Лабораторная работа № 2

Параллельное вычисление произведения двух матриц

Цель работы

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.

Произведение двух матриц

Требуется вычислить произведение матриц $A(N \times L)$ и $B(L \times L)$ и вычислить квадрат евклидовой нормы результирующей матрицы:

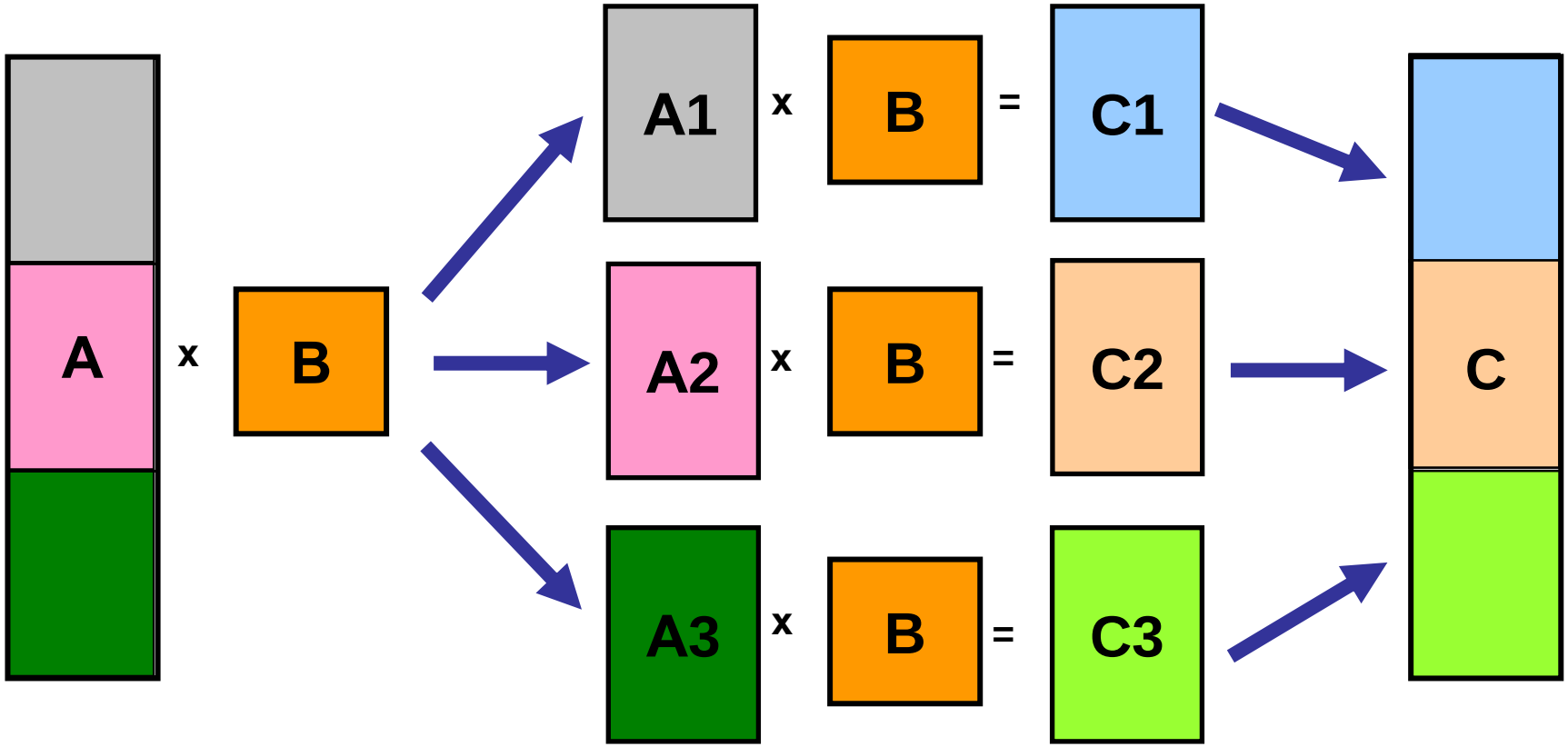
$$\|C\|^2 = \sum_{i=1}^{N} \sum_{j=1}^{L} c_{ij}^2, \qquad C = AB.$$

Матричное умножение выражается формулой

$$c_{ik} = \sum_{j=1}^{L} a_{ij} b_{jk}, \qquad i = 1,\ldots,N;\ j,k = 1,\ldots,L.$$

Произведение двух матриц

Способ №1. Строчная непрерывная декомпозиция матрицы A при ее равномерном распределении по процессам.



Задание

1. Написать программу с использованием коллективных функций MPI
2. В программе предусмотреть следующее

а) размерность ***L*** вводится пользователем (аргумент командной строки), размерность ***N=10L***;

б) матрицы заполняются случайными элементами вещественного типа;

в) замер времени выполнения параллельной программы (вычисления + коммуникации).

Задание

3. Запустить программу на кластере при числе процессов `p = 1, 2, 4, 8, 16, 32, 64, 128`, размерность подобрать так, чтобы время работы программы при `p = 1` составляло около 60 и 600 с.

4. Результаты замеров времени работы программы занести в таблицу. Вычислить ускорение и эффективность, построить их графики в зависимости от числа процессов.

5. Составить отчет с оценкой полученных результатов.

УГАТУ

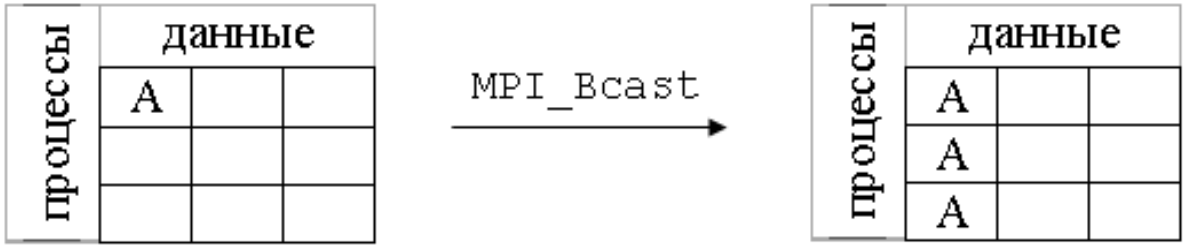
Некоторые функции коллективного взаимодействия

Функции коллективного взаимодействия

УГАТУ

```
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.



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.

Пример

Пусть требуется передать значения целочисленного массива **a**, состоящего из пяти элементов, из нулевого процесса во все остальные процессы группы.

С помощью функции **MPI_Bcast** это может быть реализовано следующим образом:

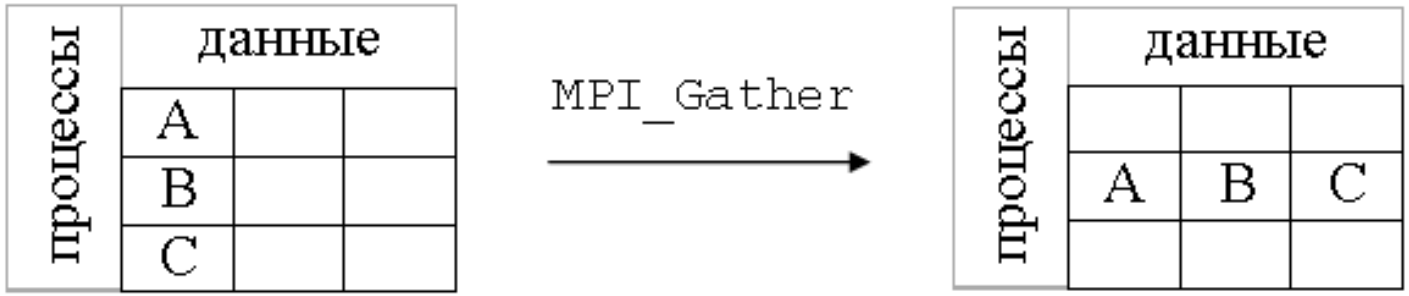
```
int a[5];
. . .
MPI_Bcast(a, 5, MPI_INT, 0, MPI_COMM_WORLD);
. . .
```

Функции коллективного взаимодействия

УГАТУ

```
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“).



Функции коллективного взаимодействия

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.

Пример

Пусть во всех процессах группы, состоящей из 5 процессов (**NumProc = 5**), имеется вещественный массив **AS** из 10 элементов. Требуется собрать все массивы в единый общий массив **AR** в порядке следования номеров процессов на процессе с номером 2.

Тогда достаточно одного вызова функции **MPI_Gather**:

```
double AS[10], AR[50];
. . .
MPI_Gather(AS, 10, MPI_DOUBLE, AR, 10,
           MPI_DOUBLE, 2, MPI_COMM_WORLD);
. . .
```

Функции коллективного взаимодействия

УГАТУ

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);
```

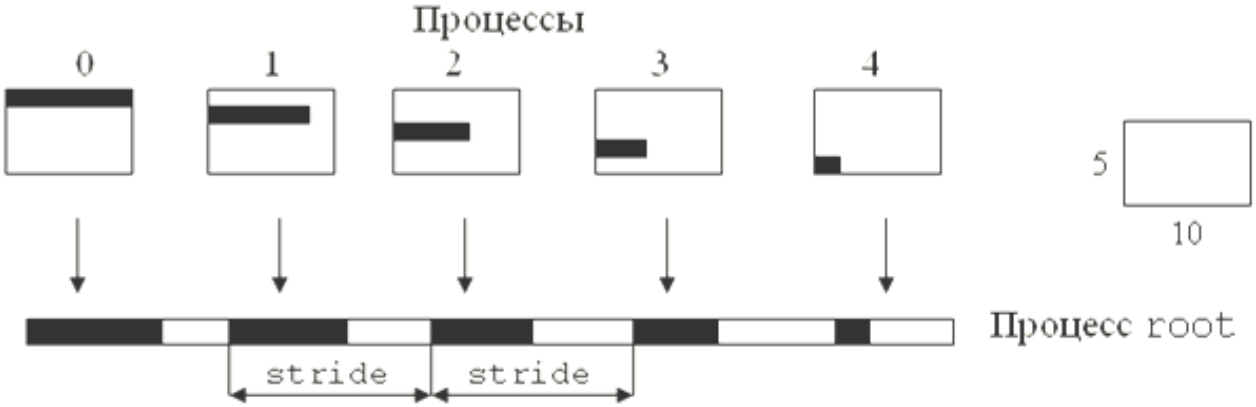
Характерные отличия:

rcounts – массив длин принимаемых от процессов сообщений;

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

Пример

Пусть имеется группа из пяти процессов, в каждом из которых определен двумерный целочисленный массив **`AS[5][10]`**. Требуется осуществить выборочную сборку данных со всех процессов на третьем процессе по следующему правилу: с *i*-го процесса (***`i = 0,1…,4`***) выбираются (**`10–2i`**) первых значений из ***`i`***-й строки массива **`AS`** и пересылаются в одномерный целочисленный массив **`AR`**. При этом расстояние между началами принятых блоков (для него используется обозначение **`stride`**) должно составлять 12 элементов.



Функции коллективного взаимодействия

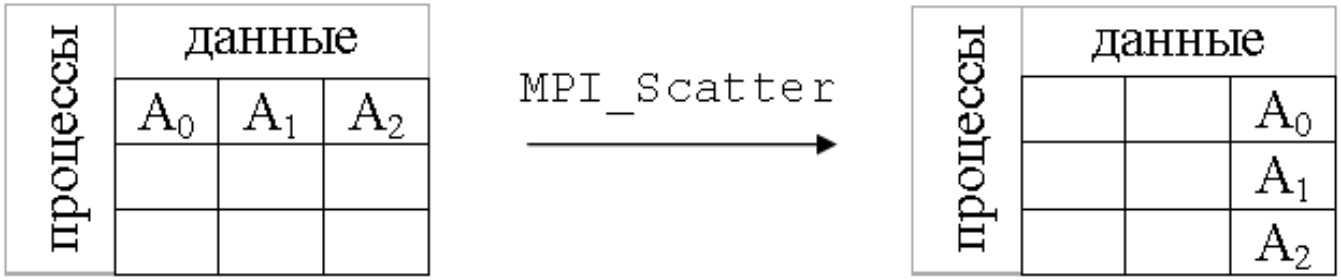

```
#include "mpi.h"
int main(int argc, char *argv[])
{int AS[5][10], AR[60], rcounts[5], displs[5];
 int i, MyID, NumProc, stride=12, scount, root=3;
 MPI_Init(&argc, &argv);
 MPI_Comm_size(MPI_COMM_WORLD, &NumProc);
 MPI_Comm_rank(MPI_COMM_WORLD, &MyID);
 . . . // Заполнение массива AS
 scount=10-2*MyID;  // Количество отсылаемых элементов
 if(MyID==root)                    // Заполнение массивов
   for(i=0; i<NumProc; i++)        // displs и rcounts
   { displs[i]=i*stride;           // на процессе-сборщике
     rcounts[i]=10-2*i;}           // сообщений.
 MPI_Gatherv(&AS[MyID][0], scount, MPI_INT, AR, rcounts,
             displs, MPI_INT, root, MPI_COMM_WORLD);
 . . . // Продолжение программы
 MPI_Finalize();
 return 0;}
```

Функции коллективного взаимодействия

УГАТУ

```
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”).



Функции коллективного взаимодействия

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.

Пример

Пусть необходимо вещественный массив **AS**, состоящий из **10** элементов и хранящийся на первом процессе, распределить между всеми процессами группы, состоящей из **5** процессов. В результате в каждый процесс пересылается **2** элемента, которые записываются в массив **AR**.

```
double AS[10], AR[2];
. . .
MPI_Scatter(AS, 2, MPI_DOUBLE, AR, 2,
            MPI_DOUBLE, 1, MPI_COMM_WORLD);
. . .
```

Функции коллективного взаимодействия

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.

Пример

Рассмотрим операцию, обратную рассмотренной в примере на функцию **`MPI_Gatherv`**. Пусть требуется распределить массив **`ASN[60]`** со структурой, аналогичной структуре массива **`AR[60]`** примера на функцию **`MPI_Gatherv`**, по массивам **`ARN[5][10]`** по правилу: на процессе с номером ***`i`*** сообщения размещаются с нулевой позиции ***`i`***-й строки.

Функции коллективного взаимодействия

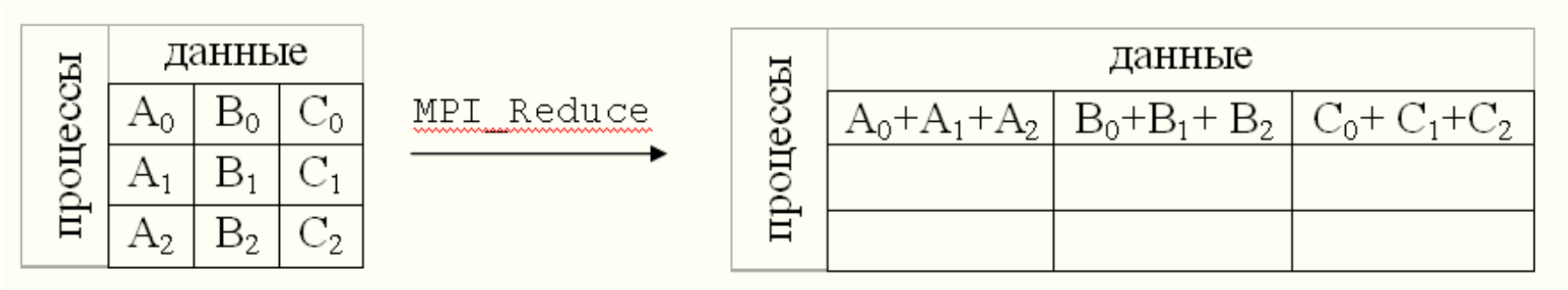

```
#include "mpi.h"
int main(int argc, char *argv[])
{int ARN[5][10], ASN[60], scounts[5], displs[5];
 int i, MyID, NumProc, stride=12, rcount, root=3;
 MPI_Init(&argc, &argv);
 MPI_Comm_size(MPI_COMM_WORLD, &NumProc);
 MPI_Comm_rank(MPI_COMM_WORLD, &MyID);
 . . . // Заполнение массива ASN
 rcount=10-2*MyID;  // Количество принимаемых элементов
 if(MyID==root)                    // Заполнение массивов
   for(i=0;i<NumProc;i++)   // displs и rcounts
   { displs[i]=i*stride;      // на процессе-отправителе
     rcounts[i]=10-2*i;}      // сообщения
 MPI_Scatterv(ASN,scounts,displs,MPI_INT,&ARN[MyID][0],
              rcount,MPI_INT,root,MPI_COMM_WORLD);
 . . . // Продолжение программы
 MPI_Finalize();
 return 0;}
```

Функции коллективного взаимодействия

УГАТУ

```
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**.



Функции коллективного взаимодействия

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.

Функции коллективного взаимодействия

УГАТУ

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.