. Et il y a fort à parier qu'une fraction du bloc du second thread occupera le reste de la ligne ainsi qu'une portion d'une autre ligne. Le second thread va être pénalisé par des *Split Load* (deux accès à la L2 nécessaires au lieu d'un seul pour une lecture). Les blocs sont dits « faussement partagés ». En effet, dès que l'un des deux threads y accède en écriture, il faut aussi mettre à jour le bloc de l'autre thread (imposé par le mécanisme de cohérence de cache). Pour pallier ce défaut, il suffit le plus souvent d'aligner les données manipulées par les threads sur 128 octets et de combler l'espace libre par une donnée inutilisée (technique du *padding*).

Pour aller plus loin

Autant la technologie de l'*Hyper-Threading* est relativement bien connue et commence à être exploitée par les programmeurs, autant les deux technologies que nous allons décrire ici peuvent laisser dubitatif. En tout cas, puisqu'il s'agit encore de travaux de recherche au moment où nous écrivons ces lignes, elles peuvent être sujettes à modification.

Des recherches menées par Intel poussent le principe du prefetch induit par HT à son paroxysme. Commençons par étudier le cas dans un système HT. Nous verrons par la suite le prototype d'implémentation avec un processeur à architecture EPIC (qui ne doit pas être nécessairement SMT pour être fonctionnel puisqu'il s'agit d'une autre voie de recherche). Avec la technologie HT, nous pouvons imaginer un outil qui serait à même de résoudre les situations qui prennent systématiquement en défaut les prefetchers matériels des processeurs. Les indirections multiples en sont un excellent exemple. L'idée est de générer à partir d'un même code source, par compilation, un thread qualifié de délinquant (*delinquent thread*) et un thread de calcul (*worker thread*). Ce dernier effectue le travail demandé par le programme. Le premier, le délinquant, est expurgé des instructions de calcul pour ne conserver que les instructions de lecture en mémoire reconnues par le compilateur comme étant potentiellement pénalisantes pendant l'exécution du programme (par exemple, les indirections multiples citées précédemment). Un mécanisme de déclenchement en amont est ajouté au thread de calcul pour initier l'exécution du thread délinquant, suite à la reconnaissance d'un motif d'instructions du thread de calcul. Le compilateur est dans ce cas responsable de l'exactitude de l'arithmétique et de la distance de prefetch. Au final, pendant l'exécution du programme, le thread de calcul et son thread délinquant sont associés aux processeurs logiques d'un processeur physique. Le thread délinquant va remplir son dessin en générant systématiquement et par anticipation des défauts de cache à chacun de ses accès en mémoire. Cette piètre performance est ici du meilleur effet puisque les instructions de lecture du thread de calcul sont, elles, immédiatement servies par les caches L2/L3. Les performances du programme s'en trouvent améliorées puisque le thread principal accèdera à ses données en une dizaine de cycles CPU, là où le délinquant attendra plusieurs centaines de cycles. Et c'est d'autant plus méritoire que les techniques d'optimisation traditionnelles ne peuvent rien contre ces accès chaotiques du point de vue du couple processeur/compilateur.

La prochaine mouture des compilateurs Intel (version 9.0) sera à même de générer partiellement ce type d'optimisation et utilisera un mécanisme comparable à la compilation par profil ou PGO (lire les chapitres 3 et 4).

Comme nous vous l'indiquions précédemment, il n'est pas nécessaire d'avoir un processeur HT pour imaginer un tel mécanisme. En revanche, un système multiprocesseur vrai n'en bénéficiera pas à cause de la séparation physique des mémoires cache des processeurs.

Remarque

Nous verrons plus loin qu'un processeur multi-core qui partagerait un niveau de mémoire cache peut théoriquement bénéficier même partiellement d'un tel mécanisme.

Il existe une troisième voie. Il s'agit de la technologie VMT (Virtual Multi Threading). Dans son principe, VMT est très proche de ce que nous vous avons décrit avec HT. Des motifs d'accès en mémoire imprédictibles sont extraits du code et associés à un thread délinquant. Ce dernier n'est plus activé par le thread principal (*worker thread*), mais par des événements micro-architecturaux qui engendrent des *cache miss*

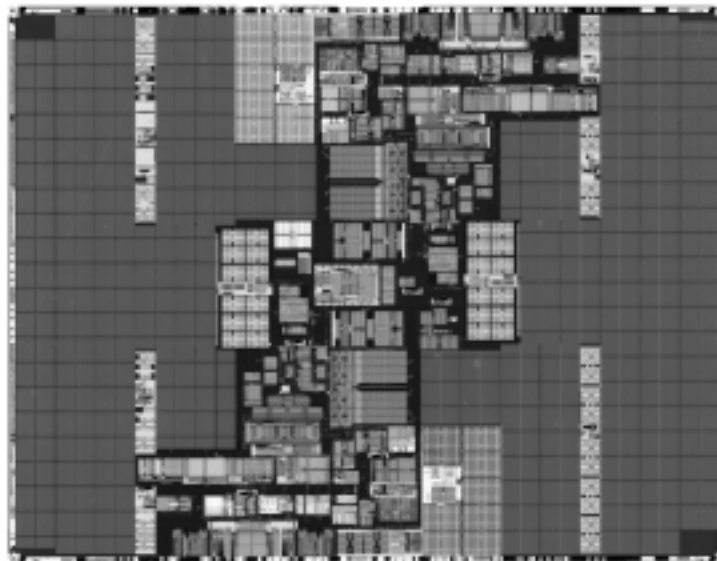
ou défauts de cache. Ils préchargent de ce fait les données à accès problématique dans la mémoire cache pour le thread principal. Mais alors, comment cela peut-il fonctionner avec un processeur Itanium ? Simplement en se souvenant que l'architecture EPIC définit en plus du processeur un ensemble de composants système. Il s'agit des PAL (Processor Abstraction layer) et des SAL (System Abstraction layer). Ils sont aujourd'hui principalement associés aux mécanismes de détection, de confinement, de correction et de *reporting* d'erreurs matérielles. Mais, puisqu'ils permettent la configuration du processeur – sans aucune intervention du système d'exploitation –, les chercheurs d'Intel ont implémenté un mécanisme de changement de contexte ultra rapide à l'instar de ce que le système d'exploitation fait. Là où celui-ci requiert plus d'un millier de cycles CPU, l'implémentation expérimentale de *fast-switching* ou changement de contexte ultrarapide procède à l'entrée et à la sortie du nouveau contexte en une centaine de cycles CPU. Certes, c'est encore un peu long (l'idéal étant une dizaine de cycles par demi-changement de contexte), mais le prototype fonctionne et améliore considérablement les performances des applications testées (SGBDR essentiellement) d'environ 8 % à 20 %. Du point de vue du système d'exploitation, il ne s'est rien produit et au niveau du processeur, seul le PAL a dû être modifié. L'autre avantage de cette méthode est qu'il n'est pas nécessaire de modifier le système d'exploitation. Enfin, s'il est toujours possible d'imaginer que le processeur puisse exposer au compilateur une instruction pour effectuer un tel changement de contexte, celui-ci s'opère pour l'heure uniquement sur les événements architecturaux qui peuvent mener le processeur à un décrochage du pipeline ou STALL (détaillé au chapitre 5). Gageons que cette technologie sera prochainement disponible pour les processeurs Itanium du commerce.

Le concept de cœurs multiples (*multi-core*) n'est pas une révolution en soit, et ne fait que confirmer l'existence des problèmes d'évolution des performances décrits par Pollack. Il s'agit de scinder en tout ou partie le budget de transistors alloué à une puce et de les utiliser pour graver deux processeurs sur la même *die*. Déjà, IBM adopte cette technologie avec ses processeurs Power 4 et Power 5 et propose une forme de parallélisme (SMP) au niveau architectural. D'autres fondateurs, dont Intel, sont en lice pour rendre cette technologie accessible au grand public. Pour autant, la première implémentation de cette formule verra le jour dans le processeur Montecito qui intégrera, entre autres, deux cœurs sur la même die (figures I-1 et I-2).

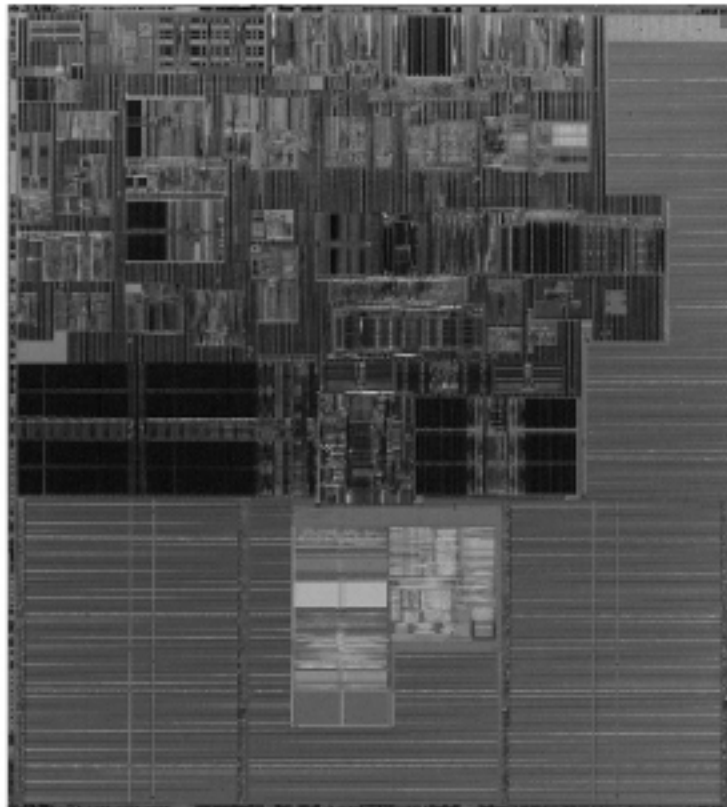
Nous voyons une fois de plus comment la loi de Gordon Moore sera exploitée dans le futur pour répondre à la problématique de la performance. Il faudra réfléchir également aux différentes implémentations possibles du multi-core. Si, pour l'heure, seuls deux cœurs entrent dans la composition des puces multi-core, les fondateurs annoncent de véritables prouesses technologiques à venir. Ainsi, Intel a dans sa feuille de route, après le Montecito, le processeur Tukwila qui ne devrait intégrer pas moins de quatre cœurs Itanium 2 sur la même puce. Avec deux cœurs, différentes implémentations sont déjà possibles. Nous ne savons pas – à part pour le Montecito – quelle organisation sera retenue par Intel pour ses puces multi-core.

Figure I-1

Sur cette image – simulée – de la die du processeur Montecito, nous voyons clairement les deux cœurs symétriques, ainsi que l'arbitre d'accès au bus en position centrale. Nous voyons nettement la prédominance des mémoires cache (26,5 Mo au total).

**Figure I-2**

En comparaison, le processeur Itanium 2 avec 9 Mo de mémoire cache de troisième niveau semble... petit. Il n'intègre pourtant pas moins de 410 millions de transistors, là où le Montecito embarque 1 720 000 000 transistors et est gravé en 90 nm.



Organisations possibles des puces multi-core

Les combinaisons possibles diffèrent de part le niveau de mémoire cache partagé entre les cœurs (L2, L2 et L3, L3 uniquement, aucun partage) et l'arbitrage d'accès au bus frontal du processeur – à supposer qu'il est unique bien entendu.

Il est d'ailleurs fort à parier qu'il subsistera a fortiori plusieurs combinaisons en fonction du niveau de performance exigé par les clients et les applications.

Dans le cas du processeur Montecito, les mémoires cache ne seront pas partagées par les cœurs. Chaque cœur disposera ainsi de 12 Mo de cache L3. Parmi les innovations de cette puce, signalons ici le fait que la mémoire cache L2 n'est plus unifiée comme pour les Itanium 2, et que la L2I aura une taille de 1 Mo (là où la L2D sera de 256 Ko).

Montecito et la technologie Pellston

Le processeur Montecito intégrera la technologie Pellston qui lui permettra une gestion encore plus fine de la technologie MCA (Machine Check Architecture). Il sera ainsi possible à la puce d'invalider dynamiquement des lignes de cache défectueuses ou sur le point de le devenir. Pour sa part, la technologie Foxton apportera à l'Itanium des fonctions d'économie d'énergie – ou plus exactement de confinement de l'enveloppe thermique – équivalentes à ce dont les processeurs mobiles disposent à ce jour.

Les mémoires cache L1I et L1D restent inchangées avec une taille de 16 Ko. En ce qui concerne les latences de ces mémoires, elles devraient être très proches de ce qu'offre le processeur Itanium 2 à 9 Mo de cache L3. Intel prévoit également de revoir et d'améliorer les files d'attente aux mémoires cache (lire les chapitres 2 et 5). Sans entrer ici dans des détails qui pourraient changer, l'accès à l'interface système (bus frontal ou *Front Side Bus*) se fera via le circuit d'arbitrage de bus suivant un mécanisme de ping-pong. Ce mécanisme permettra également, sous certaines conditions exceptionnelles, d'allouer un accès exclusif au bus à l'un des cœurs. Ainsi équipée, une plate-forme quadri-processeur à base de processeur Montecito sera vue par le système d'exploitation et les applications comme un système SMP 16 voies !

En résumé

Conçue conjointement par Hewlett-Packard et Intel comme l'architecture des processeurs pour serveurs haut de gamme pour la prochaine décennie, l'architecture EPIC des Itanium fondus par Intel est vouée à un bel avenir. Déjà adoptée par plus de 70 constructeurs, les processeurs Itanium disposent d'un portfolio applicatif riche de plus de 2 700 références. En outre, des amendements de l'implémentation d'EPIC améliorent régulièrement et de façon notable le niveau de performance de ces processeurs. Et Intel semble bien vouloir remplir ses engagements en proposant une nouvelle mouture de son CPU haut de gamme tous les ans. Ainsi le Montecito, attendu pour l'année 2005, sera la première puce d'Intel à intégrer deux cœurs sur la même die et offrira la technologie

Hyper-Threading. Tukwila, son successeur attendu pour l'année 2006, poussera l'intégration et le parallélisme encore plus loin en proposant au moins quatre cœurs par die et probablement plus. En outre, de nombreuses innovations technologiques verront conjointement le jour telles que le *Virtual Multi Threading*, Foxton, Pellston et bien entendu des progrès dans les techniques de compilation et d'optimisation. Alors, pour profiter dès aujourd'hui de la performance accrue de l'architecture EPIC – et vous donner ainsi accès à ce nouveau marché en pleine expansion –, nous vous souhaitons une agréable lecture.

Bonne optimisation !