Національний технічний університет України

«Київський Політехнічний Інститут»

Факультет інформатики і обчислювальної техніки

Кафедра обчислювальної техніки

**Лабораторна робота №8**

**З предмету «Паралельні та розподілені обчислення»**

**MPI**

Виконав:

Студент  
IІІ курсу ФІОТ  
групи ІО-12  
Бута С. О.

Залікова книжка №1205

Київ-2014

**Техническое задание**

1. Структура ПКС с ЛП: 
2. Задача: .
3. Язык программирования: C++.
4. Средства взаимодействия задач: MPI.

**Выполнение работы**

1. Разработка параллельного математического алгоритма.
2. MAH = MB\*(MC\*MOH) – l\*ME; OP: l, MB, MC;
3. Разработка алгоритмов процессов.

**Задача Т1:**

1. Ввод l, MB, MC, MO, ME
2. Передать l, MB, MC, MOH, MEH в T2-T4; l, MB, MC, MO3Н, ME3Н в Т5
3. Счёт
   1. MAH = MB\*(MC\*MOH) – l\*MEH
4. Принять MAH из T2-T4; MA3H из Т5
5. Вывод MA

**Задача Тj (j = 2..4):**

1. Принять l, MB, MC, MOH, MEH из T1
2. Счёт
   1. MAH = MB\*(MC\*MOH) – l\*MEH
3. Передать MAH в T1

**Задача Т5:**

1. Принять l, MB, MC, MO3Н, ME3Н из T1
2. Передать l, MB, MC, MO2Н, ME2Н в T6
3. Счёт
   1. MAH = MB\*(MC\*MOH) – l\*MEH
4. Принять MA2H из T6
5. Передать MA3Н в T1

**Задача Т6:**

1. Принять l, MB, MC, MO2Н, ME2Н из Т5
2. Передать l, MB, MC, MOH, MEH в Т7
3. Счёт
   1. MAH = MB\*(MC\*MOH) – l\*MEH
4. Принять MAH из T7
5. Передать MA2H в T5

**Задача Т7:**

1. Принять l, MB, MC, MOH, MEH из Т6
2. Счёт
   1. MAH = MB\*(MC\*MOH) – l\*MEH
3. Передать MAH в T6

**Разработка программы.**

**Листинг**:

**Файл Runner.cpp**:

//--------------------------------

//Лабораторная работа №8

//Задание: MA = MB\*(MC\*MO) – l\*ME

//Бута С.О.

//Дата: 06.06.2014

//--------------------------------

#include <mpi.h>

#include <iostream>

#include "Matrix.h"

#include <time.h>

using namespace std;

int N = 14;

int P = 7;

int H = N/P;

int FILLER = 1;

MPI\_Group world\_group, star\_group;

MPI\_Comm star\_comm;

int star\_ranks[5] = {0, 1, 2, 3, 4};

int counts[] = {H\*N, H\*N, H\*N, H\*N, 3\*H\*N};

int displs[] = {0, H\*N, 2\*H\*N, 3\*H\*N, 4\*H\*N};

int displs1[] = {0, H\*N, 2\*H\*N, 3\*H\*N, 4\*H\*N};

int recvcounts[] = {N\*N, N\*N, N\*N, N\*N, N\*N};

void calculate(Matrix& MA, Matrix& MB, Matrix& MC, Matrix& MO, Matrix& ME, int l){

Matrix res(N, H);

MC.mult(res, MO);

MB.mult(MA, res);

int\* maIt = (int\*)MA.getPtrToArray();

int\* meIt = (int\*)ME.getPtrToArray();

for(int i = 0; i < N\*H; i++){

\*maIt -= l\*(\*meIt);

maIt++;

meIt++;

}

}

void\* shiftPtr(void\* inp, int s){

return (void\*)((int\*)inp + s\*H\*N);

}

int size(Matrix& MA){

return MA.getColumns()\*MA.getRows();

}

void matrOut(Matrix& M){

for(int i = 0; i < M.getRows(); i++){

for(int j = 0; j < M.getColumns(); j++){

cout << M[i][j] << " ";

}

cout << endl;

}

}

void sendLeftMAh(int rank, Matrix& MAh, int Hs){

Matrix MAh2(N, Hs);

if(rank != 6){

MPI\_Recv(MAh2.getPtrToArray(), size(MAh2), MPI\_INT, rank+1, 31, MPI\_COMM\_WORLD, MPI\_STATUSES\_IGNORE);

MAh.merge(MAh2);

}

MPI\_Send(MAh.getPtrToArray(), size(MAh), MPI\_INT, rank-1, 31, MPI\_COMM\_WORLD);

}

void threadFunc0(MPI\_Comm& star){

Matrix MB(N), MC(N), MO(N), ME(N);

int l;

MB.input();

MC.input();

MO.input();

ME.input();

l = FILLER;

MPI\_Bcast(MB.getPtrToArray(), N\*N, MPI\_INT, 0, star);

MPI\_Bcast(MC.getPtrToArray(), N\*N, MPI\_INT, 0, star);

MPI\_Bcast(&l, 1, MPI\_INT, 0, star);

int rank;

MPI\_Comm\_rank(star, &rank);

int recvcount = counts[rank];

int \*recvbuf = new int[recvcount];

MPI\_Scatterv(MO.getPtrToArray(), counts, displs, MPI\_INT, recvbuf,

recvcount, MPI\_INT, 0, star);

MPI\_Scatterv(ME.getPtrToArray(), counts, displs, MPI\_INT, recvbuf,

recvcount, MPI\_INT, 0, star);

MO.delExcessive(0, H);

ME.delExcessive(0, H);

Matrix MAh(N, H);

calculate(MAh, MB, MC, MO, ME, l);

Matrix MA(N, N);

MPI\_Gatherv(MAh.getPtrToArray(), counts[0], MPI\_INT, MA.getPtrToArray(),

recvcounts, displs1, MPI\_INT, 0, star);

if(N < 15){

cout << "Matrix MA:\n";

matrOut(MA);

}

}

void threadFunc13(MPI\_Comm& star){

Matrix MB(N), MC(N), MO(N, H), ME(N, H);

int l;

MPI\_Bcast(MB.getPtrToArray(), N\*N, MPI\_INT, 0, star);

MPI\_Bcast(MC.getPtrToArray(), N\*N, MPI\_INT, 0, star);

MPI\_Bcast(&l, 1, MPI\_INT, 0, star);

int rank;

MPI\_Comm\_rank(star, &rank);

int recvcount = counts[rank];

MPI\_Scatterv(MO.getPtrToArray(), counts, displs, MPI\_INT, MO.getPtrToArray(),

recvcount, MPI\_INT, 0, star);

MPI\_Scatterv(ME.getPtrToArray(), counts, displs, MPI\_INT, ME.getPtrToArray(),

recvcount, MPI\_INT, 0, star);

Matrix MA(N, H);

calculate(MA, MB, MC, MO, ME, l);

MPI\_Gatherv(MA.getPtrToArray(), counts[rank], MPI\_INT, MA.getPtrToArray(),

recvcounts, displs1, MPI\_INT, 0, star);

}

void threadFunc4(MPI\_Comm& star){

Matrix MB(N), MC(N), MO(N, 3\*H), ME(N, 3\*H);

int l;

MPI\_Bcast(MB.getPtrToArray(), N\*N, MPI\_INT, 0, star);

MPI\_Bcast(MC.getPtrToArray(), N\*N, MPI\_INT, 0, star);

MPI\_Bcast(&l, 1, MPI\_INT, 0, star);

int rank;

MPI\_Comm\_rank(star, &rank);

int recvcount = counts[rank];

MPI\_Scatterv(MO.getPtrToArray(), counts, displs, MPI\_INT, MO.getPtrToArray(),

recvcount, MPI\_INT, 0, star);

MPI\_Scatterv(ME.getPtrToArray(), counts, displs, MPI\_INT, ME.getPtrToArray(),

recvcount, MPI\_INT, 0, star);

MPI\_Send(MB.getPtrToArray(), N\*N, MPI\_INT, 5, 0, MPI\_COMM\_WORLD);

MPI\_Send(MC.getPtrToArray(), N\*N, MPI\_INT, 5, 1, MPI\_COMM\_WORLD);

MPI\_Send(shiftPtr(MO.getPtrToArray(), 1), 2\*N\*H, MPI\_INT, 5, 2, MPI\_COMM\_WORLD);

MPI\_Send(shiftPtr(ME.getPtrToArray(), 1), 2\*N\*H, MPI\_INT, 5, 3, MPI\_COMM\_WORLD);

MPI\_Send(&l, 1, MPI\_INT, 5, 4, MPI\_COMM\_WORLD);

MO.delExcessive(0, H);

ME.delExcessive(0, H);

Matrix MA(N, H);

calculate(MA, MB, MC, MO, ME, l);

Matrix MAh2(N, 2\*H);

MPI\_Recv(MAh2.getPtrToArray(), 2\*H\*N, MPI\_INT, 5, 31, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

MA.merge(MAh2);

MPI\_Gatherv(MA.getPtrToArray(), counts[4], MPI\_INT, MA.getPtrToArray(),

recvcounts, displs1, MPI\_INT, 0, star);

}

void threadFunc5(){

Matrix MB(N), MC(N), MO(N, 2\*H), ME(N, 2\*H);

int l;

MPI\_Recv(MB.getPtrToArray(), N\*N, MPI\_INT, 4, 0, MPI\_COMM\_WORLD, MPI\_STATUSES\_IGNORE);

MPI\_Recv(MC.getPtrToArray(), N\*N, MPI\_INT, 4, 1, MPI\_COMM\_WORLD, MPI\_STATUSES\_IGNORE);

MPI\_Recv(MO.getPtrToArray(), 2\*H\*N, MPI\_INT, 4, 2, MPI\_COMM\_WORLD, MPI\_STATUSES\_IGNORE);

MPI\_Recv(ME.getPtrToArray(), 2\*H\*N, MPI\_INT, 4, 3, MPI\_COMM\_WORLD, MPI\_STATUSES\_IGNORE);

MPI\_Recv(&l, 1, MPI\_INT, 4, 4, MPI\_COMM\_WORLD, MPI\_STATUSES\_IGNORE);

MPI\_Send(MB.getPtrToArray(), N\*N, MPI\_INT, 6, 0, MPI\_COMM\_WORLD);

MPI\_Send(MC.getPtrToArray(), N\*N, MPI\_INT, 6, 1, MPI\_COMM\_WORLD);

MPI\_Send(shiftPtr(MO.getPtrToArray(), 1), N\*H, MPI\_INT, 6, 2, MPI\_COMM\_WORLD);

MPI\_Send(shiftPtr(ME.getPtrToArray(), 1), N\*H, MPI\_INT, 6, 3, MPI\_COMM\_WORLD);

MPI\_Send(&l, 1, MPI\_INT, 6, 4, MPI\_COMM\_WORLD);

MO.delExcessive(0, H);

ME.delExcessive(0, H);

Matrix MA(N, H);

calculate(MA, MB, MC, MO, ME, l);

sendLeftMAh(5, MA, H);

}

void threadFunc6(){

Matrix MB(N), MC(N), MO(N, H), ME(N, H);

int l;

MPI\_Recv(MB.getPtrToArray(), N\*N, MPI\_INT, 5, 0, MPI\_COMM\_WORLD, MPI\_STATUSES\_IGNORE);

MPI\_Recv(MC.getPtrToArray(), N\*N, MPI\_INT, 5, 1, MPI\_COMM\_WORLD, MPI\_STATUSES\_IGNORE);

MPI\_Recv(MO.getPtrToArray(), H\*N, MPI\_INT, 5, 2, MPI\_COMM\_WORLD, MPI\_STATUSES\_IGNORE);

MPI\_Recv(ME.getPtrToArray(), H\*N, MPI\_INT, 5, 3, MPI\_COMM\_WORLD, MPI\_STATUSES\_IGNORE);

MPI\_Recv(&l, 1, MPI\_INT, 5, 4, MPI\_COMM\_WORLD, MPI\_STATUSES\_IGNORE);

Matrix MA(N, H);

calculate(MA, MB, MC, MO, ME, l);

sendLeftMAh(6, MA, H);

}

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

{

MPI\_Init(&argc, &argv);

int rank;

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

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

H = N/P;

MPI\_Comm\_group(MPI\_COMM\_WORLD, &world\_group);

MPI\_Group\_incl(world\_group, 5, star\_ranks, &star\_group);

MPI\_Comm\_create(MPI\_COMM\_WORLD, star\_group, &star\_comm);

cout << "Thread " << rank << " started.\n";

switch(rank){

case 0: threadFunc0(star\_comm);

break;

case 1: threadFunc13(star\_comm);

break;

case 2: threadFunc13(star\_comm);

break;

case 3: threadFunc13(star\_comm);

break;

case 4: threadFunc4(star\_comm);

break;

case 5: threadFunc5();

break;

case 6: threadFunc6();

break;

}

cout << "Thread " << rank << " finish.\n";

if(rank == 0)

getchar();

MPI\_Finalize();

return 0;

}