//tính số pi

#include<stdio.h>

#include<conio.h>

#include<math.h>

double pi(double n)

{

double tong = 0;

double x;

for (int i = 0; i <= n; i++)

{

x = (i - 1/2) / n;

tong = tong + x;

}

return 1/n \* tong;

}

void main()

{

double a;

printf("Nhập n:");

scanf\_s("%d",a);

printf("%.16lf\n", pi(a));

}

**//ma tran nhan ma tran**

#include<stdio.h>

#include<conio.h>

#define NRA 62

#define NCA 15

#define NCB 7

void main()

{

int a[100][100], b[100][100], matrantich[100][100];

int i;

int j;

int k;

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

{

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

{

a[i][j] = i + j;

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

{

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

{

b[i][j] = i \* j;

}

}

}

}

printf("\nTich cua hai ma tran la:\n");

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

{

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

{

matrantich[i][j] = 0;

for (k = 0; k<NCB; k++)

{

matrantich[i][j] += a[i][k] \* b[k][j];

}

}

}

// hien thi ket qua

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

{

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

{

printf("%d\t", matrantich[i][j]);

}

printf("\n");

}

}

//song song

#include <stdio.h>

#include <math.h>

#include <mpi.h>

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

{

int n, myid, numprocs, i;

double PI25DT = 3.141592653589793238462643;

double mypi, pi, h, sum, x;

MPI\_Init(&argc, &argv);

MPI\_Comm\_size(MPI\_COMM\_WORLD, &numprocs);

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &myid);

while (1)

{

if (myid == 0)

{

printf("Enter the number of intervals: (0 quits)");

scanf\_s("%d",&n);

}

MPI\_Bcast(&n, 1, MPI\_INT, 0, MPI\_COMM\_WORLD);

if (n == 0)

break;

else

{

h = 1.0 / (double)n;

sum = 0.0;

for (i = myid + 1; i <= n; i += numprocs)

{

x = h \* ((double)i - 0.5);

sum += (4.0 / (1.0 + x\*x));

}

mypi = h \* sum;

MPI\_Reduce(&mypi, &pi, 1, MPI\_DOUBLE, MPI\_SUM, 0, MPI\_COMM\_WORLD);

if (myid == 0)

printf("pi is approximately %.16f, Error is %.16f\n", pi, fabs(pi - PI25DT));

}

}

MPI\_Finalize();

return 0;

}

//ma tran nhan ma tran

#include <stdio.h>

#include <stdlib.h>

#include <mpi.h>

#define NRA 62

#define NCA 15

#define NCB 7

#define MASTER 0

#define FROM\_MASTER 1

#define FROM\_WORKER 2

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

{

int numtasks, taskid, numworkers, source, dest, mtype, rows, averow, extra, offset, i, j, k, rc;

double a[NRA][NCA], b[NCA][NCB], c[NRA][NCB];

MPI\_Status status;

MPI\_Init(&argc, &argv);

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &taskid);

MPI\_Comm\_size(MPI\_COMM\_WORLD, &numtasks);

if (numtasks < 2)

{

printf("Need at l e a s t two MPI ta s ks . Qui tt in g . . . \ n");

MPI\_Abort(MPI\_COMM\_WORLD,&rc);

exit(1);

}

numworkers = numtasks - 1;

//∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ master task ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗

if (taskid == MASTER)

{

printf("mpi\_mm has s t a r t e d with %d tas k s . \ n", numtasks);

printf(" I n i t i a l i z i n g a rr ays . . . \ n");

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

{

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

{

a[i][j] = i + j;

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

{

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

{

b[i][j] = i \* j;

}

}

}

}

// Send matrix data to the worker tasks

averow = NRA / numworkers;

extra = NRA%numworkers;

offset = 0; mtype = FROM\_MASTER;

for (dest = 1; dest <= numworkers; dest++)

{

rows = (dest <= extra) ? averow + 1 : averow;

printf(" Sending %d rows to task %d o f f s e t=%d\n", rows, dest, offset);

MPI\_Send(&offset, 1, MPI\_INT, dest, mtype, MPI\_COMM\_WORLD);

MPI\_Send(&rows, 1, MPI\_INT, dest, mtype, MPI\_COMM\_WORLD);

MPI\_Send(&a[offset][0], rows \* NCA, MPI\_DOUBLE, dest, mtype, MPI\_COMM\_WORLD);

MPI\_Send(&b, NCA \* NCB, MPI\_DOUBLE, dest, mtype, MPI\_COMM\_WORLD);

offset = offset + rows;

//Receive r e s u l t s from worker t a sks

mtype = FROM\_WORKER;

for (i = 1; i <= numworkers; i++)

{

source = i;

MPI\_Recv(&offset, 1, MPI\_INT, source, mtype, MPI\_COMM\_WORLD, &status);

MPI\_Recv(&rows, 1, MPI\_INT, source, mtype, MPI\_COMM\_WORLD, &status);

MPI\_Recv(&c[offset][0], rows \* NCB, MPI\_DOUBLE, source, mtype, MPI\_COMM\_WORLD, &status);

printf(" Received r e s u l t s from task %d\n", source);

}

//Print results

printf(" ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ \ n");

printf(" Res ult Matrix : \ n");

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

{

printf("\n");

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

printf("%6.2 f ", c[i][j]);

}

printf("\n∗∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ \ n");

printf("Done . \ n");

}

// ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ worker task ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗ ∗

if (taskid > MASTER)

{

// Receive matrix data from master t ask

mtype = FROM\_MASTER;

MPI\_Recv(&offset, 1, MPI\_INT, MASTER, mtype, MPI\_COMM\_WORLD, &status);

MPI\_Recv(&rows, 1, MPI\_INT, MASTER, mtype, MPI\_COMM\_WORLD, &status);

MPI\_Recv(&a, rows \* NCA, MPI\_DOUBLE, MASTER, mtype, MPI\_COMM\_WORLD, &status);

MPI\_Recv(&b, NCA \*NCB, MPI\_DOUBLE, MASTER, mtype, MPI\_COMM\_WORLD, &status);

// Do matrix mu ltiply

for (k = 0; k < NCB; k++)

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

{

c[i][k] = 0.0;

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

c[i][k] = c[i][k] + a[i][j] \* b[j][k];

}

//Send results back to master t ask

mtype = FROM\_WORKER;

MPI\_Send(&offset, 1, MPI\_INT, MASTER, mtype, MPI\_COMM\_WORLD);

MPI\_Send(&rows, 1, MPI\_INT, MASTER, mtype, MPI\_COMM\_WORLD);

MPI\_Send(&c, rows \* NCB, MPI\_DOUBLE, MASTER, mtype, MPI\_COMM\_WORLD);

}

MPI\_Finalize();

}

}