TP d'initiation à l'optimisation programmes séquentiels

—

INF-445 STP 3
ENST Bretagne

Ronan.Keryell@enstb.org

29 mai 2006

Ce TP est un extrait d'un TP plus global qu'on peut trouver dans : `http://enstb.org/~keryell/cours/MR2/IAHP/html` et qui sera fait l'année suivante par ceux qui auront le bonheur de suivre le master recherche en informatique.

1 Optimisation de programme séquentiel

Le code du produit scalaire cité en 1.2 montre un exemple de procédure de mesure de temps qu'on peut utiliser dans les TP suivants en général. Chacun peut donc partir de ce programme pour développer les siens.

On programmera en C, voire en C++, pour rester près de la machine. Si on fait du C++ on ne perdra pas de vue que c'est le cœur du programme qui nous intéresse ici et non les interfaces hyper jolies. ☺

1.1 Addition de matrice

On essayera de programmer une addition de matrice qu'on s'acharnera à optimiser.

On instrumentera le code pour qu'il affiche le nombre de MFLOPS et le débit mémoire en Mo/s.

La marche à suivre :

- écrire un beau programme bien configurable[1] ;
- initialiser avec des valeurs quelconques 2 matrices en double précision de taille 1000×1000 ;
- faire un certain nombre de fois des additions de matrices afin d'amortir le temps de démarrage du programme ;
- calculer la complexité de l'algorithme et comparer à la vitesse théorique du processeur[2] trouvée sur Internet ;
- essayer d'optimiser le programme :

[1] Préférer les `enum` nommés aux `#define`. Pourquoi, au fait ?

[2] Pour avoir le modèle de processeur on essayera sous Linux de regarder /proc/cpuinfo, sous Solaris on exécutera `prtconf`, ou de manière générale on farfouillera dans les messages de fonctionnement dans `/var/log/syslog`,... ou renvoyés par l'exécution de `dmesg`.

  - regarder l'assembleur généré par le compilateur (penser à l'option de compilation `-S`) et essayer de retrouver ses petits ;
  - jouer avec les options du compilateur (regarder la documentation et du côté de `-O`) ;
  - inverser les boucles ;
  - tester avec différents types de données (`float`, `double`, `int`, `short int`, `char`...) ;
  - estimer l'importance du cache et de la pagination ;
- faire varier la taille des matrices et regarder comment évolue le temps d'exécution.

Quel est le facteur limitant dans cette histoire ?

1.2 Produit scalaire

On pourra s'inspirer de `~keryell/sources/produit_scalaire/produit_scalaire.c` qui contient aussi des procédures de mesure du temps.

1.3 Multiplication de matrices

Les plus fous pourront essayer de s'attaquer à la multiplication de matrice. C'est un exercice beaucoup plus long et complexe que l'addition...

```
for(i = 0; i < N; i++)
   for(j = 0; j < N; j++) {
      c[i,j] = 0;
      for(k = 0; k < N; k++)
         c[i,j] += a[i,k]*b[k,j]
   }
```

On cherchera à exploiter le maximum d'opérations en parallèle d'une part et à augmenter la localité dans les caches.

1.4 Calcul de fractales

On pourrait essayer de calculer des fractales qui ont l'intérêt d'avoir peu d'accès mémoire par rapport aux calculs.

1.5 Vers une analyse automatique de la machine ?

Concevoir[3] un petit programme de test qui serait capable de déterminer (plus ou moins) automatiquement les caractéristiques de la hiérarchie mémoire simplement en utilisant une routine de mesure de temps :

- taille de la mémoire vive ;
- taille du cache de second niveau et son organisation (taille des lignes du cache, associativité) ;
- idem pour le premier niveau (et éventuellement le troisième niveau le cas échéant) ;
- nombre de registres entiers du jeu d'instructions ;
- nombre de registres entiers physiques totaux utilisés pour le renommage ;
- la même chose pour les registres flottants.

C'est un problème difficile mais intellectuellement valorisant.

[3] On en restera au niveau de la théorie...

1.6 Dénormalisation

Comparer sur une boucle optimisée la différence de temps d'exécution entre des opérations flottantes normales et dénormalisées.