TP2:Multiplication matricielle de Fox

EL AFRIT Mohamed Amine HABASSI Seif Eddine

1 Introduction

Les trois méthodes parallèles de multiplication de matrices denses par blocs les plus populaires sont les algorithmes de Cannon, de Fox, et de Snyder.

Le présent TP implémente la méthode de multiplication de Fox.

Soient **A**, **B**, et **C** des matrices carrées constituées de n^2 blocs carrés de même taille, notés $a_{i,j}$, $b_{i,j}$, et $c_{i,j}$, avec $0 \leq i < n$ et $0 \leq j < n$. L'algorithme de Fox consiste à calculer et accumuler localement les termes de la somme :

$$c_{i,j} = a_{i,0} \times b_{0,j} + a_{i,1} \times b_{1,j} + ... + a_{i,n-1} \times b_{n-1,j}$$

, en fournissant au processus responsable du bloc $c_{i,j}$ les données nécessaires. Pour cela, on effectue n étapes, telles qu'à la kième étape $(0 \leq k < n)$:

1. pour chaque ligne, on diffuse à tous les processus d'une ligne le bloc de **A** situé k positions à droite du bloc diagonal ;
2. pour chaque bloc, on accumule dans $c_{i,j}$ le produit du bloc de **A** reÃ§u par diffusion et du bloc de **B** courant ;
3. pour chaque colonne, on permute circulairement vers le haut les blocs de **B**.

2 La répartition entre les processus

On suppose que seulement un processus(le processu 0) a accès au contenu complet des matrices A et B. C'est à lui donc de se charger de la distribution des données.

La matrice de départ est partitionnée dans des fichiers extrieur, en un nombre carré de blocs. Le programme se charge de lire les différents blocs. Chaque bloc est envoyé à un processus.

3 L'algorithme de Fox :

L'algorithme se décompose en trois étapes qui sont itérées n fois (n est le nombre de bloc par dimension).
A la k^{me} itération :

3.1 Diffusion horizontale dans A :

Le processus qui contient le bloc sur la diagonale avec un décalage de k, le diffuse sur toute la ligne. Cette diffusion est faite en boucle avec MPI_Send dans les communicateurs MPI_COMM_WORLD.

```
//envoyer a chaque processeur le bloc des matrices A et B qui lui correspond
MPI_Send((void*)A,9,MPI_INT,k,1, MPI_COMM_WORLD);
MPI_Send((void*)B,9,MPI_INT,k,1, MPI_COMM_WORLD);
```

3.2 Décalage cyclique verticale vers le haut dans B :

Ce décalage est fait par les processus fils du processus parent 0.

```
      if (myid!=3 && myid!=6) {
MPI_Send((void*)B_rcv,9,MPI_INT,(myid-N+(N*N))%(N*N),1, MPI_COMM_WORLD);
MPI_Recv(B_rcv,9, MPI_INT,(myid+N)%(N*N) , 1, MPI_COMM_WORLD, &status) ;
      }else if (myid==3){
MPI_Send((void*)B_rcv,9,MPI_INT,(N*N),1, MPI_COMM_WORLD);
MPI_Recv(B_rcv,9, MPI_INT,(myid+N)%(N*N) , 1, MPI_COMM_WORLD, &status) ;
      }
      else if (myid==6){
MPI_Send((void*)B_rcv,9,MPI_INT,(myid-N)%(N*N),1, MPI_COMM_WORLD);
MPI_Recv(B_rcv,9, MPI_INT,(N*N) , 1, MPI_COMM_WORLD, &status) ;
      }
```

3.3 Multiplication matricielle

3.3.1 Sans $BLAS$

Le calcul sans $BLAS$ de la multiplication matricielle est implémenté par une simple fonction qui parcours les deux matrices et fait des multiplications séquentielles. La complexité des multiplication bloc-bloc reste en $O(taille\_block^3)$.

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

3.3.2 Multiplication avec BLAS

Pour optimiser les multiplactions matricielles locales dans chaque processus, on utilise la routine BLAS : *cblas_dgemm* Cette routine permet d'effectuer le calcule suivant : $C = \alpha * AB + \beta * C$
Pour accumuler le rsultat des multiplications dans C, on utilise : $\alpha = 1$ et $\beta = 1$.

```
cblas_dgemm(CblasRowMajor,CblasNoTrans,CblasNoTrans,N,N,N,1,
 (double*)B_rcv,N,(double*)A_rcv,N,1,(double*)C,N)
```

4 Mini-Manuel utilisation

Un makefile est fournit. La compilation gnre l'excutable ***fox***.
Le programme prend les paramtres suivants :

```
mpirun -np 10 fox [-r] [-b]
```

- Il faut exécuter avec 10 processeur pour notre exemple
- -r pour afficher les résultats.
- -b pour activer le calcul avec les blas.

5 Comparaison des performances

5.1 Tests

Les tests suivants ont été effectués avec 10 processus sur une machine double coeur double processeurs en utilisant MPICH.

5.1.1 Sans BLAS

Processus parent	Δt 1^{re} partie	Δt Total
0	0.002961 s	0.140359 s

Processus	Δt 1^{re} partie	Δt 1^{re} permutation	Δt 2^{me}	Δt Total permutation
1	0.000160 s	0.024707 s	0.000406 s	0.140169 s
2	0.000495 s	0.013692 s	0.000026 s	0.140427 s
3	0.000336 s	0.036156 s	0.000489 s	0.139269 s
4	0.000712 s	0.000034 s	0.000028 s	0.139644 s
5	0.001179 s	0.136152 s	0.000501 s	0.139781 s
6	0.001967 s	0.124242 s	0.000020 s	0.140194 s
7	0.001104 s	0.024242 s	0.000012 s	0.139064 s
8	0.001672 s	0.135837 s	0.000147 s	0.139258 s
9	0.002102 s	0.136254 s	0.000032 s	0.139686 s

5.1.2 Avec BLAS

Processus parent	Δt 1^{re} partie	Δt Total
0	0 : 0.002688 s	0.139914 s

Processus	Δt 1^{re} partie	Δt 1^{re} permutation	Δt 2^{me}	Δt Total permutation
1	0.000544 s	0.018777 s	0.000018 s	0.139567 s
2	0.001088 s	0.018457 s	0.000311 s	0.139908 s
3	0.000735 s	0.135667 s	0.000017 s	0.139386 s
4	0.001095 s	0.116232 s	0.000477 s	0.139456 s
5	0.001422 s	0.018678 s	0.000024 s	0.139617 s
6	0.001794 s	0.135563 s	0.000215 s	0.139664 s
7	0.001574 s	0.116133 s	0.000074 s	0.139370 s
8	0.001781 s	0.018624 s	0.000025 s	0.139105 s
9	0.001659 s	0.135738 s	0.000101 s	0.138880 s

$\Rightarrow$ La comparaison des temps totals montre que le calcul avec $BLAS$ est plus rapide.

6 Conclusion

L'utilisation des $BLAS$ donne des rsultats importantes quand on manipule des blocs de grandes tailles. Le prsent TP tait une occasion pour découvrir les optimisations offertes par cette bibliothque. Le code fournis présente une solution qui reste sûrement à améliorer, les communications entre les processus et la manipulation des matrices restent des points à optimiser.