ТЕХНИЧЕСКИ УНИВЕРСИТЕТ - СОФИЯ
ФИЛИАЛ ПЛОВДИВ

ФАКУЛТЕТ ПО ЕЛЕКТРОНИКА И АВТОМАТИКА

КУРСОВА ЗАДАЧА

Тема: *Умножаване на матрици реализирано с библиотеката „openmpi“*

дисциплина: *Паралелно програмиране*
изготвил: *Богомил Василев Василев*
специалност: *КСТ*
група: *42б*
фак. №: *357840* *одобрил: ....*
/гл. ас. д-р Мария Маринова/

Умножаването на матрици е реализирано и изпробвано под Linux, използвайки open source библиотеката „openmpi“ и компилатора „GCC“.

Компилирането се извършва лесно в терминал по следния начин:

```
mpicc mpi-mm.c -o mpi-mm
```

За по-сигурна компилация и бързодействие може да се използва:

```
mpicc mpi-mm.c -O2 -Wall -o mpi-mm
```

След компилацията, приложението се стартира само с 1 параметър – броя на колоните и редовете на матрицата. Например:

```
./mpi-mm 9
```

Вариантите за изпълнение са два. Единият е автоматичен, който запълва матриците A и B с „9”, а другия е ръчен и запълва матриците с посочени от клавиатурата стойности. При въведен параметър „9” и автоматичен метод на попълване, резултата би следвал да бъде:

```
729.000000 729.000000 729.000000 729.000000 729.000000 729.000000 729.000000 729.000000 729.000000
729.000000 729.000000 729.000000 729.000000 729.000000 729.000000 729.000000 729.000000 729.000000
729.000000 729.000000 729.000000 729.000000 729.000000 729.000000 729.000000 729.000000 729.000000
729.000000 729.000000 729.000000 729.000000 729.000000 729.000000 729.000000 729.000000 729.000000
729.000000 729.000000 729.000000 729.000000 729.000000 729.000000 729.000000 729.000000 729.000000
729.000000 729.000000 729.000000 729.000000 729.000000 729.000000 729.000000 729.000000 729.000000
729.000000 729.000000 729.000000 729.000000 729.000000 729.000000 729.000000 729.000000 729.000000
729.000000 729.000000 729.000000 729.000000 729.000000 729.000000 729.000000 729.000000 729.000000
729.000000 729.000000 729.000000 729.000000 729.000000 729.000000 729.000000 729.000000 729.000000
```

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

#define TAG 13

int main(int argc, char *argv[])
{
    double **A, **B, **C, *tmp;
    double startTime, endTime;
    int numElements, offset, stripSize, myrank, numnodes, N, i, j, k, input;

    MPI_Init(&argc, &argv);

    MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
    MPI_Comm_size(MPI_COMM_WORLD, &numnodes);

    N = atoi(argv[1]);

    // allocate A, B, and C --- note that you want these to be
    // contiguously allocated.  Workers need less memory allocated.

    if (myrank == 0) {
        tmp = (double *) malloc (sizeof(double ) * N * N);
        A = (double **) malloc (sizeof(double *) * N);
        for (i = 0; i < N; i++)
            A[i] = &tmp[i * N];
    } else {
        tmp = (double *) malloc (sizeof(double ) * N * N / numnodes);
        A = (double **) malloc (sizeof(double *) * N / numnodes);
        for (i = 0; i < N / numnodes; i++)
            A[i] = &tmp[i * N];
    }


    tmp = (double *) malloc (sizeof(double ) * N * N);
    B = (double **) malloc (sizeof(double *) * N);
    for (i = 0; i < N; i++)
        B[i] = &tmp[i * N];


    if (myrank == 0) {
        tmp = (double *) malloc (sizeof(double ) * N * N);
        C = (double **) malloc (sizeof(double *) * N);
        for (i = 0; i < N; i++)
            C[i] = &tmp[i * N];
    } else {
        tmp = (double *) malloc (sizeof(double ) * N * N / numnodes);
        C = (double **) malloc (sizeof(double *) * N / numnodes);
        for (i = 0; i < N / numnodes; i++)
            C[i] = &tmp[i * N];
    }

    if (myrank == 0) {
       printf( "Fill methods:\n"
               "1. Automatic.\n"
               "2. Manual.\n\n"
               "\tINPUT: ");
       scanf("%d", &input);

       if (input == 1) {
           // initialize A and B
           for (i=0; i<N; i++) {
```

```
            for (j=0; j<N; j++) {
                A[i][j] = N;
                B[i][j] = N;
            }
         }
      } else if (input == 2) {
         // initialize A and B
         for (i=0; i<N; i++) {
            for (j=0; j<N; j++) {
                printf("A[%d][%d] = ", i + 1, j + 1);
                scanf("%lf", &A[i][j]);
            }
         }

         for (i=0; i<N; i++) {
            for (j=0; j<N; j++) {
                printf("B[%d][%d] = ", i + 1, j + 1);
                scanf("%lf", &B[i][j]);
            }
         }
      } else {
         fprintf(stderr, "Incorrect input!\n");
         exit(1);
      }
    }

    // start timer
    if (myrank == 0) {
        startTime = MPI_Wtime();
    }

    stripSize = N/numnodes;

    // send each node its piece of A -- note could be done via MPI_Scatter
    if (myrank == 0) {
        offset = stripSize;
        numElements = stripSize * N;
        for (i=1; i<numnodes; i++) {
            MPI_Send(A[offset], numElements, MPI_DOUBLE, i, TAG, MPI_COMM_WORLD);
            offset += stripSize;
        }
    } else { // receive my part of A
        MPI_Recv(A[0], stripSize * N, MPI_DOUBLE, 0, TAG, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);
    }

    // everyone gets B
    MPI_Bcast(B[0], N*N, MPI_DOUBLE, 0, MPI_COMM_WORLD);

    // Let each process initialize C to zero
    for (i=0; i<stripSize; i++) {
        for (j=0; j<N; j++) {
            C[i][j] = 0.0;
        }
    }

    // do the work
    for (i=0; i<stripSize; i++) {
        for (j=0; j<N; j++) {
            for (k=0; k<N; k++) {
                C[i][j] += A[i][k] * B[k][j];
            }
        }
```

```
    }

    // master receives from workers  -- note could be done via MPI_Gather
    if (myrank == 0) {
        offset = stripSize;
        numElements = stripSize * N;
        for (i=1; i<numnodes; i++) {
            MPI_Recv(C[offset], numElements, MPI_DOUBLE, i, TAG, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);
            offset += stripSize;
        }
    } else { // send my contribution to C
        MPI_Send(C[0], stripSize * N, MPI_DOUBLE, 0, TAG, MPI_COMM_WORLD);
    }

    // stop timer
    if (myrank == 0) {
        endTime = MPI_Wtime();
        printf("Time is %f\n", endTime-startTime);
    }

    // print out matrix here, if I'm the master
    if (myrank == 0 && N < 10) {
        for (i=0; i<N; i++) {
            for (j=0; j<N; j++) {
                printf("%f ", C[i][j]);
            }
            printf("\n");
        }
    }

    MPI_Finalize();
    return 0;
}
```