

Remarque : un rapport est à rédiger et à rendre dans les délais annoncés en cours.

Objectifs :

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

Exercices :

1. *Mettre en place le programme et comprendre sa fonction :*

Recopiez les fichiers sources dans un répertoire sur votre poste de travail, analysez les fichiers sources et expliquez ce que fait le programme (détaillez sa fonction, types de données traitées, comment elles sont accédées, il s'agit du traitement local ou global ? régulier ou irrégulier ?, ...).

2. *Compiler le programme, préparer le programme pour les mesures (le profiling) de mémoire cache et l'exécuter :*

Les mesures de performances d'un système de mémoires cache sera réalisé à l'aide du logiciel **valgrind** (<http://valgrind.org/>). Il rassemble plusieurs outils d'analyse d'utilisation des systèmes mémoire, les plus utilisés sont : memcheck (détection de fuite de mémoire dans un programme : mémoire allouée et non libérée), cachegrind (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 à 100 fois plus lentement.

Afin de pouvoir l'utiliser, le programme doit être compilé avec l'option **-g** du compilateur **gcc**. Compilez le programme, et vérifiez le fonctionnement en l'exécutant. La ligne de commande du programme :

```
./test2 <input_image.pgm> <kernel_size>
```

3. *Appeler l'outil cachegrind, effectuer la première analyse et comprendre les résultats :*

Pour appeler cachegrind, il faut écrire la ligne de commande suivante :

```
valgrind --tool=cachegrind ./test2 <input_image.pgm> <kernel_size>
```

Après l'exécution, une partie des résultats est imprimée sur l'écran (résultats sur les cache miss). Il s'agit des statistiques sur les « misses » à tous les niveaux de la mémoire cache :

```

==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% )

```

Une deuxième partie des résultats sera mise dans le fichier cachegrind.out.3442 (3442 est le numéro pid du procesus analysé). Vous y trouverez les paramètres de la mémoire cache simulée et les événements collectés :

```

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

```

A quoi cela correspond (combien de niveau, quels types de mémoire cache, unifiées, non unifiée, à quel niveau)? A partir des information dans ce fichier, dessiner le schéma du système de mémoire cache et complétez le par les paramètres obtenus. Quels sont les événements collectés – à quoi cela correspond ?

Regardez le manuel pour plus d'informations :

<http://valgrind.org/docs/manual/cg-manual.html#cg-manual.profile>

4. *Varié les paramètres de la mémoire cache et observer les conséquences sur la performance du système de mémoire :*

Pour une taille d'image d'entrée et une taille du noyau du filtre donné (5), faites varier les paramètres de la mémoire cache D1 et analysez les conséquences sur la performance (exprimée en nombre de « misses »), remplissez le tableau ci-dessous et visualisez le résultat dans un graphe (axe x : taille de mémoire D1, axe y : total data misses).

Tab. 1 : «Total Misses» pour niveau D1

Input image : 640x480, Kernel size : 5	Taille de mémoire D1				
	1K	2K	4K	8K	16K (optionnel)
Line size : 16 B					
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 ./test2  
<input_image.pgm> <kernel_size>
```

5. *Calculer la pénalisation liées aux «data misses » et optimisation :*

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 (Cam, cycles d'attente mémoire).

Analyser le programme (la fonction de traitement d'image) et proposez une modification d'écriture de cette fonction afin de réduire cette pénalisation. Vérifier le résultat par la simulation et comparer des résultats obtenus.

6. *Mesurer le temps d'exécution, effectuer le profiling :*

Afin de mesurer le temps d'exécution d'une application et obtenir les temps d'exécution fonction par fonction, nous utilisons des logiciel de profiling. Dans ce TP, nous allons faire appel au gprof (<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 gmon.out sera créer. Afin d'accéder au informations du profiling, il faut traiter ce fichier avec la commande suivante :

```
gprof <nom du programme > ouput_file
```

Exécutez le profiling sur le programme utilisé pour les exercices précédents quelles sont les informations fournies par le gprof ? Les résultats obtenus par une seule exécution ne sont pas statistiquement représentatifs. Afin de préciser les résultats, nous devons exécuter le programme plusieurs fois et fusionner les résultats partiels selon le procédé suivant :

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
```

Effectuez le profiling (sur au minimum trois exécutions) du programme fourni, de sa version originale et optimisée dan l'exercice 5 pour réduire les « data misses ». En améliorant l'utilisation de la mémoire cache, est-ce que vous avez obtenu l'accélération de l'exécution de votre programme ?

7. *Pour allez plus loin :* Refaire les exercices 4 -6 avec la fonction mediane (dans fonction.c)

