Évaluation des performances d'un programme et optimisations de code par compilation

Sid Touati
Professeur à l'université Côte d'Azur

1 Réalisation des benchmarks

Pour les besoins de ce TP, munissez vous du code source d'un de vos projets d'étudiants programmés en C qui contiendrait idéalement plusieurs fonctions appelées. Il vous servira de programme d'exemple pour ce TP.

Si vous n'en avez pas, je vous propose de coder rapidement diverses fonctions de calculs matriciels (calculs en valeurs flottantes), dont les pseudo-codes sont décrits ci dessus :

1. Addition de deux vecteurs de valeurs flottantes $C = A + B$:

```
for (i=0; i< N; i++){
 C[i] = A[i] + B[i];
}
```

2. Multiplication de deux matrices $C = A \times B$:

```
for (i=0; i< N; i++){
     for (j=0; j< M; j++){
           for (k=0; k< P; k++){
               C[i][j] = C[i][j]  + A[i][k] * B[k][j];
             }
     }
}
```

3. Multiplication d'un vecteur par un scalaire $A = A \times s$:

```
for (i=0; i< N; i++){
 A[i] = A[i] * s;
}
```

Quelques conseils :

— Concernant les matrices/vecteurs à déclarer dans vos codes, il faut tester deux variantes :
  (a) Des données déclarées statiquement comme globales ;
  (b) Des données allouées dynamiquement `malloc`.
— Les tailles des matrices/vecteurs doivent être plus grandes que le double de la taille du plus grand cache processeur. Ainsi, il sera garanti qu'aucune données ne sera stockée sur le cache.
— Vos codes doivent renvoyer un résultat calculé (en imprimante une case au hasard de la matrice résultat par exemple). Autrement le compilateur pourra éliminer tout code mort qui ne contribue à aucun résultat. Créez un programme principal qui initialise des matrices et des vecteurs avec des valeurs aléatoires, puis fait des appels à ces fonctions de calculs matriciels.

2 Mesures des performances et profilage d'un programme

1. Évaluez le temps d'exécution de votre programme en utilisant la commande `time`. Est ce que les performances sont stables d'une exécution à une autre ? Quel est la proportion du temps d'exécution passé en mode utilisateur par rapport au mode système ?
2. Comment calculer les métriques de performances suivantes ? CPI, IPC, GFLOPS.
3. Profilez votre application avec l'outil `gprof`. Déduisez les fonctions qui ont le plus grand temps d'exécution.

3 Optimisations de code avec le compilateur gcc

Le but de cet exercice est de vous familiariser avec les options d'optimisation de code du compilateur. Nous utiliserons quelques options simples pour débuter : `-O1`, `-O2`, `-O3`, `-Os`.

1. Examinez succinctement le manuel de gcc (`man gcc`) pour déduire ce que fait chacune des options de compilation ci-dessus.
2. Pour chacune des options d'optimisation du compilateur (`-O1`, `-O2`, `-O3`, `-Os`) :
   (a) Recompilez votre application en mesurant le temps de compilation avec la commande `time`. Attention, si votre code source n'imprime aucun résultat, l'optimisation du compilateur peut effectuer une élimination de code mort.
   (b) Mesurez à nouveau les performances de votre application.
   (c) Quel est votre meilleur taux d'accélération obtenu pour votre application ? (défini comme étant le rapport entre le temps d'exécution initial de votre application et son taux d'exécution optimisé).
3. Maintenant, nous allons utiliser des méthodes d'optimisation du compilateur qui se basent sur des informations de profilage. En consultant le manuel de gcc, étudiez ce que fait l'option `-fprofile-generate`. Recompilez votre application avec l'option `-fprofile-generate`, puis exécutez la.
4. En consultant le manuel de `gcc`, étudiez ce que fait l'option `-fprofile-use`. Recompilez votre application avec l'option `-fprofile-use`, puis exécutez la. Avez vous observé une amélioration des performances ?

4 Optimisations de code avec le compilateur d'Intel `icc` ou `icx`

`gcc` est un compilateur libre, multi-plateformes : il est disponible sur la presque la totalité des architectures des processeurs généralistes. Il existe des compilateurs industriels non libres qui sont fournis ou vendus par les constructeurs de processeurs. Ces compilateurs industriels non libres optimisent le code généré sur des architectures spécifiques. Par exemple, le compilateur `icc` d'intel produit des codes C très efficaces sur les processeurs intel. La version C++ est `icx`.

Je vous invite à télécharger le paquet de logiciels Intel sous Linux, nommé Intel® HPC Toolkit, il gratuit pour les étudiants. Vous pouvez le retrouver en suivant le lien de téléchargement mis à votre disposition dans l'espace de cours, ou en recherchant sur internet *Intel HPC Toolkit*. Une fois le compilateur Intel installé en suivant la documentation en ligne, refaites l'exercice 3 en utilisant le compilateur `icx` ou `icc` au lieu de `gcc`, et comparez les performances obtenues avec icc versus gcc. Si seulement le compilateur icx est installé et non pas icc, faites l'exercice avec icx uniquement.

5 `man icc, man icx, man gcc`

Consultez les manuels des compilateurs `icx` ou `icc` et `gcc` pour lister toutes les optimisations activées à chaque niveau `-O1`, `-O2`, `-O3`, `-Os`.

6 Librairie `MKL` d'Intel

Intel fournit une librairie optimisée pour les calculs numériques (dont la multiplication de matrices). C'est la librairie *Math Kernel Library*. Elle est fournie dans la suite *Parallel Studio*. Concernant la multiplication de matrices, il y a plusieurs codes possibles, le plus connu est celui de la routine `dgemm`. Cherchez sur internet comment utiliser la routine `dgemm` de la librairie `MKL`, et comparez ses performances avec celui de votre code compilé avec `icc -O3`.

7 Compteurs matériels de performances (sur Linux/x86)

Dans les exercices précédents, vos benchmarks effectuent des traitements simples pour tester et évaluer les performances d'un programme sur un processeur. Il est parfois très difficile d'analyser et comprendre les performances obtenues en examinant uniquement les temps d'exécution mesurés en micro-secondes. Lorsqu'un processus s'exécute, il déclenche le fonctionnement de divers composants micro-architecturaux du CPU, qui sont souvent complexes à analyser avec des mesures d'expériences simple comme le temps d'exécution.

Afin d'aider un peu plus l'analyse et la compréhension des performances observées, les compteurs matériels de performances (registres spéciaux), si présents dans le CPU, peuvent être exploités. Ces compteurs enregistrent plein d'événements matériels relatifs aux performances qui sont provoqués par l'exécution de processus : nombre de cycles d'horloge, nombre d'accès mémoire, nombre d'accès au cache L1, nombre de défauts de cache, etc. La nature des événements enregistrés par les compteurs matériels de performances dépend d'une architecture de CPU à une autre. La portabilité n'est donc pas assurée, mais les mesures sont extrêmement précises.

Notons deux façons pour utiliser les compteurs matériels de performances :

1. **Analyser les événements matériels d'un CPU qui exécute plusieurs programmes en concurrence**. Par exemple, l'outil `Likwid` installé sur vos machines de l'université récupère les valeurs des compteurs matériels par cœur (et non pas par processus). Il peut agréger les compteurs matériels de tous les cœurs. Cette façon d'utiliser les compteurs matériels de performances permet d'avoir une vue globale du fonctionnement et des performances d'un cœur ou d'un système entier qui exécute plusieurs processus en même temps.
2. **Analyser les événements matériels provoqués par un seul processus qui s'exécute sur une machine**. C'est la façon à utiliser pour analyser et éventuellement optimiser les performances d'un programme particulier qui s'exécute sur une architecture matérielle précise.

Dans cet exercice nous allons utiliser la deuxième méthode ci-dessus. Là aussi, notons deux façons pour utiliser des compteurs matériels de performances pour un code donné :

1. Soit l'utilisateur est intéressé par l'analyse des performances d'un bout son programme (par exemple étudier les performances d'une boucle précise ou d'une fonction particulière). Dans ce cas, il faut détenir le code source pour l'instrumenter (le modifier). Le code modifié contiendrait des instructions utilisant des librairies d'accès aux compteurs matériels de performances. La librairie `LibPFM` permet par exemple à un programme d'accéder aux valeurs des compteurs matériels de performances.
2. Soit l'utilisateur est intéressé par l'analyse des performances d'un processus entier (pas uniquement un bout du programme). Dans ce cas de figure, l'utilisateur n'a pas besoin d'instrumenter le code source. Il peut utiliser des outils en ligne de commande comme `perf` pour évaluer les performances du code binaire.

A titre d'illustration, utilisons la commande `perf` pour analyser les performances des micro-benchmarks des exercices précédents.

1. Listez les événements matériels pouvant être enregistrés par la commande `perf` sur votre machine de test. À cet effet, la commande `perf list` peut vous aider. Intéressons nous uniquement aux

événements matériels liés à la hiérarchie mémoire : nombre de *loads* et de *stores* exécutés, nombre de défauts de cache, nombre d'accès au TLB, nombre de défauts TLB, etc.

2. Pour enregistrer les événements matériels provoqués par un processus, utilisez la commande `perf record` suivie du code binaire à exécuter. Lorsque le programme binaire finit de s'exécuter, un fichier `perf.data` est généré. La commande `perf report` permet de lire ce fichier `perf.data` et d'afficher un rapport. Testez cela sur vos codes précédents.
3. Utilisez la méthode décrite ci-dessus pour analyser les événements matériels suivants :
   (a) Le nombre d'instructions exécutées ;
   (b) Le nombre d'instructions mémoire exécutées ;
   (c) Le nombre de défauts de caches L1 et L2 ;
   (d) Le nombre de branchements exécutés.