Processeurs multicoeurs et parallélisme (Seconde partie)

Nous avons vu dans le dernier numéro les trois types de parallélisme qui pouvaient exister en informatique : à gros, moyen et grain fin. Ces trois types sont exploitables simultanément, mais encore faut-il disposer d'une plate-forme matérielle et logicielle qui le permette…

Maintenant que nous avons brièvement décrit les différentes formes de parallélisme qui existent, se pose la question de savoir si les logiciels que nous utilisons en tirent pleinement profit. À son tour, cette question se scinde en deux : quelle est la responsabilité du système d'exploitation, quelle est celle du logiciel ?

LE MATÉRIEL

La part du matériel dans le parallélisme se limite essentiellement, dans les machines grand public, à celui du gros grain. Rappelons que celui-ci consiste à fabriquer des clusters de machines reliées par un réseau et communiquant, la plupart du temps, par un réseau de type TCP/IP. Comme nous l'avons vu, ce type de parallélisme n'est intéressant que si les temps de traversée du réseau (et des couches du système d'exploitation qui le gèrent) sont courts par rapport au temps d'exécution des tâches partagées. Il faut donc que le réseau soit le plus rapide possible, c'est pourquoi on utilisera, d'une part, la norme Gigabit Ethernet si possible, et, d'autre part, on essaiera de grouper les machines sur le même routeur pour minimiser les temps de transit sur le réseau.

Quand on parle des autres formes de parallélisme, soit l'on s'adresse à des machines professionnelles comportant plusieurs processeurs (chacun pouvant être multicœur), soit à des machines grand public, monoprocesseur mais multicœur. Dans tous les cas, le matériel est adapté au fonctionnement parallèle. Il n'y a donc rien à faire de ce côté-ci.

LE SYSTÈME D'EXPLOITATION

Le système d'exploitation intervient à deux niveaux : celui du gros et du moyen grain.

Pour ce qui concerne le premier, il s'agit essentiellement de proposer des services de communication réseau efficaces. La performance de la pile logicielle TCP/IP est critique. Jusqu'ici, la palme est détenue par le système libre NetBSD. Windows n'a jamais été réputé pour bénéficier de services réseaux très performants. La pile Linux se situe entre les deux.

Pour tirer profit du parallélisme à grain moyen, le système d'exploitation doit proposer un ordonnanceur de bonne facture, qui répartit la charge équitablement entre les différents processeurs, et peut éventuellement être orienté par des primitives particulières (par exemple, on peut décider d'affecter un processeur particulier à une seule tâche, comme la recherche dans une base de données, ou bien décider de garder un processeur pour exécuter des tâches ultra-prioritaires dans une optique de réponse en temps réel). Il est parfois possible de gagner de 20 à 30 % d'efficacité en changeant d'algorithme de permutation de tâche (il en existe plusieurs, avec ou sans priorité, orientés ou non temps-réel…). Dans le même temps, le code de l'ordonnanceur ne doit pas être trop important, pour que le choix de la tâche à exécuter s'effectue dans un temps raisonnable (typiquement, moins de 1 % du quantum élémentaire de temps. Si ce dernier est de 10 ms, l'ordonnanceur doit donc décider en moins de 100 µs).

Mais il faut aussi que le système d'exploitation gère les threads et publie les API au niveau utilisateur qui permettent aux programmeurs de bénéficier des services du parallélisme sous-jacent (création de fils, destruction, synchronisation, partage mémoire). Sous les systèmes modernes Unix, par exemple, on distingue nettement la primitive fork, qui crée un nouveau processus (donc au sens du parallélisme à gros grain), lequel devient totalement autonome au niveau ressource, des appels type thread_create qui créent des fils d'exécution qui partagent le même espace mémoire.

LE NIVEAU LOGICIEL

Bien entendu, pour qu'un logiciel s'exécute en parallèle il faut que les équipes de programmeurs qui l'ont écrit aient pensé parallélisme dès le début. Potentiellement, nombre d'opérations peuvent être parallélisables (calculs, entrées/sorties…), mais il faut parfois réfléchir sur l'organisation du code, envisager des bouleversements parfois majeurs quand l'architecture initiale n'a pas été bien pensée. En outre, les algorithmes parallèles sont plus complexes à appréhender et à implémenter que leurs équivalents linéaires : plutôt que d'être inclus dès le départ dans le code des logiciels, ils sont relégués, pour des questions de temps, à des améliorations ultérieures.

Il faut également bien analyser quelles parties de l'algorithme

bénéficieraient d'un traitement parallèle, et quelle parties ne nécessitent pas une telle mesure. Si les traitements sont trop brefs, le temps passé par le système à créer le contexte d'exécution, le temps perdu dans l'ordonnanceur deviennent significatifs vis-à-vis du calcul pur, et l'efficacité s'en ressent. De même, si trop de données sont partagées, il est indispensable de créer une multitude de sémaphores de synchronisation qu'il faudra gérer, et, au total, le programme s'exécutera toujours selon la cadence du fil le plus lent.

Là-dessus, leur mise au point est délicate : autant il est facile d'utiliser un outil de débogage quand un programme se déroule séquentiellement sur un seul processeur, autant il est difficile de placer des points d'arrêt répartis, de maintenir un ensemble de fils synchronisés, ou de les dérouler en pas à pas alors que d'autres continuent à s'exécuter, même si certains outils comme gdb permettent, dans une certaine mesure, l'examen des fils et des messages de synchronisation. Tout ceci se complique encore sur les machines multiprocesseurs.

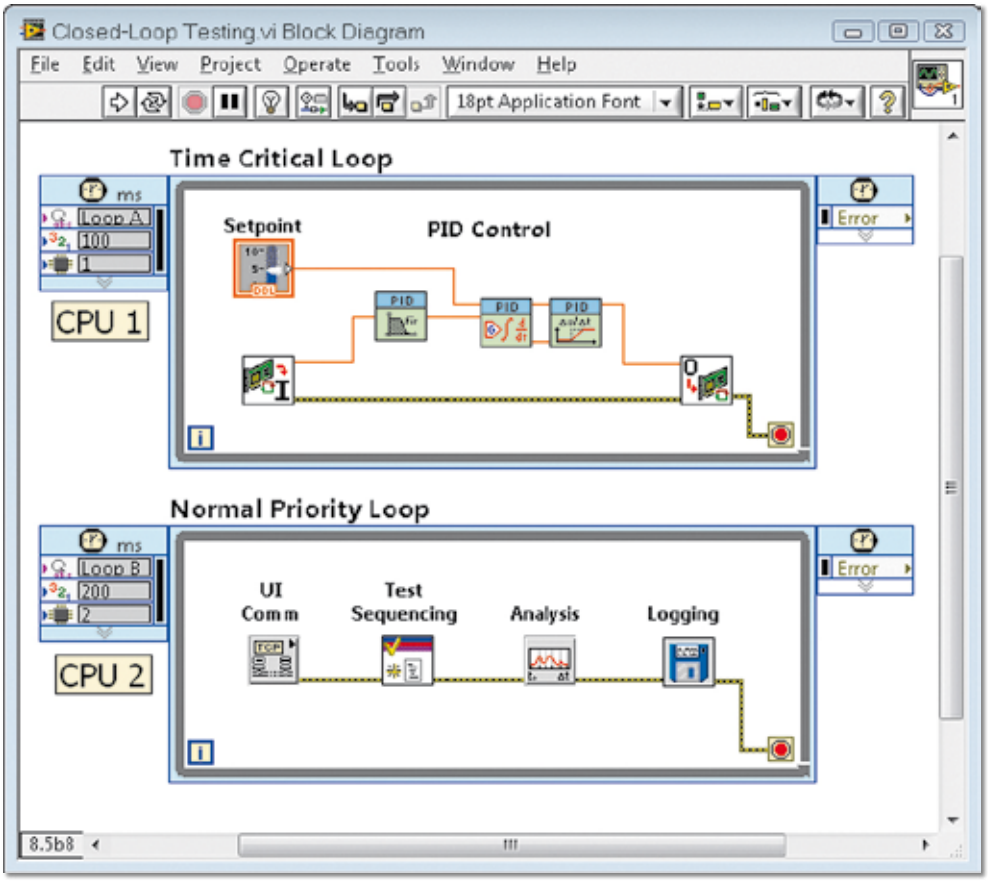
Cela signifie que les logiciels standard sont peu préparés à ce genre d'exécution. La plupart des lecteurs de courrier, par exemple, savent envoyer ou récupérer des messages en tâche de fond, et les navigateurs également. Il ne s'agit là que de tâches élémentaires, qui ne nécessitent ni algorithmie complexe, ni synchronisation. Mais pour les opérations plus complexes, comme les tris, les classements divers, les calculs dans les tableurs, l'affichage 3D, etc., le parallélisme dans les logiciels grand public n'en est qu'à ses débuts. Si tant est qu'il apparaisse un jour, certains logiciels de plusieurs millions de lignes de code étant tout simplement hors de portée d'une restructuration comme celle exigée par la parallélisation.

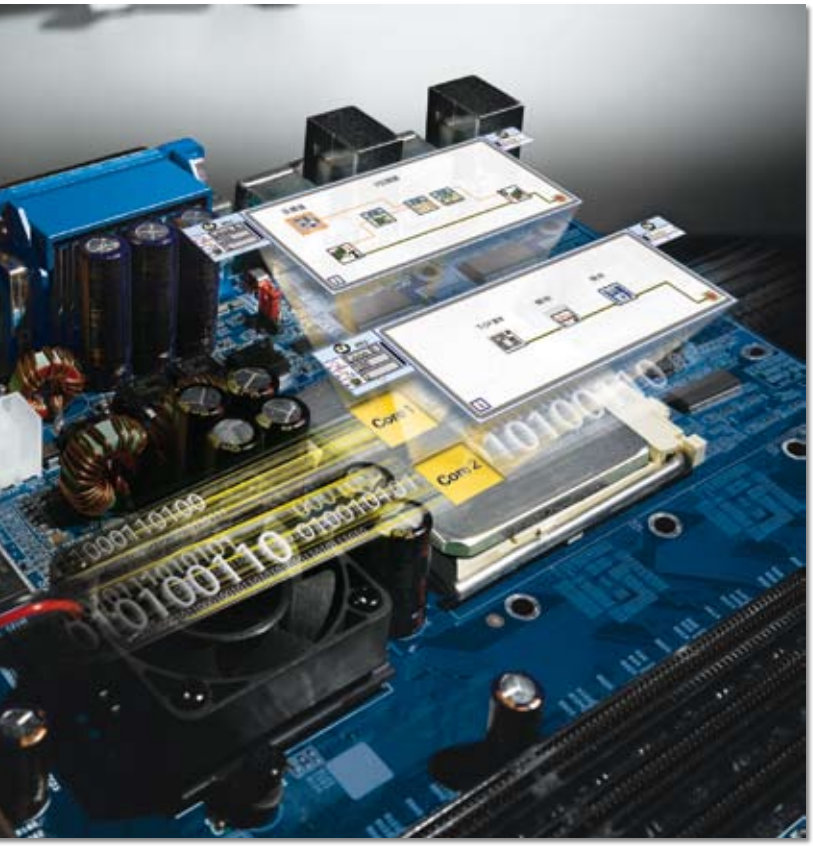
Dans les logiciels professionnels, la donne est différente. Il existe souvent des versions « parallélisées » des principaux logiciels (par exemple Oracle), mais ces dernières sont surtaxées par rapport aux versions simples. Il faut donc accepter de débourser des sommes parfois importantes pour avoir accès aux gains attendus, et ce d'autant que le coût du logiciel est indexé sur le nombre de cœurs sur lequel il est censé tourner.

NIVEAU PROCESSEUR

De ce côté, les principaux compilateurs possèdent maintenant tous des options de compilation qui permettent d'optimiser le code pour les principaux processeurs superscalaires. Ces techniques incluent le brassage d'instructions pour les rendre plus indépendantes et donc plus susceptibles de s'exécuter en parallèle, plus diverses optimisations tenant compte du fonctionnement interne des circuits, comme le recours aux instructions de type SIMD Single instruction multiple data qui permettent d'exécuter répétitivement une opération arithmétique sur un bloc de données sans avoir à écrire une boucle explicite (c'est le cas, sur les processeurs x86, des instructions de type SSE). Dans le monde Intel, on ne sera pas surpris de savoir que ce sont les compilateurs édités par le fondeur qui présentent (pour l'instant) la meilleure performance.

Malheureusement, ces optimisations ne sont disponibles que sur certains types de processeurs. Lorsque les éditeurs de logiciels fournissent un code exécutable, ils sont obligés de choisir nécessairement un minimum de « rétrocompatibilité », c'est-à-dire un modèle de processeur assez ancien, de façon à ce que le logiciel puisse s'exécuter sur la plupart des machines présentes sur le marché. Cela signifie que les « nouvelles » instructions, particulièrement SIMD ne seront pas utilisées (parfois, certains éditeurs embarquent des tests dans leurs logiciels qui permettent de déterminer sur quel type de processeur tourne le programme et de choisir des morceaux de code critiques optimisés en conséquence. C'est souvent le cas sur les décodeurs vidéo, qui nécessitent une puissance de calcul importante). En revanche, le compilateur peut toujours brasser les instructions pour éviter les relations de dépendance.





Au total, cela signifie que, à moins d'opter pour un logiciel libre que l'on compile sur sa machine en utilisant une configuration du compilateur optimisée pour son processeur, la plupart des logiciels en vente (ou disponibles gratuitement) ne sont pas aussi performants qu'ils pourraient l'être, ce qui n'est que la rançon de l'universalité.

CE N'EST QU'UN DÉBUT

Le parallélisme dans les ordinateurs personnels n'en est encore qu'à ses débuts. Il y a une nette marge d'amélioration en ce qui concerne la performance des logiciels. Sur ce plan, c'est encore une fois le logiciel libre qui se trouve à la pointe de la technique : avec la possibilité d'accès au code, s'ouvrent à la fois des perspectives en terme de remplacement des algorithmes standards utilisés par des équivalents parallèles plus performants (ou l'appui sur des bibliothèques logicielles optimisées pour les environnements parallèles) et de compilation adaptée à sa machine, donc capable de tirer pleinement profit des évolutions du matériel. En outre, les systèmes d'exploitation libres comme Linux, les différents BSD (dont MacOS X, qui n'est pas libre, mais dérive de FreeBSD) disposent d'ordonnanceurs performants qui optimisent le séquencement d'exécution des tâches et leur répartition sur les différents cœurs disponibles.

Ce n'est qu'avec la conjugaison de toutes ces conditions que l'on peut penser tirer vraiment profit des ordinateurs modernes ; ce n'est malheureusement que très peu le cas dans le monde Windows, où les contraintes de rétrocompatibilités combinées avec des codes très complexes rendent les évolutions beaucoup plus lentes. Sans parler des performances systématiquement mauvaises du système d'exploitation.

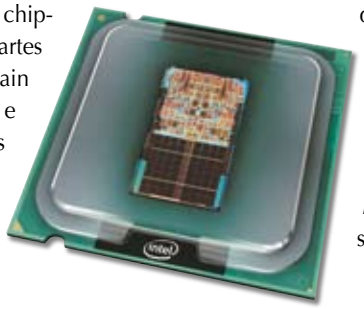
Avec l'arrivée des GPU programmables au travers de langage comme OpenCL, le parallélisme va prendre un tournant décisif : on parle déjà, sur certains algorithmes, d'accélérations de l'ordre de 10 à 100 fois. Encore là aussi faudra-t-il que les concepteurs fassent un effort théorique pour être capable d'utiliser pleinement les ressources qui sont mises à disposition, et que les systèmes d'exploitation prévoient les interfaces nécessaires.

ET DANS LA PRATIQUE ?

Mais que pensent les fournisseurs de matériels d'automatisme de ces multi-coeurs ? Nous avons demandé à plusieurs d'entre eux de répondre. Nous n'avons pas recherché l'exhaustivité mais la représentativité. C'est ainsi que Siemens, National Instrument, Cognex ou Wonderware ont accepté de détailler leur approche.

La première question concernait les aspects matériel. Comment gérez-vous le multi-cœurs et sa mémoire ? Pour éviter les goulots d'étranglement, avez-vous dû modifier l'architecture de votre matériel ?

Pour Siemens, Frederic Lentz précise que « le contrôleur mémoire est intégré dans la puce. Il était jusqu'à présent dans le chipsets des cartes mères. Le gain concerne les temps d'accès du processeur ainsi qu'aux données stockées dans la mémoire ».



Pour sa part, Emmanuel Roset Ingénieur d'Application Marketing de National Instruments utilise « les architectures standards des PC équipés de multi-cœurs en exploitant au mieux les débits des bus vers les périphériques. Les drivers des couches bas niveau des interfaces matérielles intègrent une gestion optimisée du multithreading afin de permettre une exécution parallèle des tâches. Cette gestion optimisée rend d'ailleurs possible des applications de pipelining essentielles à l'accroissement des vitesses de calcul.

En ce qui concerne la mémoire, de nouvelles fonctions spécifiques permettent le découpage de tableaux de données afin que chaque portion soit attribuée à un cœur pour un traitement en parallèle, leur résultat étant concaténés. Le gain en vitesse est ainsi pratiquement doublé sur deux cœurs ».

Siemens ne nie par la possibilité de goulot, mais donne des directions pour les contourner « l'effet goulot d'étranglement peut exister au sein des applications développées pour les processeurs mono-coeurs. Pour éviter cet effet et donc tirer partie de la fluidité de ces nouvelles puces, la programmation d'un logiciel doit être conçue de manière à répartir la charge sur les « n » cœurs. Autrement dit, les threads doivent être traiter en parallèle (d'où la notion de parallélisation), chacun sur un cœur. De nouvelles librairies appelées TPL (*Task Parallel Library*) sont proposées

pour faciliter l'écriture de code capable d'utiliser plusieurs processeurs automatiquement ».

Dans les années à venir, comment allez-vous gérer la pérennité des processeurs avec les duo, les quadri, les XXX cœurs ?

Une question qui n'est pas tabou, elle est même mise en perspective par Frederic Lentz de Siemens pour qui « les fondeurs développent des gammes de processeurs adaptées aux secteurs d'applications (station de travail, serveur, portable, électronique embarquée). Chacun de ses secteurs possède des cycles d'innovation propres qui détermineront les types et cycles de vie des puces. Il est à noter que jusqu'à présent la course à la performance et à l'innovation ne reposait que sur la fréquence des processeurs mono-coeur. L'arrivée des processeurs multi-coeurs est synonyme de performances pour les applications qui gèrent le traitement de parallélisation.

La stratégie des constructeurs d'électronique comme Siemens repose de plus en plus sur la maîtrise et l'adaptation des logiciels aux caractéristiques offertes par les processeurs multi-coeurs.

Les ingénieurs système et les développeurs qui ont vu leur influence diminuer dans les années 90 vont à nouveau être sur le devant de la scène car, à ce jour, le retard est immense entre les applications existantes et le potentiel de performance des processeurs disponible sur le marché. La réponse à la pérennité se trouve là ».

Et visiblement la course n'est pas finie, Emmanuel Roset, de National Instruments, explique que *« National Instruments travaille en étroite collaboration avec Intel ainsi qu'avec des instituts de recherche, comme l'Université de Californie Berkeley. La société y a d'ailleurs installé des bureaux afin de travailler avec les chercheurs les plus avancés dans le domaine de la programmation parallèle. Le langage de programmation graphique Labview a déjà été testé sur une architecture 128 cœurs grâce au concours Supercomputing Analytics Challenge, dont le but de ce concours était de résoudre des calculs pour le télescope E-ELT* (European Extremely Large Telescope).

Un autre défi mondial consiste à être capable d'exécuter 1 million de FFT par seconde grâce aux machines multi-coeurs. NI en est actuellement à 700.000 FFT/s. Enfin, nous travaillons sur un résolveur d'équation différentielles partielles en temps réel basé sur Labview, ainsi que sur la future génération de processeurs many-core, qui intègrera des centaines, voire des milliers de cœurs ».

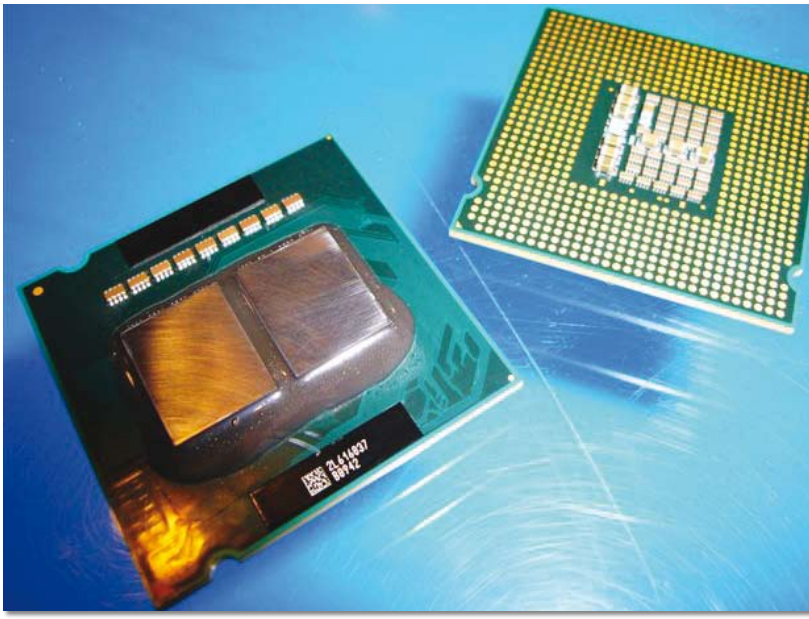
Devant de telles avancées, David Collet - Senior Applications Engineer- chez Cognex reste serein : *« Une fois le logiciel adapté à la gestion de deux cœurs, il est capable sans modification de s'adapter à autant de cœurs que disponibles. Il s'agit réellement de calcul parallèle, qui deviendra massivement parallèle à l'avenir ».*

Au plan du logiciel, le multi-cœur a t-il imposé une refonte du système d'exploitation ?

Certes, les offreurs de matériels d'automatismes ne développent pas leurs propres OS, mais ils sont bien obligés d'en tenir compte, de faire des choix et pourquoi pas de les adapter.

Si Philippe Allot, d'Ordinal, propose une offre qui *« s'appuie sur des systèmes d'exploitation standards (Gamme Windows, Linux) qui ont effectué ce travail de refonte (au moins partiellement) ».*

Frederic Lentz détaille les choix Siemens. *« Le système d'exploitation ne voit ni cœurs ni processeurs mais des «threads». Ce sont ces flux qui sont répartis sur les cœurs. Parmi les applications tirant parties des plates-formes multi-coeurs, Siemens a choisi l'environnement temps réel RTX de la société Ardence.*



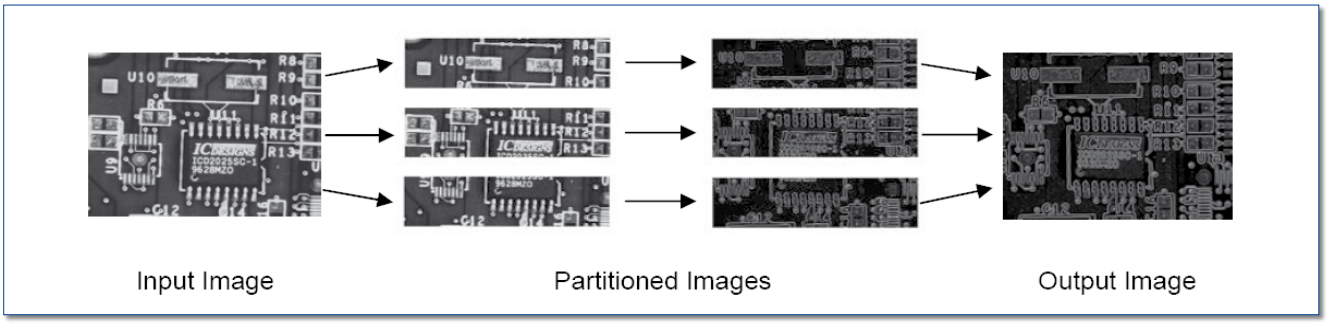
Ce sous-système à la particularité et l'avantage de réserver au boot de la machine, un cœur aux programmes d'automatismes et de contrôle libérant ainsi les autres cœurs pour les programmes attachés au système d'exploitation Windows (NT, XP, Vista).

Par conséquent, il y a séparation physique entre les applications critiques qui sont exécutées sur le « cœur temps réel » *et toutes les autres. Aucune modification du noyau de Windows n'est donc nécessaire et la compatibilité est totale ».*

Les logiciels applicatifs ont-ils été réécrits, adaptés, et dans ce cas que devient la compatibilité entre les versions ? Et la réciproque est-elle vraie d'un logiciel pour processeur multi-cœurs vers un processeur mono-cœur ?

Le passage vers les multi-coeurs étant inéluctable, les offreurs ont pris les devant. Par exemple, Gregory Guiheneuf de Wonderware confirme : *« Nous avons, lors de la sortie de la version 3.0 de la Wonderware System Platform, annoncé le support du multi-cœurs. Cette nouvelle fonctionnalité, transparente pour les développeurs, a néanmoins nécessité de nombreux changements quant au fonctionnement intrinsèque de la plate-forme. Wonderware System Platform est une plate-forme SOA, permettant ainsi de dissocier l'exécution des différents services au sein même de la plate-forme.*

Les fonctionnalités mises en œuvre au travers de la plate-forme Wonderware peuvent donc être réparties sur différents mo-



Exemple de partitionnement d'une image pour un travail sur multi-cœurs. ***Doc. Cognex.***

teurs d'exécution, ce qui permet concrètement de répartir la charge d'une application sur chacun des cœurs d'un processeur. Cette gestion, native à la plate-forme, permet d'augmenter les performances de nos applications, en exploitation mais aussi en développement. Notre System Platform appuie sa configuration sur Microsoft SQL Server 2005, produit qui, lui aussi, répond à ce fonctionnement multi-cœurs ».

David Collet de Cognex confirme que le travail reste important, *« pour nos librairies de traitement, nous avons adapté et testé nos outils majeurs pour les améliorer et les optimiser en vue d'une exécution sur plusieurs cœurs en parallèle. Ces outils, en cas de présence de plusieurs cœurs, sont capables de les utiliser de façon transparente. Le temps d'exécution est alors beaucoup plus court que sur un processeur unique. Le même outil chargé et exécuté sur un processeur monocoeur s'exécutera simplement plus lentement (traitement séquentiel et non traitement parallèle) ».*

De son côté Philippe Allot d'Ordinal rappelle que *« l'offre Coox Supervision et MES est intégralement écrite en Java. Dans le cadre des systèmes multi-cœurs, celle-ci offre deux avantages majeurs :*
a) Java prend en charge une optimisation des tâches systèmes les plus lourdes (calculs graphiques préparatoires, sérialisation avec compression...) dans le contexte d'un système multi-cœurs, au dessus des optimisations propres du système d'exploitation.
b) La programmation Java encourage dans ses bonnes pratiques l'utilisation de « threads », c'est-à-dire la programmation multi-tâches, qui permet à la plate-forme Java de tirer parti « naturellement » d'un système multi-cœur.

Nous avons appliqué massivement ces bonnes pratiques dans notre logiciel Coox dès sa conception, en particulier par la réalisation de services applicatifs, qui peuvent même être déployés sur des machines différentes ».

Et puis il y a ceux qui ont de la chance, comme National Instruments. Emmanuel Roset raconte : *« On peut qualifier l'inventeur de Labview, Jeff Kodosky, de visionnaire. Depuis sa toute première version en 1986, le langage de l'environnement de développement s'appuie sur une architecture de programmation intrinsèquement parallèle. Ce qui permet d'exécuter de multiples boucles sans que l'utilisateur se soucie d'aucune déclaration. Le noyau multitâche intégré au moteur de Labview des premières versions, a été refondu en un noyau multithread depuis de nombreuses années. L'adaptation récente afin d'exploiter les différents threads sur plusieurs cœurs a donc été facilitée. Et cette adaptation ne change en rien la compatibilité monocœur, dans un sens comme dans l'autre ».*

A-t-il fallu développer une ligne de produits spécifiques multi-coeurs ?

Difficile de mettre tout à plat, de tout reprogrammer. Les nouveaux logiciels tiennent forcément compte de cette nouvelle tendance, c'est ainsi que Siemens répond affirmativement à notre question avec « le logiciel automates S7 Simatic WinAC RTX ». Cognex a *« tout d'abord sorti un outil capable d'exécution multicoeur dans une release relativement récente, et nous avons continué cette tendance dans notre dernière version (VisionPro 5.2SR1). Il s'agit d'une évolution de la gamme existante, et non d'une ligne spécifique ».*

Pour Emmanuel Roset : *« Tous les produits de National Instruments sont compatibles avec les processeurs multi-cœurs. Ceci dit, le choix du système d'exploitation est déterminant dans l'efficacité de l'exploitation des cœurs. Par exemple Windows gère en priorité l'automatic load balancing, et il n'est pas possible encore de dédier cent pour cent d'un thread à un processeur spécifique. National Instruments recommande pour des applications critiques qui ont besoin de répartir la puissance de calcul sur chacun des cœurs, un système d'exploitation temps réel. Pour cela, nous proposons un module Labview Real-Time et des cibles PXI multicoeurs supportant le SMP (symetric multi processing). En outre, afin de contrôler que chaque cœur respecte les temps de calculs et gère bien le thread dans le temps imparti, nous proposons l'outil logiciel graphique NI Execution Trace toolkit ».*

Quels sont les retours d'expériences ? 1 + 1 fait-il toujours 2 ? Un mono-cœur tournant à 4G est-il plus ou moins performant qu'un duo-cœurs de 2 + 2 G ?

Il est encore difficile de véritablement parler de retours d'expériences. Confirmation de Philippe Allot : *« Nous n'avons pas de retour d'expérience suffisant pour donner des chiffres précis. Avec une programmation multi-tâche si 1 + 1 ne fait pas 2, il fait 1,8 ou même 1,6 ce qui n'est déjà pas si mal. Ce qui est sûr c'est que l'augmentation de la fréquence des processeurs (le cas 4G) est limité par des contraintes technologiques fortes (densité, échauffement, consommation...) qui ne vont pas dans le sens de l'histoire et tant qu'une rupture technolo-*

gique forte (supraconducteurs à température ambiante par exemple) ne sera pas introduite, seules les technologies multi-cœurs et les solutions logicielles capables d'en tirer parti permettront une augmentation de la performance (à qualité algorithmique égale bien sûr, car dans le domaine du logiciel, une optimisation de haut niveau est toujours plus efficace quand elle est possible) ».

« Le problème est beaucoup plus complexe que cela et il n'y a pas de réponse simple » commente David Collet. *« Les cœurs sont effectivement séparés et fonctionnent en parallèle, comme s'il s'agissait de machines différentes. Par contre, ces CPUs partagent des bus de données, de la mémoire, des interruptions, des éléments de la carte mère qui rend l'ensemble plus ou moins homogène et efficace. 1 + 1 ne fera jamais 2 ».*

Confirmation pour Frédéric Lentz : *« 1 + 1 = 2 ne se vérifie pas. Le gain de performance est de 20 % en moyenne entre deux processeurs de type identique (fréq) mono et double cœur. Il peut aller bien au delà (> 50) si le programme implémente la gestion de la parallélisation des threads. Plusieurs cœurs dans une même puce ne veut pas nécessairement dire traitement « n » fois plus rapide ».*

Emmanuel Roset annonce des retours similaires chez National Instruments, il indique qu'*« il a déjà été démontré que Labview, dans ses dernières versions où chaque partie du code est analysée et découpée en threads automatiquement puis dédiée à un cœur séparé, permet d'obtenir un gain en rapidité de 25 à 35 % dans l'exécution sur un PC double-cœur (à fréquence identique) d'anciennes applications écrites en utilisant du « simple » parallélisme.*

Afin d'obtenir un accroissement très important de la vitesse de calcul, il est nécessaire que les programmeurs utilisent les techniques dites de pipelinning qui sont très simples à mettre en œuvre. Des tests ont permis de faire apparaître dans certains cas, comme dans le traitement d'images par exemple, des gains proches de 1,9 ».

Il complète sa pensée en notant que *« les comparaisons de performances sont difficiles car dépendantes de nombreux facteurs. C'est d'autant plus délicat si la comparaison se fait entre un processeur mono-cœur et un processeurs multi-cœur utilisant une fréquence d'horloge plus basse ».*

Alors faut-il conseiller aux clients de passer au multi-cœur ? Est-ce toujours nécessaire ?

Aujourd'hui pour Gregory Guiheneuf, *« Wonderware suggère très fortement le déploiement de nouveaux projets sur des matériels équipés de processeurs multi-cœurs. L'utilisation de cette dernière technologie permet d'augmenter les performances des applications et il a été aussi constaté que cela permettait d'accélérer les basculements (redondance d'application et de communication) lors de problèmes matériels ou réseau.*

Un des avantages majeurs pour les industriels est de pouvoir simplement bénéficier de cette technologie. En effet, toutes les applications antérieures migrées vers la dernière version de la plate-forme peuvent bénéficier de ces améliorations, sans redéveloppement majeur de l'application. Si plusieurs moteurs d'exécution étaient déjà présents dans l'application, alors ils se répartiront naturellement vers chacun des cœurs du processeur ».

Et les faits sont là comme l'indique Frédéric Lentz : *« A ce jour environ 15 % des applications Windows tirent parti des plates-formes multi-coeurs. En particulier, celles qui gèrent des boucles de calcul, du graphisme et de l'accès aux bases de données.*

Pour les applications déterministes et temps réel, nous recommandons à nos clients de migrer systématiquement sur plateforme multi-coeurs. Elle s'applique aux architectures logicielles intégrant de l'automatisme temps réel Simatic WinAC RTX et des tâches de supervision ou de processus Batch non critiques.

En ce qui concerne la sécurité machines (norme EN 62061 et EN ISO 13849-1), nous commercialisons à partir de Juillet 2009, un automate de sécurité basée sur la technologie Core2 Duo.

La technologie multi-coeurs est parfaitement adaptée au traitement séparé des boucles de sécurité. Enfin, pour les algorithmes de calcul, il est possible à la compilation sous Visual Studio de les exécuter dans le « coeur temps réel » ou dans celui de Windows. C'est le facteur déterminisme qui sera le critère de choix ».

Certes le multi-cœurs n'est pas toujours indispensable rappelle Emmanuel Roset : *« Il n'est pas nécessaire de passer au multi-cœurs dans nombre d'applications peu exigeantes en matière de vitesse et de nombre de données à traiter.*

Mais dès que la puissance de calcul ou la quantité de données à gérer devient un critère important, l'utilisateur gagnera à passer au multi-coeurs, à condition bien sûr de disposer d'un logiciel capable d'optimiser le traitement sur plusieurs cœurs. Sinon, la baisse de fréquence peut se révéler pénalisante. L'intérêt de passer au muticoeur se fait naturellement plus fortement sentir lorsqu'il existe des contraintes temps réel, et que ses contraintes concernent plusieurs tâches ».

De toute façon on vous aura prévenu, le multi-cœurs est partout, que vous le vouliez ou non. David Collet souligne que *« les processeurs actuels sont tous multicoeurs (même un laptop possède 2 cœurs).*

Les machines de bureau haut de gamme possèdent aujourd'hui jusqu'à 16 cœurs ». Il est suivi en ce sens par Philippe Allot : *« Les machines avec lesquelles nous travaillons le plus souvent (PC) intègrent désormais presque toutes des technologies multi-cœurs ».*

Alors vive le Multi-cœurs.