IF4-ARCH Architecture
2011/2012

Travaux pratiques : Évaluation des performances de mémoire CACHE

Objectifs :

1. Apprendre à évaluer les performances de mémoire cache ;
2. Comprendre des paramètres des mémoires cache (taille, associativité, taille de ligne) ;
3. Comprendre les conséquences de la programmation sur l'efficacité d'utilisation de la mémoire cache ;
4. Apprendre à exécuter le profiling du temps d'exécution d'une application.

Exercices :

Prise en main d'outils d'évaluation des performances des mémoires cache

Les mesures des performances d'un système de mémoires cache seront réalisés à l'aide du logiciel libre **valgrind** (http://valgrind.org/). Il rassemble plusieurs outils d'analyse d'utilisation des ressources de mémorisation. Les plus utilisés sont :

- **memcheck** pour détection de fuite dans l'utilisation de mémoire dans un programme : par exemple la mémoire allouée et non libérée) ;
- **cachegrind** pour simulation d'utilisation des mémoires cache.

Nous allons utiliser cachegrind dans ce TP; cet outil réalise une simulation détaillée de tous les niveaux de mémoire cache (L1, D1, L2) du CPU. Attention, avec la simulation, le programme s'exécute de 20 Ã 100 fois plus lentement.

Afin de pouvoir l'utiliser, le programme doit être compilé avec l'option **-g** du compilateur **gcc**. Pour appeler **cachegrind**, il faut écrire la ligne de commande suivante :

valgrind --tool=cachegrind ./nom_programme <options du programme>

Après l'exécution, une partie des résultats est imprimée sur l'écran. Il s'agit des statistiques de tous les niveau de la mémoire cache évalués (Figure 1).

```
==3442== I   refs:      325,695,314
==3442== I1  misses:          1,084
==3442== L2i misses:          1,071
==3442== I1  miss rate:        0.00%
==3442== L2i miss rate:        0.00%
==3442==
==3442== D   refs:      169,916,905  (120,559,278 rd   + 49,357,627 wr)
==3442== D1  misses:        625,845  (    313,529 rd   +    312,316 wr)
==3442== L2d misses:         11,348  (      1,548 rd   +      9,800 wr)
==3442== D1  miss rate:         0.3% (        0.2%     +        0.6%  )
==3442== L2d miss rate:         0.0% (        0.0%     +        0.0%  )
==3442==
==3442== L2 refs:           626,929  (    314,613 rd   +    312,316 wr)
==3442== L2 misses:          12,419  (      2,619 rd   +      9,800 wr)
==3442== L2 miss rate:          0.0% (        0.0%     +        0.0%  )
```

Figure 1 : Exemple des résultats de d'outil cachegrind

En même temps, un fichier sera créé avec le nom cachegrind.out.3442 (3442 est le numéro PID du processus analysé). Vous y trouverez les paramètres de la mémoire

cache considérés pour simulation ainsi que les évènements collectés (Figure 2).

```
desc: I1 cache:         32768 B, 64 B, 8-way associative
desc: D1 cache:          32768 B, 64 B, 8-way associative
desc: L2 cache:         6291456 B, 64 B, 24-way associative
cmd: ./test2 im2.pgm 3
events: Ir I1mr I2mr Dr D1mr D2mr Dw D1mw D2mw
```

Figure 2 : Exemple du contenu du fichier cachegrind.out.3442

Exercice 1 :

Écrivez un programme en C qui réalise allocation dynamique de deux matrices, matrice A et matrice B chacune de taille au minimum 1000 x 1000 éléments type int. Ce programme va initialiser la matrice A à zéro et recopier les éléments de la matrice A dans B. Vous allez évaluer trois version différentes de ce programme :

- Version 1 : parcours des matrices colonne par colonne, sans utilisation des fonctions optimisées (memcpy ou autre) ;
- Version 2 : parcours des matrices ligne par ligne, sans utilisation des fonctions optimisées (memcpy ou autre) ;
- Version 3 : utilisez des fonctions optimisée du langage C memcpy pour réaliser les objectifs du programme.

Questions :

1. Pour chaque version du programme, effectuez l'évaluation des performances de mémoire cache et analysez les et comparez les.
2. Identifiez les paramètres du système de mémoires cache de votre ordinateur et représentez le graphiquement.

Exercice 2

Question :
Pour la version 2 et 3 du programme de l'exercice précédant, faites varier les paramètres de la mémoire cache D1 et analysez les conséquences sur la performance (exprimée en nombre de « misses »). Pour chacune des deux versions citées, remplissez un tableau des résultats et créez des graphes (axe x : taille de mémoire D1, axe y : total data misses). Que pouvons-nous conclure ?

Attention, Pour réalisez ces mesures, automatisez les tests, c'est-à-dire, par exemple, créez les script « shell » qui vous permettent lancer des jeux de tests avec différents paramètres, et en même temps pré-traites les résultats pour faciliter leur exploitation lors de la rédaction du rapport. Vous pouvez vous inspirer un exemple joint à l'énoncé de ce TP.

Tab. 1 : «Total Misses» pour niveau D1

Input image : 640x480, Kernel size : 5	Taille de mémoire D1				
Line size : 16 B	1K	2K	4K	8K	16K (optionnel)
1 way - associative					
1 way - associative					
4 way - associative					
8 way – associative					

Commande de valgrind pour modifier des paramètres de la mémoire D1 :

valgrind –-tool=cachegrind –-D1=mem_size, associativity, line_size
./nom_programme <options>

Exercice 3 :

Recopiez les fichiers sources (www.esiee.fr/~dokladae section Architecture, TP Mémoire Cache) dans un répertoire sur votre poste de travail, analysez les fichiers sources et expliquez ce que fait le programme. Expliquez sa fonction, types de données traitées, comment elles sont accédées, il s'agit du traitement local ou global, régulier ou irrégulier ?, ...

Commande pour utiliser le programme :

./nom_programme <input_image.pgm> <kernel_size>

En utilisant ce programme, effectuez :

1. Mesure de temps d'exécution et profiling

Afin de mesurer le temps d'exécution d'une application et obtenir les temps d'exécution fonction par fonction, nous utilisons des logiciels de profiling. Dans ce TP, nous allons faire appel au gprof, un logiciel libre (http://www.cs.utah.edu/dept/old/texinfo/as/gprof.html#SEC2).

Pour l'utiliser, il faut compiler votre application avec l'option -**pg** du gcc, par la suite exécuter votre programme. Un fichier gmont.out sera créé. Afin d'accéder au informations du profiling, il faut traiter ce fichier avec la commande suivante :

gprof nom_programme

Exécutez le profiling sur le programme quelles sont les informations fournies par le gprof ? Attention, Les résultats obtenus par une seule exécution ne sont pas statistiquement représentatifs. Afin de les préciser, il faut suivre la procédure suivante :

Run your program once.

1. Issue the command `mv gmon.out gmon.sum'.
2. Run your program again, the same as before.
3. Merge the new data in `gmon.out' into `gmon.sum' with this command:

   ```
   gprof -s executable-file gmon.out gmon.sum
   ```

4. Repeat the last two steps as often as you wish.
5. Analyze the cumulative data using this command:

   ```
   gprof executable-file gmon.sum > output-file
   ```

Questions :

1. Effectuez le profiling (sur au minimum 5 exécutions) du programme fourni. Quelle partie de l'application fournie consomme le plus du temps d'exécution ?
2. Effectuez une évaluation des performances de la mémoire cache pour l'application (avec les paramètres par défaut).
3. Optimisez les accès à la mémoire de la partie identifiée en 1. ; et vérifiez l'efficacité de votre solution à nouveau avec cachegrind.
4. Avec cette version optimisée, pour au moins 5 tailles d'image d'entrée

significativement différentes, tracez dans le même graphe les courbes des temps d'exécution et des miss de données au niveau D1 et L2. Pouvons nous en déduire une conclusion ? Automatisez cette partie du test.

5. Pour une taille d'image d'Avec les paramètres par défaut de votre poste de travail, identifiés dans l'exercice 3, mesurez les performances du système mémoire (pour la taille du noyau <= 10) et calculer la pénalisation selon la formule présentée en cours :
   On suppose que temps d'accès D1 : 1 cycle en cas de succès, 10 cycles en cas de miss, L2d : 10 cycles en cas de succès, 100 cycles en cas de miss.