КАЗАНСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ

ИНСТИТУТ ФИЗИКИ

Кафедра радиоастрономии

О.Г. Хуторова

ОСНОВЫ РАБОТЫ С БИБЛИОТЕКОЙ MPI

Учебно-методическое пособие

Казань-2022

УДК 004.032.24

Принято на заседании учебно-методической комиссии Института физики протокол № 1 от 29 октября.2022

Рецензент:

кандидат технических наук,

доцент кафедры компьютерных систем ФГБОУ ВО «КНИТУ-КАИ» Р.К. Классен

Хуторова О.Г.

Основы работы с библиотекой MPI

Учебно-методическое пособие / Хуторова О.Г. – Казань: Казан. ун-т., 2022. – 32 с.

Методическое пособие предназначено для обучающихся по программе «Введение в высокопроизводительные вычислительные системы», «Многопоточная обработка больших данных». Пособие включает в себя описание основных функций библиотеки MPI, особенностей их работы и практические задания для освоения программирования многопроцессорных систем с распределенной памятью. Может быть полезно студентам, аспирантам, слушателям ФПК. Создано на основе переработанной части ранее изданного пособия «Введение в современные высокопроизводительные вычислительные системы»/ Тептин Г. М., Хуторова О. Г, Зинин Д. П., 2015.

Оглавление

Идеология MPI

Важной составляющей деятельности в области физики, астрономии, защиты информации является реализация сложных численных методов, требующих грамотного использования многопроцессорных вычислительных систем. MPI – message passing interface – библиотека функций, предназначенная для поддержки работы параллельных процессов в терминах передачи сообщений, технология разработки параллельных программ для многопроцессорных систем с распределенной памятью (кластерных систем, массивно-параллельных суперкомпьютеров). Библиотека MPI разработана с целью стандартизации разработки параллельных программ, что позволяет снизить остроту проблемы переносимости параллельных программ между разными компьютерными системами, содействует повышению эффективности параллельных вычислений, т.к. в настоящее время практически для каждого типа вычислительных систем существуют реализации библиотек MPI, в максимальной степени учитывающие возможности используемого компьютерного оборудования. MPI уменьшает сложность разработки параллельных программ, т.к. большая часть основных операций передачи данных предусматривается стандартом MPI, и имеется большое количество библиотек параллельных методов, созданных с использованием MPI.

Под параллельной программой понимается множество одновременно выполняемых процессов, взаимодействующих между собой с помощью сообщений. Каждый процесс порождается на основе копии одного и того же программного кода. Количество процессов и число используемых процессоров определяется в момент запуска параллельной программы средствами среды исполнения MPI-программ и в ходе вычислений меняться не может (в стандарте MPI-2 предусматривается возможность динамического изменения количества процессов).

Все процессы программы перенумерованы от 0 до p – 1, где p – общее количество процессов. Номер процесса (ранг) – целое неотрицательное число, уникальный атрибут каждого процесса.

Процессы обмениваются сообщениями. Сообщение – набор данных некоторого типа. Атрибуты сообщения – номер процесса-отправителя, номер процесса-получателя и идентификатор сообщения. Для них заведена структура MPI_Status, содержащая три поля: MPI_Source (номер процесса отправителя), MPI_Tag (идентификатор сообщения), MPI_Error (код ошибки); могут быть и добавочные поля. Идентификатор сообщения (msgtag) – атрибут сообщения, являющийся целым неотрицательным числом, лежащим в диапазоне от 0 до 32767.

Процессы параллельной программы объединяются в группы (коммуникаторы), указание используемого коммуникатора обязательно для операций передачи данных в MPI. Парные операции передачи данных выполняются для процессов, принадлежащих одному и тому же коммуникатору. Коллективные операции применяются одновременно для всех процессов коммуникатора.

В ходе вычислений могут создаваться новые и удаляться существующие группы процессов и коммуникаторы. Один и тот же процесс может принадлежать разным группам и коммуникаторам.

При необходимости передачи данных между процессами из разных групп необходимо создавать глобальный коммуникатор (intercommunicator).

MPI_COMM_WORLD – включает все процессы параллельной программы;

MPI_COMM_SELF – включает только данный процесс;

MPI_COMM_NULL – пустой коммуникатор, не содержит ни одного процесса.

При выполнении операций передачи сообщений для указания передаваемых или получаемых данных в функциях MPI необходимо указывать *тип* пересылаемых данных.

Привязка к языку C и C++

Имена подпрограмм и констант MPI в программах на языке C имеют префикс MPI_ . Константы и функции MPI описываются в файле mpi.h, который включается в MPI-программу с помощью директивы

#include “mpi.h”

Соответствие типов MPI стандартным типам языка C приведено ниже

MPI_CHAR – Signed char
MPI_SHORT – Signed short int
MPI_INT – Signed int
MPI_LONG – Signed long int
MPI_UNSIGNED_CHAR – unsigned char
MPI_UNSIGNED_SHORT – unsigned short int
MPI_UNSIGNED – unsigned int
MPI_UNSIGNED_LONG – unsigned long int
MPI_FLOAT – Float
MPI_DOUBLE – Double
MPI_LONG_DOUBLE – long double
MPI_BYTE – Нет соответствия
MPI_PACKED – Нет соответствия

Ниже приведены основные функции MPI.

Подключение MPI

```
int MPI_Init(int *argc, char **argv)
```

Аргументы argc и argv требуются только в программах на C, где они задают количество аргументов командной строки запуска программы и вектор этих аргументов.

Завершение работы с MPI

int MPI_Finalize()

После вызова данной подпрограммы нельзя вызывать подпрограммы MPI. MPI_FINALIZE должны вызывать все процессы перед завершением своей работы.

Определение размера области взаимодействия

int MPI_Comm_size(MPI_Comm comm, int *size)

Входные параметры: comm – коммуникатор.

Выходные параметры: size – количество процессов в области взаимодействия.

Пример:

MPI_Comm_size (MPI_COMM_WORLD, &ProcNum);

Коммуникатор *MPI_COMM_WORLD* создается по умолчанию и представляет все процессы выполняемой параллельной программы,

Определение ранга процесса

int MPI_Comm_rank(MPI_Comm comm, int *rank)

Входные параметры: comm – коммуникатор.

Выходные параметры: rank – ранг процесса в области взаимодействия.

Ранг, получаемый при помощи функции *MPI_Comm_rank*, является рангом процесса, выполнившего вызов этой функции, т.е. переменная *ProcRank* будет принимать различные значения в разных процессах.

Пример:

```
MPI_Comm_rank(MPI_COMM_WORLD, &ProcRank);
if ( ProcRank == 0 ) DoMasterProcess();
else DoSlaveProcesses();
```

Структура программы с MPI имеет вид:

```
#include "mpi.h"
int main ( int argc, char *argv[] ) {
int ProcNum, ProcRank;
// программный код без использования MPI функций
…
MPI_Init ( &agrc, &argv );
MPI_Comm_size ( MPI_COMM_WORLD, &ProcNum);
MPI_Comm_rank ( MPI_COMM_WORLD, &ProcRank);
// программный код с использованием MPI функций
…
MPI_Finalize();
// программный код без использования MPI функций return 0;
```

Определение имени узла, на котором выполняется данный процесс

MPI_Get_processor_name(char *name, int *resultlen)

Выходные параметры:

name – идентификатор вычислительного узла. Массив не менее чем из MPI_MAX_PROCESSOR_NAME элементов;

resultlen – длина имени.

Среди функций MPI различаются парные (двухточечные) операции передачи сообщений между двумя процессами и коллективные коммуникационные действия. Для двухточечного обмена используют функции блокирующие (прием/передача приостанавливают выполнение процесса на время приема сообщения) и неблокирующие (прием/передача выполнение процесса продолжается в фоновом режиме, а программа в нужный момент может запросить подтверждение завершения приема).

Стандартная блокирующая передача сообщений

int MPI_Send(void *buf, int count, MPI_Datatype type, int dest, int tag,
MPI_Comm comm)

buf – адрес буфера памяти, в котором располагаются данные отправляемого сообщения,

count – количество элементов данных в сообщении,

type – тип элементов данных пересылаемого сообщения,

dest – ранг процесса, которому отправляется сообщение,

tag – значение-тег, используемое для идентификации сообщений,

comm – коммуникатор, в рамках которого выполняется передача данных.

Стандартный блокирующий прием сообщений

int MPI_Recv(void *buf, int count, MPI_Datatype datatype, int source,int tag, MPI_Comm comm, MPI_Status *status)

Входные параметры:

count – максимальное количество элементов в буфере приема. Фактическое их количество можно определить с помощью подпрограммы MPI_Get_count;

datatype – тип принимаемых данных.

source – ранг источника. Можно использовать специальное значение MPI_ANY_SOURCE, соответствующее произвольному значению ранга.

tag – тег сообщения или "джокер" MPI_ANY_TAG, соответствующий произвольному значению тега;

comm – коммуникатор.

Выходные параметры:

buf – начальный адрес буфера приема. Его размер должен быть достаточным, чтобы разместить принимаемое сообщение, иначе при выполнении приема произойдет сбой

status – статус обмена.

Определение времени выполнения MPI-программы

double MPI_Wtime(void) – относительное время в секундах.

Для оценки времени выполнения фрагмента программы используют следующую конструкцию

```
double t1, t2, dt;
t1 = MPI_Wtime();
…
t2 = MPI_Wtime();
dt = t2 – t1;
```

Пример:

```
#include <stdio.h>
#include "mpi.h"
int main(int argc, char* argv[]){
int ProcNum, ProcRank, RecvRank;
MPI_Status Status;
MPI_Init(&argc, &argv);
MPI_Comm_size(MPI_COMM_WORLD, &ProcNum);
MPI_Comm_rank(MPI_COMM_WORLD, &ProcRank);
if ( ProcRank == 0 )
        { // Действия, выполняемые только процессом с рангом 0
        printf ("\n Hello from process %3d", ProcRank);
        for ( int i=1; i<ProcNum; i++ )
         { MPI_Recv(&RecvRank, 1, MPI_INT, MPI_ANY_SOURCE,
                MPI_ANY_TAG, MPI_COMM_WORLD, &Status);
                printf("\n Hello from process %3d", RecvRank); } }
else
// Сообщение, отправляемое всеми процессами, кроме процесса с рангом 0
        MPI_Send(&ProcRank,1,MPI_INT,0,0,MPI_COMM_WORLD);
        MPI_Finalize();
        return 0;
}
```

Определение размера полученного сообщения (count)

```
int MPI_Get_count(MPI_Status *status, MPI_Datatype datatype, int *count)
```

Аргумент datatype должен соответствовать типу данных, указанному в операции передачи сообщения.

Неблокирующая проверка завершения приема или передачи сообщения

int MPI_Test(MPI_Request *request, int *flag, MPI_Status *status)

Входной параметр:

request – идентификатор операции обмена.

Выходные параметры:

flag – "истина", если операция, заданная идентификатором request, выполнена;

status – статус выполненной операции.

Широковещательная рассылка

int MPI_Bcast(void *buffer, int count, MPI_Datatype datatype, int root, MPI_Comm comm)

Параметры этой процедуры одновременно являются входными и выходными:

buffer – адрес буфера;

count – количество элементов данных в сообщении;

datatype – тип данных MPI;

root – ранг главного процесса, выполняющего широковещательную рассылку;

comm – коммуникатор.

Пример коллективных сообщений – сумма элементов вектора:

```
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include "mpi.h"
int main(int argc, char* argv[])
{
double x[100], TotalSum, ProcSum = 0.0;
int ProcRank, ProcNum, N=100;
MPI_Status Status;
// инициализация
MPI_Init(&argc,&argv);
MPI_Comm_size(MPI_COMM_WORLD,&ProcNum);
MPI_Comm_rank(MPI_COMM_WORLD,&ProcRank);
// подготовка данных
if ( ProcRank == 0 ) DataInitialization(x,N);
// рассылка данных на все процессы
MPI_Bcast(x, N, MPI_DOUBLE, 0, MPI_COMM_WORLD);
// вычисление частичной суммы на каждом из процессов
int k = N / ProcNum;
int i1 = k * ProcRank;
int i2 = k * ( ProcRank + 1 );
if ( ProcRank == ProcNum-1 ) i2 = N;
        for ( int i = i1; i < i2; i++ )
                ProcSum = ProcSum + x[i];
// сборка частичных сумм на процессе с рангом 0
if ( ProcRank == 0 )
```

```
        {
        TotalSum = ProcSum;
            for ( int i=1; i < ProcNum; i++ ) {
            MPI_Recv(&ProcSum, 1, MPI_DOUBLE,
MPI_ANY_SOURCE, 0, MPI_COMM_WORLD, &Status);
        TotalSum = TotalSum + ProcSum;
            }
        }
    else   // все процессы отсылают свои частичные суммы
    MPI_Send(&ProcSum, 1, MPI_DOUBLE, 0, 0, MPI_COMM_WORLD);
    // вывод результата
    if ( ProcRank == 0 )
        printf("\nTotal Sum = %10.2f",TotalSum);
    MPI_Finalize();
    }
```

Синхронизация процессов

int MPI_Barrier(MPI_Comm comm)

При синхронизации с барьером выполнение каждого процесса из данного коммуникатора приостанавливается до тех пор, пока все процессы не выполнят вызов процедуры синхронизации MPI_Barrier.

Распределение данных

int MPI_Scatter(void *sendbuf, int sendcount, MPI_Datatype sendtype, void *rcvbuf, int rcvcount, MPI_Datatype rcvtype, int root, MPI_Comm comm)

Входные параметры:

sendbuf – адрес буфера передачи;

sendcount – количество элементов, пересылаемых каждому процессу;

sendtype – тип передаваемых данных;

rcvcount – количество элементов в буфере приема;

rcvtype – тип принимаемых данных;

root – ранг передающего процесса;

comm – коммуникатор.

Выходной параметр: rcvbuf – адрес буфера приема.

Процесс с рангом root распределяет содержимое буфера передачи sendbuf среди всех процессов. Содержимое буфера передачи разбивается на несколько фрагментов, каждый из которых содержит sendcount элементов. Первый фрагмент передается процессу 0, второй процессу 1 и т. д. Аргументы send имеют значение только на стороне процесса root.

Сбор сообщений от остальных процессов в буфер главной задачи

int MPI_Gather(void *sendbuf, int sendcount, MPI_Datatype sendtype, void *rcvbuf, int rcvcount, MPI_Datatype rcvtype, int root, MPI_Comm comm)

Каждый процесс в коммуникаторе comm пересылает содержимое буфера передачи sendbuf процессу с рангом root. Процесс root "склеивает" полученные данные в буфере приема.

Порядок склейки определяется рангами процессов.

Сбор данных от всех процессов и распределение их всем процессам

int MPI_Allgather(void *sendbuf, int sendcount, MPI_Datatype sendtype, void *rcvbuf, int rcvcount, MPI_Datatype rcvtype, MPI_Comm comm)

Входные параметры:

sendbuf – начальный адрес буфера передачи;

sendcount – количество элементов в буфере передачи;

sendtype – тип передаваемых данных;

rcvcount – количество элементов, полученных от каждого процесса;

rcvtype – тип данных в буфере приема;

comm – коммуникатор.

Выходной параметр: rcvbuf – адрес буфера приема.

Способы распределения итераций циклов

- ✓ Блочное распределение – по □N/P□ итераций.
- ✓ Циклическое распределение – циклически по одной итерации.
- ✓ Блочно-циклическое распределение.

Рассмотрим простейший цикл:

```
for(i=0; i<N; i++)
        {
        a[i] = a[i] + b[i];
        }
```

Блочное распределение для P процессоров

```
// размер блока итераций
        k = (N-1)/P + 1
// начало блока итераций процессора MyProc
        ibeg = MyProc * k + 1
// конец блока итераций процессора MyProc
        iend = (MyProc + 1) * k
// если не досталось итераций
        if (ibeg > N) {iend = ibeg – 1;}
```

```
// если досталось меньше итераций
        if (iend > N) {iend = N;}
for(i= ibeg;i< iend;i++)
        {a[i] = a[i] + b[i];}
```

Циклическое распределение для P процессоров

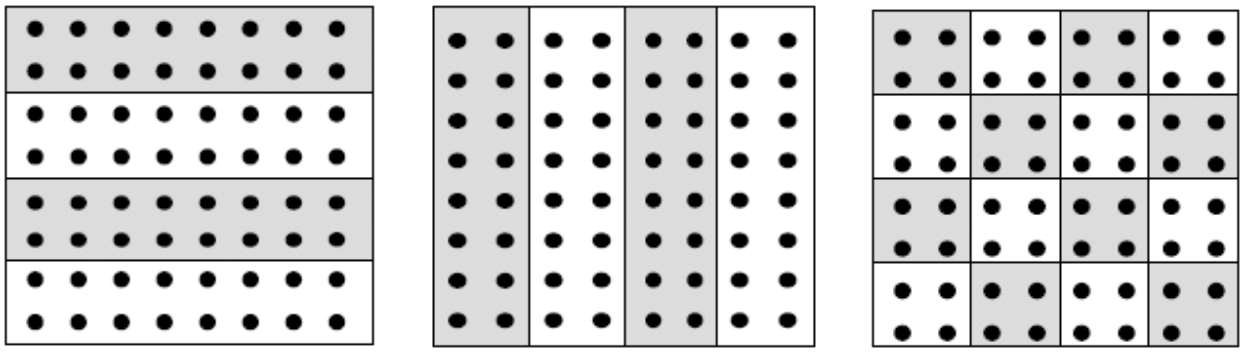
```
for(i= MyProc; i< N; i+=P)
        {
                a[i] = a[i] + b[i];
        }
```

Принципы распараллеливания – декомпозиция матричных вычислений

Для многих методов матричных вычислений характерным является повторение одних и тех же вычислительных действий для разных элементов матриц. Распараллеливание матричных операций сводится к разделению обрабатываемых матриц между процессорами используемой вычислительной системы. Наиболее широко используемые способы разделения матриц состоят в разбиении данных на полосы (по вертикали или горизонтали) или на прямоугольные фрагменты (блоки).

При ленточном разбиении каждому процессору (ядру) выделяется то или иное подмножество строк (горизонтальное разбиение) или столбцов (вертикальное разбиение) матрицы.

При блочном разделении матрица делится на прямоугольные наборы элементов.



a b c

Рисунок 1 Ленточное (а,б) и блочное (с) разделение матрицы

Виртуальные топологии

Кроме списка процессов и контекста обмена с коммуникатором может быть связана дополнительная информация. Важнейшей разновидностью такой информации является топология обменов. В MPI топология позволяет сопоставить процессам, принадлежащим некоторой группе, других, отличных от обычной, схем адресации. Топологии обменов сообщениями в MPI являются виртуальными. Это значит, что они не связаны с физической топологией коммуникационной сети параллельной вычислительной системы. Топологией в данном случае называют структуру соединений линий и узлов сети без учета характеристик самих узлов. Узлами здесь являются процессы, соединениями - каналы обмена сообщениями, а сетью мы считаем все процессы, входящие в состав параллельной программы. Часто в прикладных программах процессы естественно упорядочить в соответствии с логикой задачи. Такая ситуация возникает, например, если выполняются расчеты, в которых используются решетки (сетки). Это может быть при программировании сеточных методов решения дифференциальных уравнений, а также в других случаях. В MPI существуют два типа топологии:

• декартова топология – прямоугольная решетка произвольной размерности

• топология графа.

Над топологиями можно выполнять различные операции. Декартовы решетки можно расщеплять на гиперплоскости, удаляя некоторые измерения. Данные можно сдвигать вдоль выбранного измерения декартовой решетки. Сдвигом называют пересылку данных между процессами вдоль определенного измерения. Вдоль избранного измерения могут быть организованы коллективные обмены. Для того, чтобы связать структуру декартовой решетки с коммуникатором MPI_COMM_WORLD, необходимо задать следующие параметры:

- ✓ размерность решетки (значение 2 соответствует плоской, двумерной решетке);
- ✓ размер решетки вдоль каждого измерения (размеры {10, 15}, например, соответствуют плоской прямоугольной решетке, протяженность которой вдоль оси x составляет 10 узлов-процессов, а вдоль оси y ¾ 15 узлов);
- ✓ периодичность вдоль каждого измерения (решетка может быть периодической, если процессы, находящиеся на противоположных концах ряда, взаимодействуют между собой).

MPI дает возможность системе оптимизировать отображение виртуальной топологии процессов на физическую с помощью изменения порядка нумерации процессов в группе. Подпрограмма MPI_Cart_create (описания интерфейсов соответствующих подпрограмм имеются в Методическом пособии) создает новый коммуникатор, наделяя декартовой топологией исходный коммуникатор. MPI_Cart_create является коллективной операцией (эту подпрограмму должны вызывать все процессы из коммуникатора, наделяемого декартовой топологией). После создания виртуальной топологии можно использовать соответствующую схему адресации процессов, но для этого требуется пересчет ранга процесса в его декартовы координаты и наоборот. Определить декартовы координаты процесса по его рангу в группе можно с помощью подпрограммы MPI_Cart_coords.

Обратным действием по отношению к MPI_Cart_coords обладает подпрограмма MPI_Cart_rank. С ее помощью можно определить ранг процесса по его декартовым координатам в соответствующем коммуникаторе. Между процессами, организованными в декартову решетку, могут выполняться обмены особого вида. Это сдвиги, о которых мы уже упоминали. Имеются два типа сдвигов данных по группе из N процессов:

- ✓ циклический сдвиг на J позиций вдоль ребра решетки. Данные от процесса K пересылаются процессу с номером (J + K) mod N;
- ✓ линейный сдвиг на J позиций вдоль ребра решетки, когда данные в процессе K пересылаются процессу с номером J + K, если ранг адресата находится в пределах между 0 и N.

Задания

1. В исходном тексте программы на языке С пропущены вызовы процедур широковещательной рассылки. Добавить эти вызовы, откомпилировать и запустить программу.

```
#include "mpi.h"
#include <stdio.h>
int main(int argc,char *argv[])
{
char data[24];
int myrank, count = 25;
MPI_Status status;
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
if (myrank == 0)
{
strcpy(data, "Hi, Parallel Programmer!");
…
printf("send: %s\n", data);
}
else
{
…
```

```
printf("received: %s\n", data);
}
MPI_Finalize();
return 0;
}
```

2. В программе на языке С предполагается, что три численных значения, введенных с клавиатуры, пересылаются широковещательной рассылкой всем прочим процессам. Вызовы подпрограмм широковещательной рассылки пропущены. Добавить эти вызовы, откомпилировать и запустить программу.

```
#include "mpi.h"
#include <stdio.h>
int main(int argc, char *argv[])
{
int myrank;
int root = 0;
int count = 1;
float a, b;
int n;
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
if (myrank == 0)
{
printf("Enter a, b, n\n");
scanf("%f %f %i", &a, &b, &n);
…
}
else
{
…
printf("%i Process got %f %f %i\n", myrank, a, b, n);
}
```

```
MPI_Finalize();
return 0;
}
```

3. В программе на языке С создается новый коммуникатор, а затем сообщения между процессами, входящими в него, пересылаются широковещательной рассылкой. Вызовы подпрограмм создания новой группы процессов (на 1 меньше, чем полное количество запущенных на выполнение процессов) и нового коммуникатора пропущены. Добавить эти вызовы, откомпилировать и запустить программу.

```
#include "mpi.h"
#include <stdio.h>
int main(int argc,char *argv[])
{
char message[24];
MPI_Group MPI_GROUP_WORLD;
MPI_Group group;
MPI_Comm fcomm;
int size, q, proc;
int* process_ranks;
int rank, rank_in_group;
MPI_Status status;
MPI_Init(&argc, &argv);
MPI_Comm_size(MPI_COMM_WORLD, &size);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
printf("New group contains processes:");
q = size - 1;
process_ranks = (int*) malloc(q*sizeof(int));
for (proc = 0; proc < q; proc++)
{
process_ranks[proc] = proc;
printf("%i ", process_ranks[proc]);
}
printf("\n");
…
if (fcomm != MPI_COMM_NULL) {
MPI_Comm_group(group, &fcomm);
MPI_Comm_rank(fcomm, &rank_in_group);
if (rank_in_group == 0) {
```

```
strcpy(message, "Hi, Parallel Programmer!");
MPI_Bcast(&message, 25, MPI_BYTE, 0, fcomm);
printf("0 send: %s\n", message);
}
else
{
MPI_Bcast(&message, 25, MPI_BYTE, 0, fcomm);
printf("%i received: %s\n", rank_in_group, message);
}
MPI_Comm_free(&fcomm);
MPI_Group_free(&group);
}
MPI_Finalize();
return 0;}
```

4. В программе на языке С сначала создается подгруппа, состоящая из процессов с рангами 1, 3, 5 и 7, и соответствующий ей коммуникатор. Затем выполняется редукция (суммирование) по процессам, входящим в новую группу. Вызов подпрограммы редукции и некоторые другие важные фрагменты пропущены. Добавить эти вызовы, откомпилировать и запустить программу.

```
#include "mpi.h"
#include <stdio.h>
int main(int argc,char *argv[])
{
int myrank, i;
int count = 5, root = 1;
MPI_Group MPI_GROUP_WORLD, subgroup;
int ranks[4] = {1, 3, 5, 7};
MPI_Comm subcomm;
int sendbuf[5] = {1, 2, 3, 4, 5};
int recvbuf[5];
MPI_Init(&argc, &argv);
MPI_Comm_group(MPI_COMM_WORLD,
&MPI_GROUP_WORLD);
MPI_Group_incl(MPI_GROUP_WORLD, 4, ranks, &subgroup);
MPI_Group_rank(subgroup, &myrank);
…
if(myrank != MPI_UNDEFINED)
{
```

```
MPI_Reduce(&sendbuf, &recvbuf, count, MPI_INT, MPI_SUM,
root,
subcomm);
if(myrank == root) {
printf("Reduced values");
for(i = 0; i < count; i++){
printf(" %i ", recvbuf[i]);}
}
printf("\n");
MPI_Comm_free(&subcomm);
MPI_Group_free(&MPI_GROUP_WORLD);
…
}
MPI_Finalize();
return 0;
}
```

5. В программе на языке С коммуникатор grid_comm наделяется топологией двумерной решетки с периодическими граничными условиями, причем системе разрешено изменить порядок нумерации процессов. В исходном тексте пропущены вызовы процедур, с помощью которых каждый процесс может определить свой ранг и декартовы координаты. Добавьте эти вызовы, откомпилируйте и запустите программу.

```
#include "mpi.h"
#include <stdio.h>
int main(int argc,char *argv[])
{
MPI_Comm grid_comm;
int dims[2];
int periodic[2];
int reorder = 1, q = 5, ndims = 2, maxdims = 2;
int coordinates[2];
int my_grid_rank;
int coords[2];
MPI_Comm row_comm;
dims[0] = dims[1] = q;
periodic[0] = periodic[1] = 1;
coords[0] = 0; coords[1] = 1;
MPI_Init(&argc, &argv);
MPI_Cart_create(MPI_COMM_WORLD, ndims, dims, periodic,
reorder,
```

```
&grid_comm);
printf("Process rank %i has coordinates %i %i\n", my_grid_rank,
coordinates[0], coordinates[1]);
MPI_Finalize();
return 0;
}
```

6. Добавьте в программу работы с коммуникатором операции циклического и/или линейного сдвига вдоль определенного измерения решетки:

```
#include "mpi.h"
#include <stdio.h>
int main(int argc,char *argv[])
{
MPI_Comm grid_comm;
int dims[2];
int periodic[2];
int reorder = 1, q = 5, ndims = 2, maxdims = 2;
int coordinates[2];
int my_grid_rank;
int coords[2];
MPI_Comm row_comm;
dims[0] = dims[1] = q;
periodic[0] = periodic[1] = 1;
coords[0] = 0; coords[1] = 1;
MPI_Init(&argc, &argv);
MPI_Cart_create(MPI_COMM_WORLD, ndims, dims, periodic, reor-
der,
&grid_comm);
printf("Process rank %i has coordinates %i %i\n", my_grid_rank,
coordinates[0], coordinates[1]);
MPI_Finalize();
return 0;
}
```

7. Измените следующую программу, наделив коммуникатор топологией графа вместо топологии решетки.

```
#include "mpi.h"
#include <stdio.h>
int main(int argc,char *argv[])
{
MPI_Comm grid_comm;
```

```
int dims[2];
int periodic[2];
int reorder = 1, q = 5, ndims = 2, maxdims = 2;
int coordinates[2];
int my_grid_rank;
int coords[2];
MPI_Comm row_comm;
dims[0] = dims[1] = q;
periodic[0] = periodic[1] = 1;
coords[0] = 0; coords[1] = 1;
MPI_Init(&argc, &argv);
MPI_Cart_create(MPI_COMM_WORLD, ndims, dims, periodic, reor-
der,
&grid_comm);
printf("Process rank %i has coordinates %i %i\n", my_grid_rank,
coordinates[0], coordinates[1]);
MPI_Finalize();
return 0;
}
```

8. С использованием библиотеки MPI написать программу вычисления скалярного произведения двух векторов. Предусмотреть оценку времени решения задачи средствами библиотеки MPI. Оценить ускорение параллельной программы относительно последовательной.
9. С использованием библиотеки MPI написать программу вычисления среднего значения для двумерного массива числовых данных. Предусмотреть оценку времени решения задачи средствами библиотеки MPI. Оценить ускорение параллельной программы относительно последовательной.
10. С использованием библиотеки MPI написать программу поиска максимального значения заданного двумерного массива числовых данных. Предусмотреть оценку времени решения задачи средствами библиотеки MPI. Оценить ускорение параллельной программы относительно последовательной.

11. С использованием библиотеки MPI написать программу поиска минимального значения вектора. Предусмотреть оценку времени решения задачи средствами библиотеки MPI. Оценить ускорение параллельной программы относительно последовательной.
12. С использованием библиотеки MPI написать программу умножения матрицы на вектор. Предусмотреть оценку времени решения задачи средствами библиотеки MPI. Оценить ускорение параллельной программы относительно последовательной.
13. С использованием библиотеки MPI написать программу вычисления среднего значения для трехмерного массива числовых данных. Предусмотреть оценку времени решения задачи средствами библиотеки MPI. Оценить ускорение параллельной программы относительно последовательной.
14. С использованием библиотеки MPI написать программу вычисления среднего значения для двумерного массива числовых данных при циклическом способе распределения итераций циклов. Предусмотреть оценку времени решения задачи средствами библиотеки MPI. Оценить ускорение параллельной программы относительно последовательной.
15. С использованием библиотеки MPI написать программу вычисления определенного интеграла методом средних прямоугольников. Предусмотреть оценку времени решения задачи средствами библиотеки MPI. Оценить ускорение параллельной программы относительно последовательной.
16. С использованием библиотеки MPI написать программу вычисления определенного интеграла методом трапеций. Предусмотреть оценку времени решения задачи средствами библиотеки MPI. Оценить ускорение параллельной программы относительно последовательной.
17. С использованием библиотеки MPI написать программу вычисления определенного интеграла методом Симпсона. Предусмотреть оценку

времени решения задачи средствами библиотеки MPI. Оценить ускорение параллельной программы относительно последовательной.

18. С использованием библиотеки MPI написать программу вычисления матрицы C как функции от заданных матриц A и B, заполненных случайными числами: $C = A^2+B^2$. Предусмотреть оценку времени решения задачи средствами библиотеки MPI. Оценить ускорение параллельной программы относительно последовательной.
19. С использованием библиотеки MPI написать программу вычисления матрицы C как функции от заданных матриц A и B: $C = (A – B)^2$. Предусмотреть оценку времени решения задачи средствами библиотеки MPI. Оценить ускорение параллельной программы относительно последовательной.
20. С использованием библиотеки MPI написать программу вычисления матрицы C как функции от заданных матриц A и B, заполненных случайными числами: $C = (B–A)+A^2$. Предусмотреть оценку времени решения задачи средствами библиотеки MPI. Оценить ускорение параллельной программы относительно последовательной.
21. С использованием библиотеки MPI написать программу вычисления матрицы C как функции от заданных матриц A и B, заполненных случайными числами: $C = (A\cdot 5 – B\cdot 8) + A\cdot B$. Предусмотреть оценку времени решения задачи средствами библиотеки MPI. Оценить ускорение параллельной программы относительно последовательной.
22. С использованием библиотеки MPI написать программу вычисления матрицы C как функции от заданных матриц A и B, заполненных случайными числами: $C = A^2 – B^2$. Предусмотреть оценку времени решения задачи средствами библиотеки MPI. Оценить ускорение параллельной программы относительно последовательной.
23. С использованием библиотеки MPI написать программу вычисления матрицы C как функции от заданных матриц A и B, заполненных случай-

ными числами: C = (A·5 – B·8) + A·B. Предусмотреть оценку времени решения задачи средствами библиотеки MPI. Оценить ускорение параллельной программы относительно последовательной.

24. С использованием библиотеки MPI написать программу вычисления матрицы C как функции от заданных матриц A и B, заполненных случайными числами: C = A·B – 11. Предусмотреть оценку времени решения задачи средствами библиотеки MPI. Оценить ускорение параллельной программы относительно последовательной.
25. С использованием библиотеки MPI написать программу вычисления матрицы C как функции от заданных матриц A и B, заполненных случайными числами: $C = A \cdot B + A^2$. Предусмотреть оценку времени решения задачи средствами библиотеки MPI. Оценить ускорение параллельной программы относительно последовательной.
26. С использованием библиотеки MPI написать программу вычисления матрицы C как функции от заданных матриц $C = (B–A)^2–B^2$. Предусмотреть оценку времени решения задачи средствами библиотеки MPI. Оценить ускорение параллельной программы относительно последовательной.
27. С использованием библиотеки MPI написать программу вычисления матрицы C как функции от заданных матриц A и B, заполненных случайными числами: C = A·B–B+3. Предусмотреть оценку времени решения задачи средствами библиотеки MPI. Оценить ускорение параллельной программы относительно последовательной.
28. С использованием библиотеки MPI написать программу вычисления матрицы C как функции от заданных матриц A и B, заполненных случайными числами: $C = B+3 \cdot A–B^2$. Предусмотреть оценку времени решения задачи средствами библиотеки MPI. Оценить ускорение параллельной программы относительно последовательной.

29. С использованием библиотеки MPI написать программу вычисления матрицы C как функции от заданных матриц A и B, заполненных случайными числами: $C = (A+B)^2$. Предусмотреть оценку времени решения задачи средствами библиотеки MPI. Оценить ускорение параллельной программы относительно последовательной.
30. С использованием библиотеки MPI написать программу, реализующую дискретное преобразование Фурье сигнала (прямое и обратное), представление результата в виде амплитуд и фаз гармоник, сигнал – гармоника с заданной амплитудой и фазой. Предусмотреть оценку времени решения задачи средствами библиотеки MPI. Оценить ускорение параллельной программы относительно последовательной.
31. С использованием библиотеки MPI написать программу, реализующую цифровой полосовой фильтр, сигнал – сумма гармоник и шума. Предусмотреть оценку времени решения задачи средствами библиотеки MPI. Оценить ускорение параллельной программы относительно последовательной.
32. С использованием библиотеки MPI написать программу, реализующую вейвлет преобразование сигнала с материнской функцией Морле. Предусмотреть оценку времени решения задачи средствами библиотеки MPI. Оценить ускорение параллельной программы относительно последовательной.
33. С использованием библиотеки MPI написать программу, реализующую вейвлет преобразование сигнала с материнской функцией Гаусса 1-го порядка. Предусмотреть оценку времени решения задачи средствами библиотеки MPI. Оценить ускорение параллельной программы относительно последовательной.
34. С использованием библиотеки MPI написать программу, реализующую вейвлет преобразование сигнала с материнской функцией МНАТ. Предусмотреть оценку времени решения задачи средствами библиотеки MPI.

Оценить ускорение параллельной программы относительно последовательной.

35. С использованием библиотеки MPI написать программу, реализующую вейвлет преобразование сигнала с материнской функцией Хаара. Предусмотреть оценку времени решения задачи средствами библиотеки MPI. Оценить ускорение параллельной программы относительно последовательной.
36. С использованием библиотеки MPI написать программу, реализующую вейвлет преобразование сигнала с материнской функцией Гаусса 4-го порядка. Предусмотреть оценку времени решения задачи средствами библиотеки MPI. Оценить ускорение параллельной программы относительно последовательной.
37. С использованием библиотеки MPI написать программу, реализующую вейвлет преобразование сигнала с материнской функцией Добеши. Предусмотреть оценку времени решения задачи средствами библиотеки MPI. Оценить ускорение параллельной программы относительно последовательной.
38. С использованием библиотеки MPI написать программу, реализующую вейвлет преобразование сигнала с материнской функцией Пауля. Предусмотреть оценку времени решения задачи средствами библиотеки MPI. Оценить ускорение параллельной программы относительно последовательной.

Литература

1. Антонов, А. С. Параллельное программирование с использованием технологии MPI : учебное пособие / А. С. Антонов. – 2-е изд. – Москва : ИНТУИТ, 2016. – 83 с. – Текст : электронный // Лань : электронно-библиотечная система. – URL: https://e.lanbook.com/book/100359 (дата обращения: 14.07.2022). – Режим доступа: для авториз. пользователей.
2. Воеводин, В. В. Вычислительная математика и структура алгоритмов : учебное пособие / В. В. Воеводин. – 2-е изд. – Москва : ИНТУИТ, 2016. – 145 с. – Текст : электронный // Лань : электронно-библиотечная система. – URL: https://e.lanbook.com/book/100738 (дата обращения: 14.07.2022). – Режим доступа: для авториз. пользователей.
3. Гергель В.П., Стронгин, Р.Г. Основы параллельных вычислений для многопроцессорных вычислительных систем. Учебное пособие – Нижний Новгород: Изд-во ННГУ им. Н.И. Лобачевского, 2003. – 184 с.
4. Тептин Г. М., Хуторова О. Г, Зинин Д. П. Введение в современные высокопроизводительные вычислительные системы: учебно-методическое пособие. Казань: КФУ, 2015. – 45 с.