BLOCKING FUNCTIONS (MAT ADDITION)

### SOURCE CODE:

#include <mpi.h> #include <stdio.h> #include <time.h> #include <stdlib.h>

int main(int argc, char \*argv[])

{

int rank, size; MPI\_Init(&argc, &argv);

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank); MPI\_Comm\_size(MPI\_COMM\_WORLD, &size); if(rank==0){

double startTime,endTime; int colA,rowA,colB,rowB; int partcol,fraccol;

int countrank;

float \*\*matrixA,\*\*matrixB,\*\*matrixres; int i,j;

rowA = colA = rowB = colB = 500 ; partcol=(int)colA/size; fraccol=colA-(partcol\*size);

matrixA = (float\*\*)malloc(sizeof(float\*)\*rowA); for (i = 0; i < rowA; i++)

matrixA[i]=(float\*)malloc(sizeof(float)\*colA); matrixB = (float\*\*)malloc(sizeof(float\*)\*rowB); for (i = 0; i < rowB; i++)

matrixB[i]=(float\*)malloc(sizeof(float)\*colB); matrixres = (float\*\*)malloc(sizeof(float\*)\*rowA); for (i = 0; i < rowA; i++)

matrixres[i]=(float\*)malloc(sizeof(float)\*colA); for(i = 0 ; i < rowA ; i++)

for(j = 0; j < colA; j++) matrixres[i][j]=0;

for (i = 0; i < rowA; i++)

for (j = 0; j < colA; j++)

matrixA[i][j] = (float)(rand()%(rowA\*rowA));

for (i = 0; i < rowB; i++)

for (j = 0; j < colB; j++)

matrixB[i][j] = (float)(rand()%rowB\*rowB); startTime = MPI\_Wtime();

for(i = 0 ; i < rowA ; i++) for(j = 0; j < partcol; j++)

matrixres[i][j]=matrixA[i][j]+matrixB[i][j]; printf("Rank 0 addition completed\n");

if (size!=1)

{

for (countrank = 1; countrank < size; countrank++)

{

MPI\_Send(&rowA,1,MPI\_INT,countrank,1,MPI\_COMM\_WORLD); MPI\_Send(&partcol,1,MPI\_INT,countrank,2,MPI\_COMM\_WORLD);

}

for (countrank = 1; countrank < size; countrank++) for(i = 0 ; i < rowA ; i++)

for(j = partcol\*countrank; j < partcol\*(countrank+1); j++){

MPI\_Send(&matrixA[i][j],1,MPI\_FLOAT,countrank,3,MPI\_COMM\_WORLD); MPI\_Send(&matrixB[i][j],1,MPI\_FLOAT,countrank,4,MPI\_COMM\_WORLD);

}

printf("Rank 0 send matrix completed\n"); if(fraccol!=0){

for (i = 0; i < rowA; i++)

for (j = partcol\*(size); j < (partcol\*(size))+fraccol ; j++) matrixres[i][j]=matrixA[i][j]+matrixB[i][j];

}

for (countrank = 1; countrank < size; countrank++) for(i = 0 ; i < rowA ; i++)

for(j = partcol\*countrank; j < partcol\*(countrank+1); j++){

MPI\_Recv(&matrixres[i][j],1,MPI\_FLOAT,countrank,5,MPI\_COMM\_WORLD,MPI\_STATUS\_IGNORE

);

}

}

endTime = MPI\_Wtime();

printf("Matrix Addition Completed\n");

/\*for(i=0;i<rowA;i++){ for(j=0;j<colA;j++)

printf("%.1f ",matrixres[i][j]); printf("\n");

}\*/

printf("Timings : %f Sec\n", endTime - startTime);

}

else{

int i,j;

int row,partcol;

MPI\_Recv(&row,1,MPI\_INT,0,1,MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE); MPI\_Recv(&partcol,1,MPI\_INT,0,2,MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

printf("recv %d %d via rank %d\n",row,partcol,rank);

float \*\*matrixA = (float\*\*)malloc(sizeof(float\*)\*row); for (i = 0; i < row; i++)

matrixA[i]=(float\*)malloc(sizeof(float)\*partcol);

float \*\*matrixB = (float\*\*)malloc(sizeof(float\*)\*row); for (i = 0; i < row; i++)

matrixB[i]=(float\*)malloc(sizeof(float)\*partcol);

float \*\*matrixres = (float\*\*)malloc(sizeof(float\*)\*row); for (i = 0; i < row; i++)

matrixres[i]=(float\*)malloc(sizeof(float)\*partcol); printf("alloc matrix from rank %d completed\n",rank);

for(i = 0 ; i < row ; i++){

for(j = 0; j < partcol; j++){ MPI\_Recv(&matrixA[i][j],1,MPI\_FLOAT,0,3,MPI\_COMM\_WORLD,MPI\_STATUS\_IGNORE);

MPI\_Recv(&matrixB[i][j],1,MPI\_FLOAT,0,4,MPI\_COMM\_WORLD,MPI\_STATUS\_IGNORE);

matrixres[i][j]=matrixA[i][j]+matrixB[i][j];

}

}

printf("recv matrix from rank %d completed\n",rank);

for(i = 0 ; i < row ; i++){

for(j = 0; j < partcol; j++){ MPI\_Send(&matrixres[i][j],1,MPI\_FLOAT,0,5,MPI\_COMM\_WORLD);

}

}

printf("Rank %d send back completed\n",rank);

}

MPI\_Finalize(); return 0;

}