Efficacité d'un programme séquentiel

L'objectif est de mettre en œuvre des techniques d'optimisation pour les programmes séquentiels.

1 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 ?

Reproduire l'expérience en ajoutant une troisième puis une quatrième boucle, obtenez-vous une accélération encore meilleure ? Pourquoi ?

2 Déroulement de boucle

Tout d'abord, observons le coût d'une boucle : le programme `deroulement.c` 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, 4 et 8.

Essayez d'optimiser le cœur de la boucle en utilisant des variables supplémentaires ou ne faisant qu'une seule affectation par tour.

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

3 Multiplication et somme de matrices

Les programmes `mul_mat.c` et `som_mat.c` effectuent sommes et multiplications de matrices de façon très basique. Cette façon de faire est loin d'être optimale.

1. Lancer plusieurs fois le programme `som_mat` pour vérifier que les procédures `somMat` et `somMat2` ont un temps d'exécution similaire.
2. Modifier la procédure `somMat2` en permutant l'ordre des boucles sur i et sur j. Mesurer l'accélération obtenue.

   Cette accélération est due à une meilleure exploitation de la mémoire cache qui, grossièrement, charge dans sa mémoire, non seulement la valeur de la case mémoire demandée par le processeur, mais aussi celles de ses cases voisines.
3. A votre avis, quelle case de `t[i][j+1]` ou de `t[i+1][j]` est rangée en mémoire à coté de `t[i][j]` ? Est-ce vraiment toujours le cas ?

4. Recommencer l'expérience en faisant varier la taille de la matrice (prendre N = 1024, N = 256, N = 64). Expliquer les résultats obtenus en vous aidant de la commande lstopo pour connaitre la taille des caches.

On s'intéresse maintenant au produit de matrices.

1. Modifier le code de `prodMat2` afin d'utiliser plus efficacement le cache du processeur. Le gain obtenu est-il décevant, correct ou plus que satisfaisant ?
2. Il est probable que quelques défauts de cache évitables subsistent dans votre code. Les repérez-vous ? Quelle permutation des boucles sur i, j, k induit le plus petit nombre de défauts de cache ? Modifier votre code en conséquence.
3. Lorsque N est assez grand il est probable quelques défauts de cache évitables subsistent dans votre code. Supposons que le cache fasse 8 Mo pour quelle valeur de N apparaissent ces défauts de cache ?

4 Comparaison d'images - juin 2011

On dispose d'un film composé d'un millier d'images codées sur 32 bits au format 1280 × 1024, chaque image pèse donc 5Mo. Pour chaque couple d'images consécutives on cherche à calculer le nombre de pixels différents. Les T images sont présentes dans un tableau déclaré pixel `film[T][1024][1280]` et le résultat dans un tableau noté `long int diff[T-1]` (`diff[i]` contiendra le nombre de pixels différents entre les images i et i+1). Voici un exemple de code calculant ce tableau.

```
for(int n=0; n < T-1; n++)
  for(int i=0; i < 1024; i++)
    for(int j=0; j < 1280; j++)
      if (film[n][i][j] != film[n+1][i][j])
        diff[n]++;
```

1. Comment éviter le test ?
2. Admettons que l'on ait trois images consécutives à traiter sur un seul cœur. Quels sont les défauts de cache regrettables (et en fait tous évitables) générés par l'exécution du programme ci-dessus ? Les tailles de cache sont 32ko, 256ko et 8Mo et la taille d'une image est de 5Mo.
3. Comment réorganiser le calcul afin de minimiser le nombre de défauts de cache ? Il s'agit de décrire un schéma de calcul favorisant la localité des données en assurant que tout pixel ne soit chargé en cache qu'une seule fois en cache L1.
4. Ajouter au fichier `film.c` des fonctions optimisées pour calculer le tableau diff afin de comparer les différentes optimisations.
5. Comparer l'effet de quelques optimisations de compilation.
6. Comment optimiser les défauts de TLB ?