**Федеральное государственное бюджетное образовательное учреждение высшего образования**

**«Уфимский университет науки и технологий»**

**Кафедра** Высокопроизводительных вычислительных технологий и систем

**Дисциплина:** Параллельное программирование

**Отчет по лабораторной работе № 2**

**Тема:** «Параллельное вычисление произведения двух матриц»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Группа ПМИ-354 | Фамилия И.О. | Подпись | Дата | Оценка |
| Студент | Мова И.А. |  |  |  |
| Принял | Спеле В.В. |  |  |  |

**Уфа 2023**

**Цель:** научиться использовать принцип геометрической декомпозиции в параллельных алгоритмах и создавать параллельные программы для систем с распределенной памятью с использованием коллективных функций MPI на примере вычисления произведения двух матриц

**Теоретический материал**

*MPI (Message Passing Interface****)*** – это стандартизованная библиотека функций, призванная обеспечить совместную работу параллельных процессов путем организации передачи сообщений между ними.

*Основные понятия MPI*

* Каждая программа представляет собой совокупность одновременно работающих процессов, которые могут обмениваться сообщениями;
* Все процессы объединяются в группы;
* Обмен сообщениями возможен между процессами одной группы, которой поставлена в соответствие своя область связи;
* Коммуникатор – идентификатор области;
* При запуске программы все доступные ей процессы объединяются в начальную группу с общей областью связи, имеющей коммуникатор MPI\_COMM\_WORLD.

*Особенности реализации MPI для C/C++*

1. Первая строка программы: #include “mpi.h”;
2. В MPI принят ANSI C стандарт;
3. Нумерация массивов начинается с 0;
4. Массивы хранятся по строкам;
5. Логические переменные являются переменными типа integer со значением 0 в случае false и любым не нулевым значением, обозначающем true.

*Базовые функции MPI*

1. int MPI\_Init(int \*argc, char \*\*argv[]); – инициализация параллельной части программы.

Возвращает предопределенные константы

MPI\_SUCCESS – возвращается в случае успешного выполнения,

MPI\_ERR\_ARG – ошибка неправильного задания аргумента,

MPI\_ERR\_INTERN – внутренняя ошибка (нехватка памяти),

MPI\_ERR\_UNKNOWN – неизвестная ошибка.

1. int MPI\_Finalize(void); – завершение параллельной части программы.
2. int MPI\_Comm\_size(MPI\_Comm comm, int\* size); – определение числа процессов size в коммуникационной группе с коммуникатором comm.
3. int MPI\_Comm\_rank(MPI\_comm comm, int\* rank); – определение номера rank вызвавшего ее процесса, входящего в коммуникационную группу с коммуникатором comm
4. int MPI\_Send(void\* sbuf, int count, MPI\_Datatype datatype, int dest, int tag, MPI\_Comm comm) – передача сообщений от одного процесса к другому

sbuf – адрес в памяти, начиная с которого размещаются передаваемые данные;

count – количество передаваемых элементов;

datatype – тип передаваемых элементов;

dest – номер процесса-получателя сообщения;

tag – метка передаваемого сообщения;

comm – коммуникатор.

1. int MPI\_Recv(void\* rbuf, int count, MPI\_Datatype datatype, int source, int tag, MPI\_comm comm, MPI\_Status \*status) – прием сообщений от одного процесса к другому.

*Входные параметры:*

count – количество получаемых элементов;

datatype – тип получаемых элементов;

source – номер процесса-отправителя сообщения;

tag – метка принимаемого сообщения;

comm – коммуникатор.

*Выходные параметры:*

rbuf – адрес в памяти, начиная с которого размещаются принимаемые данные;

status – структура, содержащая информацию о принятом сообщении.

Структура status имеет три поля.

Status.MPI\_SOURCE – номер процесса-отправителя;

Status.MPI\_TAG – метка принимаемого сообщения;

Status.MPI\_ERROR – код завершения приема сообщения.

*Функция совмещенного приема/передачи.*

int MPI\_Sendrecv(void\* sbuf, int scount, MPI\_Datatype sdatatype, int dest, int stag, void\* rbuf, int rcount, MPI\_Datatype rdatatype, int source, int rtag, MPI\_comm comm, MPI\_Status \*status);

*Входные параметры:*

sbuf – адрес в памяти, начиная с которого размещаются передаваемые данные;

scount – количество передаваемых элементов;

sdatatype – тип отправляемых элементов;

dest – номер процесса-получателя сообщения;

stag – метка отправляемого сообщения;

rcount – количество получаемых элементов;

rdatatype – тип получаемых элементов;

source – номер процесса-отправителя сообщения;

rtag – метка принимаемого сообщения;

comm – коммуникатор.

*Выходные параметры:*

rbuf – адрес в памяти, начиная с которого размещаются принимаемые данные;

status – структура, содержащая информацию о принятом сообщении.

Для анализа данных, полученных в ходе проведения лабораторной работы, будут использоваться следующие определения:

* Отношение времени выполнения параллельной программы на одном процессоре (ядре) ко времени выполнения параллельной программы на процессорах называется ускорением при использовании p процессоров:

* Отношение ускорения к количеству процессоров называется эффективностью при использовании процессоров:

*Функции коллективного взаимодействия*

1. int ***MPI\_Bcast***(void \*buf, int count, MPI\_Datatype datatype, int root, MPI\_Comm comm);Функция предназначена для рассылки данных, хранящихся на одном процессе, всем остальным процессам группы.

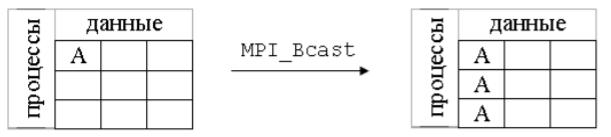


Рисунок 1. MPI\_Bcast

*Входные параметры:*

buf – адрес в памяти, начиная с которого размещаются передаваемые данные;

count – количество рассылаемых элементов;

datatype – тип отправляемых данных;

root – номер процесса-отправителя сообщения;

comm – коммуникатор.

1. int ***MPI\_Gather***(void \*sbuf, int scount, MPI\_Datatype stype, void \*rbuf, int rcount, MPI\_Datatype rtype, int root, MPI\_Comm comm);

Функция предназначена для сбора данных со всех процессов на одном (так называемый, "совок“).



Рисунок 2. MPI\_Gather

*Входные параметры*:

sbuf – адрес в памяти на каждом процессе, начиная с которого размещаются отправляемые данные;

scount – количество элементов, отправляемых с каждого процесса;

stype – тип отправляемых данных;

rcount – количество принимаемых элементов от каждого процесса;

rtype – тип принимаемых данных;

root – номер процесса, на котором осуществляется сборка сообщений;

comm – коммуникатор.Выходные параметры:

rbuf – адрес в памяти на процессе с номером root, начиная с которого размещаются принимаемые сообщения.

1. Вариант функции с варьируемым кол-вом собираемых элементов:

int ***MPI\_Gatherv***(void \*sbuf, int scount, MPI\_Datatype stype, void \*rbuf, int \*rcounts, int \*displs, MPI\_Datatype rtype, int root, MPI\_Comm comm);

*Характерные отличия:*

rcounts – массив длин принимаемых от процессов сообщений;

displs – массив позиций в приемном буфере, по которым размещаются принимаемые сообщения.

1. int ***MPI\_Scatter***(void \*sbuf, int scount, MPI\_Datatype stype, void \*rbuf, int rcount, MPI\_Datatype rtype, int root, MPI\_Comm comm);

Функция предназначена для рассылки данных с одного процесса всем остальным процессам (так называемый, “разбрызгиватель”).

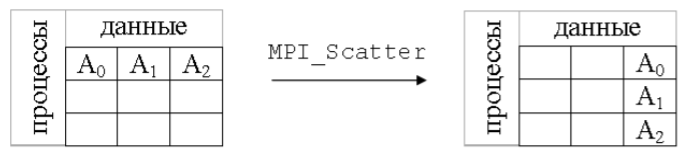


Рисунок 3. MPI\_Scatter

*Входные параметры:*

sbuf – адрес в памяти на процессе-отправителе сообщения, начиная с которого размещаются отправляемые данные;

scount – количество элементов, отправляемых каждому процессу;

stype – тип отправляемых данных;

rcount – количество элементов, принимаемых каждым процессом (длина принимаемого сообщения);

rtype – тип принимаемых данных;

root – номер процесса-отправителя сообщения;

comm – коммуникатор.Выходные параметры:

rbuf – адрес в памяти на каждом процессе, начиная с которого размещаются принимаемые сообщения.

1. Вариант функции с варьируемым кол-вом рассылаемых элементов:

int ***MPI\_Scatterv***(void \*sbuf, int \*scounts, int \*displs, MPI\_Datatype stype, void \*rbuf, int rcount, MPI\_Datatype rtype, int root, MPI\_Comm comm);

*Характерные отличия:*

scounts – массив, содержащий количество элементов в каждой части, на которые разбивается сообщение;

displs – массив позиций, определяющий начальные положения каждой части сообщения.

1. int ***MPI\_Reduce***(void \*sbuf, void \*rbuf, int count, MPI\_Datatype datatype, MPI\_Op op, int root, MPI\_Comm comm);

Функция проделывает операцию op над данными, хранящимися в sbuf в каждом процессе группы, результат которой записывается в rbuf в процесс с номером root.

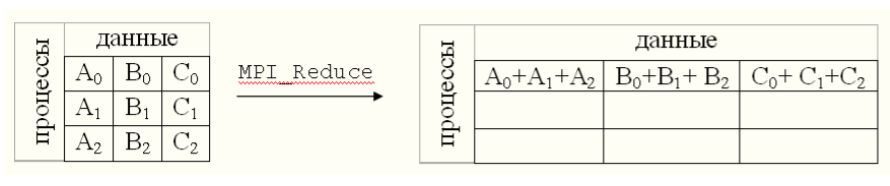


Рисунок 4. MPI\_Reduce

*Входные параметры:*

sbuf – адрес в памяти на каждом процессе, по которому хранятся исходные данные для распределенной операции;

count – количество элементов в sbuf;

datatype – тип данных, над которыми производится распределенная операция;

root – номер процесса, на котором осуществляется размещение результата выполнения операции;

op – название распределенной операции;

comm – коммуникатор.

*Выходные параметры:*

rbuf – адрес в памяти, по которому размещаются результаты выполнения операции.

***12 предопределенных операций:***

MPI\_MAX – поиск поэлементного максимума;

MPI\_MIN – поиск поэлементного минимума;

MPI\_SUM – вычисление суммы векторов;

MPI\_PROD – вычисление поэлементного произведения векторов;

MPI\_LAND – логическое “И”;

MPI\_LOR – логическое “ИЛИ”;

MPI\_LXOR – логическое исключающее “ИЛИ”;

MPI\_BAND – бинарное “И”;

MPI\_BOR – бинарное “ИЛИ”;

MPI\_BXOR – бинарное исключающее ИЛИ;

MPI\_MAXLOC – поиск индексированного максимума;

MPI\_MINLOC – поиск индексированного минимума.

**Практическая часть**

|  |  |  |
| --- | --- | --- |
|  |  |  |
| 1 | 60,49 | 563,83 |
| 2 | 30,79 | 283,96 |
| 4 | 18,04 | 150,67 |
| 8 | 10,95 | 80,81 |
| 16 | 5,56 | 40,88 |
| 32 | 2,94 | 20,83 |
| 64 | 1,68 | 10,76 |
| 96 | 1,23 | 7,33 |

Таблица 1. Зависимость времени работы программы от количества процессов

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | |  | |
|  |  |  |  |
| 1 | 1 | 1 | 1 | 1 |
| 2 | 1,9645989 | 1,98559656 | 0,982299 | 0,992798 |
| 4 | 3,21926557 | 3,742151172 | 0,804816 | 0,935538 |
| 8 | 5,52420091 | 6,97723054 | 0,690525 | 0,872154 |
| 16 | 10,8794964 | 13,792319 | 0,679969 | 0,86202 |
| 32 | 20,5748299 | 27,0681709 | 0,642963 | 0,84588 |
| 64 | 36,0059524 | 52,4005576 | 0,562593 | 0,818759 |
| 96 | 49,1788618 | 76,9208731 | 0,51228 | 0,801259 |

Таблица 2. Вычисление ускорений и эффективности для разного числа процессов

Рисунок 5. График зависимости ускорения от числа процессов

Рисунок 6. График зависимости эффективности от числа процессов

**Вывод**: в ходе лабораторной работы на примере задачи параллельного умножения матриц научились программно реализовывать простейшие параллельные вычислительные алгоритмы и проводить анализ их эффективности.

**Код программы.**

#include "mpi.h"

#include <iostream>

#include <math.h>

#include <stdlib.h>

#include <stdio.h>

using namespace std;

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

{

srand((unsigned int)1);

int MyID, NumProc;

ierror = MPI\_Init(&argc, &argv);

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

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

long long L;

L = atoi(argv[1]);

long long N = 10 \* L, p = N / NumProc, o = N % NumProc;

double \*A = (double\*)calloc(N\*L, sizeof(double)),

\*B = (double\*)calloc(L\*L, sizeof(double)),

\*C = (double\*)calloc(N\*L, sizeof(double)),

\*Ap = (double\*)calloc((p + 1)\*L, sizeof(double)),

\*Cp = (double\*)calloc((p + 1)\*L, sizeof(double)),

\*normid = (double\*)calloc(1, sizeof(double)),

\*Norma = (double\*)calloc(1, sizeof(double));

int \*count = (int\*)calloc(NumProc, sizeof(int)), \*displs = (int\*)calloc(NumProc, sizeof(int)),

\*end=(int\*)calloc(NumProc, sizeof(int)),

ccount;

double tstart, tfinish;

if (MyID == 0)

{

for (int i = 0; i < N\*L; i++) A[i] = -0.5 + (double)rand() / RAND\_MAX;;

for (int i = 0; i < L\*L; i++) B[i] = -0.5 + (double)rand() / RAND\_MAX;;

tstart = MPI\_Wtime();

}

MPI\_Barrier(MPI\_COMM\_WORLD);

if (MyID == 0)

{

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

{

if (o > i)

{

count[i] = (p + 1)\*L;

displs[i] = i \* (p + 1)\*L;

}

else

{

count[i] = p \* L;

displs[i] = o \* (p + 1)\*L + (i - o)\*p\*L;

}

}

}

if (MyID < o)

{

ccount = (p + 1)\*L;

end[MyID] = p + 1;

}

else

{

end[MyID] = p;

ccount = (p)\*L;

}

MPI\_Scatterv(A, count, displs, MPI\_DOUBLE, Ap, ccount, MPI\_DOUBLE, 0, MPI\_COMM\_WORLD);

MPI\_Bcast(B, L\*L, MPI\_DOUBLE, 0, MPI\_COMM\_WORLD);

MPI\_Barrier(MPI\_COMM\_WORLD);

for (int i = 0; i < end[MyID]; i++)

{

for (int j = 0; j < L; j++)

{

Cp[L \* (i)+j] = .0;

for (int k = 0; k < L; k++)

{

Cp[L \* (i)+j] += Ap[L \* (i)+k] \* B[L \* (k)+j];

}

normid[0] += Cp[L \* (i)+j] \* Cp[L \* (i)+j];

}

}

MPI\_Gatherv(Cp, ccount, MPI\_DOUBLE, C, count, displs, MPI\_DOUBLE, 0, MPI\_COMM\_WORLD);

MPI\_Reduce(normid, Norma, 1, MPI\_DOUBLE, MPI\_SUM, 0, MPI\_COMM\_WORLD);

if (MyID == 0)

{

tfinish = MPI\_Wtime() - tstart;

cout << "> Norm = " << Norma[0] << endl;

cout << "> Time = " << tfinish << endl;

}

free(A);

free(C);

free(Ap);

free(B);

free(Cp);

free(normid);

free(Norma);

free(count);

free(displs);

free(end);

MPI\_Finalize();

}