

TechDays 2008 : Sessions sur la programmation parallèle

par Patrick OTTAVI MVP Visual C++ ([Site](#)) ([Blog](#))

Date de publication : 25/02/2008

Dernière mise à jour : 25/02/2008

Le 11, 12, 12 Février j'ai assisté aux **TechDays 2008** en compagnie de quelques collègues (garçons et fille) de la rédaction, Notre point de ralliement était le stand de DVP dans l'emplacement des communautés.

Je vais essayer de vous faire une synthèse des sessions auxquelles j'ai assistées sous forme d'articles.

je commencerai par un sujet qui retient mon attention en ce moment le développement d'applications multithread ou plus exactement la parallélisation du code.

- I - Sessions sur la parallélisation et la programmation multitâches
- II - Conclusions

I - Sessions sur la parallélisation et la programmation multitâches

Lors de ces 3 jours j'ai assisté à trois sessions sur la programmation parallèle et la programmation multitâches, le but étant pour moi de faire le point sur le sujet

Mais avant de rentrer dans le vif du sujet un petit rappel s'impose :

Depuis de nombreuses années, la course à la fréquence sur les microprocesseurs n'a cessé d'augmenter la performance brute de nos applications et ce, sans changement dans le code.

Mais cette vitesse ne peut augmenter indéfiniment à cause de la surchauffe excessive du composant.

Sont ainsi apparues de nouvelles techniques visant à augmenter la puissance,

- Les machines multiprocesseurs souvent réservées aux serveurs.
- Plus récemment l'hyperthreading qui consiste à intégrer au sein du même microprocesseur plusieurs unités logiques.
- Enfin les multi-cœurs : plusieurs microprocesseurs physiques intégrés dans une seule puce.

C'est cette dernière technologie qui pour l'instant semble la plus prometteuse, et qui est développée activement par les deux fondeurs Intel et AMD.

Pas convaincus ?,

un autre argument de taille plaide pour cette technologie : son bilan énergétique .

Le rapport entre la consommation électrique et la puissance d'un processeur n'est pas linéaire.

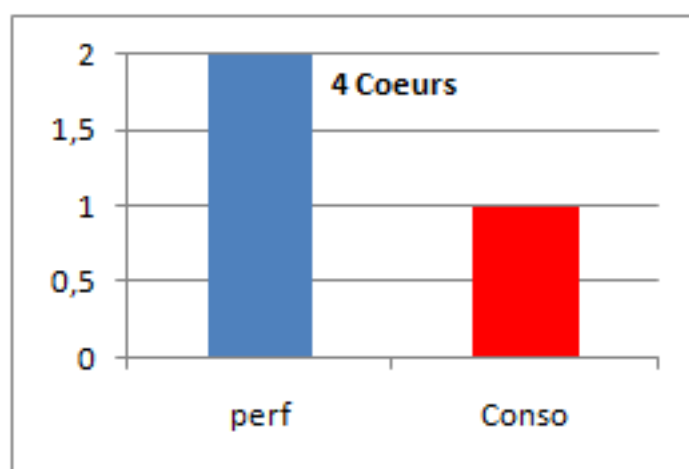
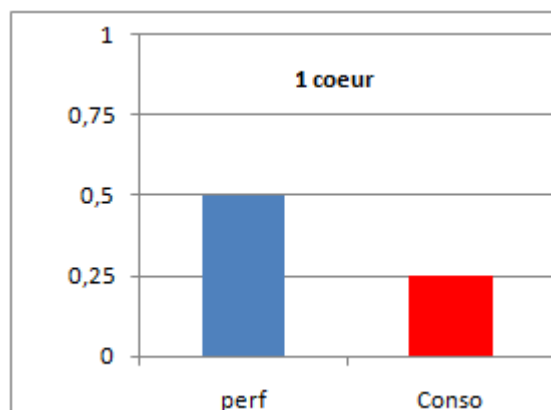
Exemple :

Un processeur mono cœur overclocké de 20% bénéficie d'un gain de performance de seulement 17%, mais voit sa consommation électrique augmenter de plus de 70 % !.

En multipliant les cœurs, on peut réduire la fréquence (et donc la puissance) des cœurs.

Exemple: pour un mono cœur si on diminue les performances de moitié on divise par 4 la consommation.

Avec quatre cœurs on maintient la consommation à 1 et le ratio de puissance passe à deux.



On obtient donc une puissance théorique plus importante pour une consommation électrique contrôlée..

Maintenant que le décor est planté, quelles sont les implications du coté de nos applications ?

Globalement les machines supportent beaucoup mieux le fait d'avoir plusieurs applications qui tournent en même temps, le niveau de performance obtenue dans ce contexte est supérieur à celui d'une machine monoprocesseur.

En résumé elles tiennent mieux la charge.

Mais qu'en est-il pour une seule application ?

Hé bien c'est là que le bât blesse, une application classique mono thread ne tirera pas partie de cette architecture et ne tournera effectivement que sur un microprocesseur physique ou virtuel.

Dans ce contexte Je vous laisse imaginer le rendement, pour un dual c#ur on verra la montée cpu divisée par deux pour un quatre c#urs par quatre etc..

Pour reprendre l'expression de Herb sutter : **"The Free Lunch is Over "** ,

The Free Lunch Is Over: A Fundamental Turn Toward Concurrency in Software

Jusqu'à présent nous étions habitués au fil des ans avec l'évolution des processeurs à bénéficier d'un rendement accru pour nos programmes sans rien faire #

Cette époque est maintenant révolue et il va falloir mettre les mains dans le cambouis.

Alors pour tirer parti de cette architecture, nos applications devront être multithread.

La cause est entendue mais la réalisation est délicate, entraînant des contraintes de coordinations des différents threads, ou tout simplement de partage de données entre thread.

Exemple :

J'ai développé il y a quelques années un service serveur de fichiers séquentiel indexé avec des connexions sockets clientes entrantes, le constat est immédiat ma structure interne est bien multithread mais l'accès a la ressource physique (les fichiers) est mono-thread.

A travers cet exemple on comprend bien que le sujet est délicat#

Quels sont les outils dont dispose un programmeur C/C++ pour bénéficier de cette nouvelle architecture ?

Vaste débat :

Sommes nous condamnés à tous programmer en multithread ?

Les Os actuels tirent-ils vraiment parti de cette évolution ?

Il est clair que nous vivons un moment clef ou le hardware semble avoir pris de l'avance sur le software et qu'il risque d'y avoir quelques grincements de dents chez nos clients qui ne comprendront pas qu'avec une machine plus rapide nos programmes tournent plus lentement..

Dans ce contexte plusieurs solutions visant à simplifier la programmation multithread existent

Comme **OPEN MP** qui permet de la parallélisation explicite.

C'est cette dernière méthode qui nous a été démontrée avec pour exemple le calcul du nombre PI#

Le principe de cette méthode est de placer directement des directives dans le code source pour indiquer les parties de code qui devront s'exécuter de manière parallèle.

L'impact sur le code est assez minime, mais à mon avis réservé à de petites portions de code utilisant des boucles ou des programmes faisant beaucoup de calculs, en ce qui me concerne je ne vois pas comment mettre en #uvre ce type de bibliothèque très orientée calculs scientifiques.

Je ne vais pas rentrer dans les détails sur OpenMp vous pouvez consulter [ce tutoriel](#) sur le sujet :

Une autre bibliothèque nous a été présentée par un intervenant d'**Intel**, **TBB** ou **Intel(R) Threading Building Blocks** .

Cette bibliothèque commerciale est maintenant libérée de cette contrainte et passe en open source sous licence GPLv2 , elle supporte plusieurs plateformes comme Windows, GNU/Linux et Mac OS X et fonctionne sur plusieurs compilateurs Microsoft ,Intel, GCC.

On trouvera nombre de ressources sur le sujet sur le site d'Intel, forum faq etc.

De plus un livre sur le sujet est paru aux éditions **O'Reilly** : **Intel Threading Building Blocks: Outfitting C++ for Multi-Core Processor Parallelism**.

Voici quelques détails de fonctionnalités de cette bibliothèque C++ :

- Un aspect non négligeable: la solution est indépendante du compilateur et de la plateforme.
- **TBB** est orientée **C++** alors que **OpenMP** est plus **C**.
- Il est possible de mixer du code **C++** et **C++/CLI** et d'utiliser **TBB**
- **TBB** est compatible **OpenMP**: il est tout a fait possible d'utiliser les directives **OpenMp** avec **TBB**

On peut la répartir en cinq éléments :

Les algorithmes génériques pour la parallélisation :

- parallel_for
- parallel_while
- parallel_reduce
- pipeline
- parallel_sort
- parallel_scan

Des containers STL thread-safe :

-concurrent_hash_map

-concurrent_queue

-concurrent_vector

Des objets de synchronisation :

atomic, spin_mutex, spin_rw_mutex,

queuing_mutex, queuing_rw_mutex, mutex

Gestion de l'allocation mémoire:

-cache_aligned_allocator

-scalable_allocator

Et enfin un **task scheduler** qui correspond à la gestion d'un pool de thread.

Voilà pour ce bref aperçu qui a l'air prometteur.

Quelques liens :

- [Site Web open Source](#)

- [le produit Commercial](#)

- [Code Demo: Destroy the Castle](#) un exemple qui met en exergue le traitement multithread..

Articles:

- [Demystify Scalable Parallelism with Intel Threading Building Block's Generic Parallel Algorithms](#)

- [Enable Safe, Scalable Parallelism with Intel Threading Building Block's Concurrent Containers](#)

- [Product Review: Intel Threading Building Blocks](#)

Enfin pour terminer, le monde **.Net** n'est pas en reste avec la bibliothèque **Parallel FX (PFX)**, globalement les mêmes principes liés aux boucles sont disponibles avec une syntaxe adaptée.

Je vous conseille la lecture de cet article sur MSDN : [Optimisation du code géré pour les machines multic#ur](#)

Eric Vernié qui était speaker lors de ces sessions a écrit pour les utilisateurs de **C++/CLI** un billet permettant de résoudre des problèmes syntaxiques propres au langage C++/CLI: **[Billet sur l'extension parallèle à .NET PFX et C++/CLI](#)**

Vous pouvez télécharger cette bibliothèque sur ce lien: **[Microsoft Parallel Extensions to .NET Framework 3.5, December 2007 Community Technology Preview](#)**

II - Conclusions

Les trois sessions que j'ai suivies n'ont pas données de réponses immédiates à mon problème, mais beaucoup de questions sur les développements de programmes où la vitesse de traitement est importante.

Penser parallèle risque de ne pas être simple, quelles méthodes employer ?, utiliser de nouveaux patterns ?, comme ceux proposés dans ce livre **Patterns for Parallel Programming**

La nouvelle génération de machines à quatre cœurs étant déjà là, on ne peut plus se permettre d'ignorer le problème#

