ЛР-4

Вариант 1

Тюльников Михаил ПИН-32

[Год]

#include <iostream>

#include <cstdlib>

#include <cmath>

#include <mpi.h>

#include <cstddef>

constexpr int MAX\_SIZE = 1000;

constexpr int SIZE = 10;

struct LONG\_INT {

int size;

int base;

int number[MAX\_SIZE];

};

LONG\_INT multiply(LONG\_INT left, LONG\_INT right);

void printLongInt(LONG\_INT x);

int getNewSize(LONG\_INT x);

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

MPI\_Init(&argc, &argv);

int rank, size;

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

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

int n\_structure\_per\_process = 4;

int n\_structure = n\_structure\_per\_process \* size;

LONG\_INT\* data = nullptr;

if (rank == 0)

{

srand(rank \* 10);

data = new LONG\_INT[n\_structure];

for (int i = 0; i < n\_structure; ++i) {

data[i].size = SIZE;

data[i].base = 1;

for (int j = 0; j < MAX\_SIZE; ++j) {

data[i].number[j] = (j < data[i].size ? rand() % 10 : 0);

}

std::cout << i << ": ";

printLongInt(data[i]);

std::cout << std::endl;

}

}

MPI\_Aint displacements[3] = { offsetof(LONG\_INT, size), offsetof(LONG\_INT, base), offsetof(LONG\_INT, number) };

int block\_lengths[3] = { 1, 1, MAX\_SIZE };

MPI\_Datatype types[3] = { MPI\_INT, MPI\_INT, MPI\_INT };

MPI\_Datatype custom\_dt;

MPI\_Type\_create\_struct(3, block\_lengths, displacements, types, &custom\_dt);

MPI\_Type\_commit(&custom\_dt);

LONG\_INT\* gathered\_data = nullptr;

LONG\_INT\* scatter\_data = new LONG\_INT[n\_structure\_per\_process];

LONG\_INT result;

result.size = 1;

result.base = 1;

for (int j = 0; j < MAX\_SIZE; ++j)

result.number[j] = (j == 0 ? 1 : 0);

if (rank == 0)

gathered\_data = new LONG\_INT[size];

MPI\_Scatter(data, n\_structure\_per\_process, custom\_dt, scatter\_data, n\_structure\_per\_process, custom\_dt, 0, MPI\_COMM\_WORLD);

for (int i = 0; i < n\_structure\_per\_process; i++) {

result = multiply(result, scatter\_data[i]);

}

MPI\_Gather(&result, 1, custom\_dt, gathered\_data, 1, custom\_dt, 0, MPI\_COMM\_WORLD);

if (rank == 0) {

result = gathered\_data[0];

for (int i = 1; i < size; ++i) {

result = multiply(result, gathered\_data[i]);

}

printLongInt(result);

}

MPI\_Finalize();

return 0;

}

LONG\_INT multiply(LONG\_INT left, LONG\_INT right) {

int base = 10;

LONG\_INT result;

result.size = left.size + right.size;

result.base = 1;

for (int j = 0; j < MAX\_SIZE; ++j)

result.number[j] = 0;

for (int i = 0; i < left.size; i++) {

int carry = 0;

for (int j = 0; j < right.size || carry != 0; j++) {

int cur = result.number[i + j] + left.number[i] \* (j < right.size ? right.number[j] : 0) + carry;

result.number[i + j] = cur % base;

carry = cur / base;

}

}

result.size = getNewSize(result);

return result;

}

int getNewSize(LONG\_INT x) {

int newSize = x.size;

while (x.number[newSize - 1] == 0) {

newSize--;

}

return newSize;

}

void printLongInt(LONG\_INT x) {

for (int i = x.size - 1; i >= 0; --i) {

std::cout << x.number[i];

}

}



