Paul Fournel
Pierre Richaud

Master SITN
29/08/2014

Factorisation de Cholesky

I Introduction

La factorisation de Cholesky consiste, si on se donne une matrice A symétrique définie positive, à chercher une matrice L triangulaire tel que A soit égale au produit de L et de sa transposée t(L). L'algorithme de Cholesky permet principalement de faciliter la résolution de système linéaire.

Plus précisément, c'est la forme triangulaire des matrices qui permet de simplifier les équations. Par exemple : A*x=L*t(L)*x=b se résout en deux temps en posant y=t(L)*x et b=L*y. Pour ces deux équations la forme triangulaire simplifie grandement la résolution.
Une autre utilité de cet algorithme est de calculer le déterminant. En effet le déterminant d'une matrice triangulaire est égale au produit de ses coefficients diagonaux.

Le but de ce travail est d'implémenter de deux manières l'algorithme de Cholesky en langage C. Il s'agit d'implémenter l'algorithme d'une manière séquentielle et d'une manière parallèlle. Cet objectif est motiver par le fait que l'algorithme demande beaucoup de calculs différents et qu'il devient long pour des matrices de grande taille.

II Présentation de l'algorithme

1 Algorithme classique

Dans cette partie nous allons présenter le principe de l'algorithme classique. Soit A une matrice symétrique définie positive. Le but est de remplir à partir de A une matrice triangulaire supérieure L. Pour cela, on part de l'égalité A=L*t(L).
Partant de cette égalité, il est possible de déterminer une à une chacune des lignes de la matrice L comme combinaison des coefficients de la matrice A.
Cependant, pour une ligne i de la matrice L il faut d'abord calculer le coefficient diagonal, car les autres coefficients de la ligne i se calculent en fonction de celui-ci.

Voici une présentation rapide de l'algorithme :

Entrée : matrice A symétrique définie positive.

```
for 1<i < nombre de ligne de A
  for j>=i
    if i=j (calcul du coefficient diagonal L(i, i) en fonction des coeffcients de A.)
          for 1<k<i-1
                tmp=tmp+L(i,k)*L(i,k)
```

```
        end for
      L(i,j) = sqrt(A(i,j) - tmp)
    if i<j (calcul du coefficient diagonal L(i, j) en fonction des coeffcients de A et de L(i, i).)
      initialiser variable tmp ;
        for 1<k<j
            tmp=tmp+L(i,k)*L(j,k)
        end for
      L(i,j) = (A(i,j)-tmp) / L(i,i)
    end if
  end for
end for
```

Sortie : la matrice L triangulaire supérieure telle que A=L*t(L).

2 Algorithme parallèle

Dans cette partie nous présentons la version parallèle de l'algorithme. La seule chose qui a été parallélisée c'est le calcul du coefficient L(i, i) et L(i, j), c'est à dire la troisième boucle for :

```
initialiser variable tmp ;
        for 1<k<j
            tmp=tmp+L(i,k)*L(j,k)
        end for
      L(i,j) = (A(i,j)-tmp) / L(i,i)
```

Au lieu d'effectuer le calcul terme à terme cette boucle à été parallélisée de manière a effectuer simultanément chacun des produits L(i,k)*L(j,k) puis a en faire la somme une fois les calculs effectués.

III Résultats

1 Algorithme classique

L'algorithme dans le cas normal fonctionne pour toute matrice, quel que soit son rang. Néanmoins, on peut parfois observer un dépassement de capacité et ainsi obtenir la valeur *inf.* Cela est due à l'encodage d'un nombre trop grand codé sur un type double. Ce dépassement fausse le reste des calculs, c'est pour cela que l'on obtiendra *inf* dans toutes les autres cellules de la matrice, calculées après.

Une solution pour résoudre ce problème serait de changer le codage et d'opter pour un type qui admet des nombres plus grands. Néanmoins, il y aura toujours à un moment ou un autre un dépassement de capacité.

2 Algorithme parallèle

L'algorithme dans le cas parallèle fonctionne pour des matrices de petites tailles (n<10). Dès que la dimension augmente une erreur de calcul apparaît et se propage de proche en proche. Ainsi le

résultat obtenu est faux. On s'en rend compte, car on a C*t(C)!=A.

Ici aussi, le problème de dépassement de capacité peut apparaître. Les *inf* obtenus viennent perturber la suite de l'algorithme.

3 Comparaison des algorithmes

On test les deux algorithmes 10 fois sur des matrices de différentes tailles. Le tableau ci-dessous nous donne la moyenne des résultats. N représente le rang de la matrice. Les temps de calculs, pour chacun des algorithmes, sont donnés en millisecondes.

N	5	100	300	700	1400
Algo Normal	0	0	110	1300	10000
Algo Parallèle	0	200	2300	12400	60000

On remarque dans le cas présent que la parallélisation a un effet néfaste sur l'algorithme. Plusieurs facteurs peuvent expliquer ce phénomène.

Le premier facteur est que la parallélisation se fait dans la dernière boucle. Ceci a comme effet de créer et d'attendre la fin des threads à chaque itération. Le temps gagné par le calcul parallèle et absorbé par le temps de gestion des thread. C'est pourquoi la pénalisation (en temps) n'est pas linéaire par rapport à la valeur de N.

Le deuxième facteur et l'utilisation du mutex dans chacun des threads. Cette utilisation a pour conséquence de ralentir à chaque appel d'un thread tout l'algorithme. Il doit attendre que la valeur de la somme soit libérée.

Ces deux facteurs nous permettent de dire que l'algorithme de parallélisation utilisé n'est pas bon.

IV Conclusion

Il était difficile dans le cas de la factorisation de Cholesky de trouver une parallélisation efficace. Dans l'algorithme classique, celui présenté sur Wikipédia, il y a une forte dépendance entre les différentes étapes. Ainsi, nous n'avons pas trouvé un moyen de paralléliser une autre boucle que la troisième de l'ordre d'imbrication.

Une piste pour proposer un algorithme parallèle serait peut-être de trouver un moyen de faire la factorisation de Cholesky par bloc. Ainsi, l'algorithme initial serait utilisé tel quel, mais la matrice serait traitée en 4 fois séparément pour être reconstruite à la fin. Nous avons cherché en vain de ce côté là.