TP M54 : Produit de matrices par bloc en MPI.

Laurent Desbat

Octobre 2002

L'objectif de ce TP est de mettre en oeuvre un produit de matrice par bloc de taille b, $C = AB$, où C, A et B sont des matrices de taille n. Soit $C_{I,J}$ le bloc (I, J) de taille $b \times b$ de la matrice C pour $I = 0, \ldots, n/b-1$ pour $J = 0, \ldots, n/b-1$ (idem pour A et B) nous avons

```
pour I=0,..., n/b-1
   pour J=0,..., n/b-1
      InitAZero(C_{I,J},b) /* C_{I,J}=0 */
      pour K=0,..., n/b-1
          MultAndAdd(C_{I,J},A_{I,K},B_{K,J},b)
             /* C_{I,J}=  C_{I,J}+ A_{I,K}B_{K,J} */
      finpour
   finpour
finpour
```

1. Dans une première partie nous distribuons les matrices A, B et C sur un réseau de p processus organisés suivant une topologie de tore 2D. Nous supposons que $\sqrt{p}$ est pair (on prendra en pratique $p = 4$ soit $\sqrt{p} = 2$). La distribution des données conduit à $b = n/\sqrt{p}$. Construire la topologie de tore de processus. Faire le produit de matrices par blocs en faisant circuler les blocs $A_{I,K}$ et $B_{K,J}$ sur le tore. On suppose que le processus de numéro I, J sur le tore calcul le bloc $C_{I,J}$:

```
# include<stdio.h>
# include<mpi.h>

int main(int argc, char *argv[])
     /*
```

```
 Produit de Matrices en  MPI C
mpicc -c mpifile.c
mpicc -o exec  mpifile.c autrefile.o
mpirun -np 4 mpifile
     */
{
  int rank, size;
  int sqrtsize;
  MPI_Status status;
  int root;
  int moiI,moiJ; /* numerotation sur le tore du processus courant */
  int nord,sud,est,ouest; /* numero des 4 voisins sur le tore */
  int nordI,sudI,estI,ouestI;
  int nordJ,sudJ,estJ,ouestJ;
  int tag;
  int rankdest;
  int suivant, precedent; /* suivant,precedent sur l'anneau */

  int jeton; /* jeton a faire circuler sur un anneau */
  int i;

  /* initialization */
  MPI_Init(&argc,&argv);

  MPI_Comm_rank(MPI_COMM_WORLD,&rank);
  MPI_Comm_size(MPI_COMM_WORLD,&size);

  root = 0;

  /* la taille doit être un carré multiple de 4 */
  sqrtsize = sqrt((double) size);
  if(rank==root){
    printf(" sqrtsize = %d different de sqrt(size) =%lf \n",
   sqrtsize,sqrt((double)size));
  }
  if((sqrtsize%2)!=0){
    if(rank==root)
      printf("error sqrtsize = %d non pair \n",sqrtsize);

    MPI_Finalize();
  }
```

```
if((sqrtsize*sqrtsize)!=size){
  if(rank==root){
    printf("error sqrtsize*sqrtsize = %d different de size =%d \n",sqrtsize,size);
  }
  MPI_Finalize();
}

/* Construction du tore */
moiI=rank/sqrtsize;
moiJ=rank %sqrtsize;
nordI= (moiI-1) %sqrtsize;
nordJ= moiJ;
nord=nordI*sqrtsize+nordJ;
sudI= (moiI+1) %sqrtsize;
sudJ= moiJ;
sud=sudI*sqrtsize+sudJ;
estI= moiI;
estJ= (moiJ+1) %sqrtsize;
est=estI*sqrtsize+estJ;
ouestI= moiI;
ouestJ= (moiJ-1) %sqrtsize;
ouest=ouestI*sqrtsize+ouestJ;
/* Affichage de la topologie */
suivant=(rank+1)%size; /* on suit l'ordre de l'anneau pour l'affichage */
precedent=(rank-1)%size;
tag=suivant;
if(rank==root){
  printf(" je suis le processeur %d, parmi %d processeurs \n",rank,size);
  printf(" mes quatre voisins sont : \n");
  printf(" ouest=%d ; est=%d ; nord=%d ; sud =%d  \n",ouest,est,nord,sud);
  MPI_Send(&jeton,1,MPI_INT,suivant,tag,MPI_COMM_WORLD);
  MPI_Recv(&jeton,1,MPI_INT,precedent,rank,MPI_COMM_WORLD,&status);
}
else{
  MPI_Recv(&jeton,1,MPI_INT,precedent,rank,MPI_COMM_WORLD,&status);
  printf(" je suis le processeur %d, parmi %d processeurs \n",rank,size);
  printf(" mes quatre voisins sont : \n");
  printf(" ouest=%d ; est=%d ; nord=%d ; sud =%d  \n",ouest,est,nord,sud);
  MPI_Send(&jeton,1,MPI_INT,suivant,tag,MPI_COMM_WORLD);
}
```

```
  MPI_Finalize();
}
```