Transformations de boucles

Sid Touati
Professeur à l'université Côte d'Azur

A titre d'illustration pour ce TP, prenons le code d'une multiplication de deux matrices à valeurs flottantes $C = A \times B$:

```
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];
             }
     }
}
```

Les tailles des matrices doivent être assez larges pour que les données ne puissent pas résider dans les divers caches : choisissez une taille de matrice qui a au minimum le double de la capacité du plus grand cache de votre machine. Déclarez vos tableaux statiques comme tableaux globaux, à l'extérieur de la fonction `main`.

Les codes sources de ce TP doivent être optimisés avec le niveau `-O2` au maximum du compilateur. Au niveau d'optimisation supérieur (`-O3`), le compilateur risque d'effectuer des transformations de boucles qui perturberaient vos exercices.

1 Déroulage de boucle (*loop unrolling*)

1. Déroulez 7 fois la boucle la plus interne (boucle k).
2. Compilez et exécutez votre application (n'oubliez pas l'optimisation au niveau `-O2`). Quel est le facteur d'accélération obtenu par rapport à un code sans déroulage de boucle ?
3. Le compilateur gcc peut aussi dérouler des boucles avec l'option `-funroll-loops`. Utilisez cette option (combinée avec `-O2`) pour tester si elle produit de meilleures performances par rapport au déroulage de boucle au niveau source.

2 Fusion de boucles (*loop fusion*)

1. Produisez un code qui déroule deux fois la boucle extérieure i. Attention, votre code doit rester correct pour quelques soient les valeurs des paramètres `N, M, P`.
2. En vous appuyant sur le code produit à la question précédente, vous avez normalement obtenu trois copies pour la boucles j. Pouvons nous fusionner les trois boucles j ? Donnez le code produit et testez ses performances.

3 Permutation de boucles (*loop interchange*)

Le pseudo-code de multiplication de matrice proposé au début de ce TP contient trois boucles imbriquées dans l'ordre (i, j, k). Mais en fait, les dépendances de données entre les instructions du code permettent de

faire la multiplication de matrices en changeant l'ordre des boucles. N'importe quel ordre de boucles pour la multiplication de matrices produit un code correct : (i, k, j), (j, i, k), (j, k, i), etc.

1. A priori, sans effectuer d'expériences, quel serait l'ordre de boucles qui apporterait une amélioration de performances ? Pourquoi ?
2. Testez votre hypothèse en comparant les performances obtenues avec plusieurs ordres de boucles.
3. Le compilateur gcc peut aussi effectuer des permutations de boucles avec l'option `-floop-interchange`. Utilisez cette option (combinée avec `-O2`) pour tester si elle produit de meilleures performances par rapport à votre code optimisé au niveau source.

4 Tuilage ou blocage de boucle (*loop tiling, loop blocking)*

Le tuilage de boucle permet de transformer le code pour que la boucle la plus interne puisse faire des calculs sur un ensemble réduit de données qui tiendrait dans le cache. Ainsi, la boucle la plus interne effectue des calculs sur une *tuile* ou *bloc* de données chargées dans le cache, permettant aux calculs d'aller très vite avant de passer à la tuile suivante. Par exemple, le pseudo-code de multiplication de matrices deviendrait :

```
for (i0 = 0; i0 < N; i0 += B)
    for (j0 = 0; j0 < M; j0 += B)
       for (k0 = 0; k0 < P; k0 += B)
          for (i = i0; i < min(i0 + B, N); i++)
            for (j = j0; j < min(j0 + B, M); j++)
              for (k = k0; k < min(k0 + B, P); k++)
                 C[i][j] += A[i][k] * B[k][j];
```

Dans le pseudo code ci-dessus, le paramètre B (qui veut dire Bloc) doit être calculé pour que les données traitées par la boucle la plus interne tiennent dans le cache. Dans notre cas d'une multiplication de matrices, la boucle la plus interne accède à trois sous matrices (A, B, C) de taille B^2 chacune. Ainsi, si C est la capacité du cache de données (en octets), la contrainte que doit vérifier le paramètre B est la suivante :

$$3 \times B^2 \times sizeof(float) \leq C$$

Comme vous pouvez le constater, la valeur du paramètre B dépend de la capacité du cache ciblé.

1. Calculez une valeur appropriée pour B selon la capacité de cache de votre machine de test. Testez les performances obtenues.
2. Le code obtenu après tuilage de boucles a généré six boucles imbriquées. Utilisez la permutation de boucles pour tenter d'améliorer les performances. Testez.
3. Le compilateur gcc peut aussi effectuer des permutations de boucles avec l'option `-floop-block`. Utilisez cette option (combinée avec `-O2`) pour tester si elle produit de meilleures performances par rapport à votre code optimisé au niveau source. Eventuellement, vous pouvez aussi combiner `-floop-block` avec l'option `-floop-interchange`.