TP5: Plus près du matériel

L'objectif de ce TP est d'observer quelques comportements dûs aux coûts matériels, que ce soit simplement algorithmique, de prédiction de branchement, de parallélisme de niveau instruction (ILP), ...

Vérifiez avec top qu'il ne traîne pas de processus « fou » sur votre machine qui perturberait les mesures. Éventuellement, redémarrez votre machine. Les tests doivent absolument durer plusieurs secondes et répétés plusieurs fois, car vos machines adaptent automatiquement la vitesse du processeur selon la charge de calcul demandée, les première itérations sont donc lentes (il faut « chauffer » la machine).

Faites attention à ne même pas bouger la souris pendant la mesure : surtout dans un environnement KDE/gnome, toute action utilisateur entraîne une flopée de traitements :)

1 Déroulement de boucle

Tout d'abord, observons le coût d'une boucle : le programme `deroulement.c` (que l'on compile avec un simple make deroulement) effectue un calcul tout bête au sein d'une boucle qui a un nombre d'itérations connu. Tel quel, chaque mesure prend quelques secondes.

Déroulez la boucle, c'est-à-dire par exemple définir `D` à 2 pour faire 2 fois moins d'itérations, mais en répliquant le contenu de la boucle pour lui faire faire deux fois le calcul par itération. Attention aux indices de tableau (*vérifiez* que le résultat obtenu est bien le même). Essayez en déroulant 2 itérations, 3, 4, etc. jusqu'à 8 et tracez une courbe (`gnuplot` ...).

Est-il intéressant de dérouler indéfiniment ? Quelle combinaison d'options de gcc permet de dérouler les boucles ?

2 Fusion de boucle

Observez le programme `boucles.c`. Quelle optimisation auriez-vous naturellement tendance à faire ? Le gain obtenu est-il décevant, correct, ou plus que satisfaisant ? Comment l'expliquer ?

Essayez avec 3 boucles, obtenez-vous un gain encore meilleur ? Pourquoi ?

Essayez avec des entiers plutôt que des flottants, mêmes questions.

3 Prédiction de branchement

Observez le programme `prediction.c`, qui compte le nombre de fois qu'apparaissent 0 et 1 dans un tableau donné. Le calcul est effectué plusieurs fois, pour observer combien le processeur arrive à se "souvenir" quelle branche du if il vaut mieux prendre à quel moment.

Compilez en utilisant l'option "`-O1`".

Faites varier le paramètre en entrée de 1 en 1 à partir de 1, puis quand le résultat commence à augmenter, par puissance de deux, tracez une courbe, concluez. (utilisez une échelle logarithmique pour x en utilisant `set logscale x` dans gnuplot).

4 Branchements vs calculs

Observez le programme `branchement.c`, qui compte le nombre de fois qu'apparaissent 0, 1 et 2 dans un tableau donné. Son exécution est en fait relativement lente, on peut aller bien plus vite. Comment ? En évitant les branchements qui cassent le pipeline !

Trouvez donc trois moyens d'effectuer le même calcul sans utiliser de branchement (i.e. la construction `if` notamment). Pensez à exploiter les propriétés des nombres 0, 1 et 2.

5 Processeurs HyperThreadés

Pour cette section, il faut se connecter aux machines de la salle 008 qui disposent de processeurs HyperThreadés. Ainsi, chaque coeur physique de ces machines embarque deux processeurs logiques. Le système d'exploitation voit quant à lui les processeurs logiques comme des coeurs à part entière. Vous pouvez constater dans /proc/cpuinfo que le système d'exploitation considère que ces machines disposent de 16 coeurs, au lieu de 8 en réalité.

Le programme `~sathibau/bin/sched_setaffinity` permet de lancer un shell sur un processeur donné (0, 1, ...). Tous les programmes lancés à partir de ce shell seront toujours exécutés sur ce processeur.

Lisez les programmes `test-int-ht.c` et `test-float-ht.c` . Compilez-les, essayez d'abord de les lancer indépendamment pour constater que la mesure devient stable au bout de quelques secondes (le temps que le processeur « chauffe »).

À l'aide de la commande `lstopo -p` , repérez les numéro de deux processeurs logiques d'un même cœur. Sur le premier processeur logique, lancez un des deux programmes. Lancez ensuite sur le second un des deux programmes, observez l'éventuelle variation. Essayez les 4 combinaisons possibles. Comment expliquer les différents comportements obtenus ?

Dans `test-float-ht.c`, décommentez le calcul sur b. Comment expliquer le changement de résultat ?