Федеральное агенство по образованию

Сибирский федеральный университет

Карепова Е.Д., Кузьмин Д.А., Легалов А.И.,

Редькин А.В., Удалова Ю.В., Федоров Г.А.

СРЕДСТВА РАЗРАБОТКИ ПАРАЛЛЕЛЬНЫХ ПРОГРАММ

Учебное пособие по циклу лабораторных работ

Красноярск 2007

УДК 681.3.06
ББК

Рецензенты:

Учебное пособие содержит описание 12 лабораторных работ по курсу «Средства разработки параллельных программ», направленных на освоение основных приемов многопоточного и параллельного программирования. В ходе выполнения лабораторных работ студенты получат практические навыки программирования с использованием библиотеки потоков Pthread, предоставляющей API для создания и управления потоками в приложении, изучат основные функции WinAPI, используемые для создания и управления потоками в ОС Windows, освоят технологию OpenMP, как стандарт программирования для многопроцессорных ВС с общей памятью, получат практические навыки программирования с использованием библиотеки передачи сообщений MPI для вычислительных систем с распределенной памятью.

Предназначено для магистрантов группы 230100.68 «Информатика и вычислительная техника», обучающихся по направлению 230100.68.02 «Высокопроизводительные вычислительные системы», а также может быть полезно для инженеров и магистров, работающих в области высокопроизводительных параллельных вычислений.

ОГЛАВЛЕНИЕ

Введение 4
Лабораторная работа №1
Анатомия простого многопоточного приложения 5
Лабораторная работа №2
Практические приемы построения многопоточных приложений 13
Лабораторная работа №3
Механизмы синхронизации 23
Лабораторная работа №4
Блокировки чтения-записи 39
Лабораторная работа №5
Барьеры 47
Лабораторная работа №6
Создание простого приложения в среде OpenMP 52
Лабораторная работа №7
Отладка OpenMP –приложения: поиск ошибок, оптимизация 57
Лабораторная работа №8
Работа на кластере рабочих станций, работа в среде МВС-1000 67
Лабораторная работа №9
Создание простого приложения с помощью библиотеки MPI 71
Лабораторная работа №10 – 11
Коллективные обмены MPI. Отладка параллельных программ 76
Лабораторная работа №12
Решение прикладных задач с помощью MPI 81

Введение

Дисциплина «Средства разработки параллельных программ» предназначена для изучения средств и методов создания приложений для различных архитектур ВС. Основное внимание при изучении дисциплины уделяется получению практических навыков написания параллельных программ в терминах конкретных библиотек и/или языковых реализаций для вычислительных систем, как с общей, так и распределенной памятью (в том числе, многоядерных и кластерных архитектур).

Учебное пособие по циклу лабораторных работ содержит описание 12 лабораторных работ по курсу «Средства разработки параллельных программ», направленных на освоение основных приемов многопоточного и параллельного программирования.

Выполнение лабораторных работ 1 – 5 помогает приобрести практические навыки программирования с использованием библиотеки потоков Pthread, предоставляющей API для создания и управления потоками в приложении, а также изучить основные функции WinAPI, используемые для создания и управления потоками в ОС Windows. Многие алгоритмы, рассмотренные в этих лабораторных работах, предлагается реализовать двумя способами – с помощью библиотеки потоков Pthread и функций WinAPI, что позволяет сравнить различные реализации одних и тех же механизмов.

Лабораторные работы 6 – 7 направлены на освоение технологии OpenMP, как стандарта программирования для многопроцессорных ВС с общей памятью. В ходе выполнения этих работ студенты также изучат базовые методы и средства отладки многопоточных программ, в том числе и написанных с применением OpenMP с использованием отладчика Intel Thread Checker.

В лабораторных работах 8 – 12 рассматриваются проблемы параллельного программирования для вычислительных систем с распределенной памятью на примере библиотеки передачи сообщений MPI. Особое внимание в этих лабораторных работах уделяется разработке параллельных программ для задач линейной алгебры, вычислительной математики, обработки изображений и комбинаторики.

ЛАБОРАТОРНАЯ РАБОТА №1

Анатомия простого многопоточного приложения

Цели и задачи: Научиться создавать простые многопоточные приложения с помощью библиотеки Pthread.

Время: 2 часа

Потоки предоставляют возможность проведения параллельных или псевдопараллельных, в случае одного процессора, вычислений. Потоки могут порождаться во время работы программы, процесса или другого потока. Основное отличие потоков от процессов заключается в том, что различные потоки имеют различные пути выполнения, но при этом пользуются общей памятью. Путь выполнения потока задается при его создании, указанием его стартовой функции, созданный поток начинает выполнять команды этой функции, и завершается когда происходит возврат из функции.

Порядок выполнения лабораторной работы

1. Разработать алгоритм решения задания, с учетом разделения вычислений между несколькими потоками. Избегать ситуаций изменения одних и тех же общих данных несколькими потоками. Составить схему потоков.

2. Реализовать алгоритм с применением функций библиотеки Pthread и протестировать его на нескольких примерах.

3. *Самостоятельная работа.* Реализовать алгоритм с применением функций WinAPI. Сравнить возможности обоих подходов.

Пример №1

Дана последовательность натуральных чисел $a_0, \ldots, a_{99}$. Создать многопоточное приложение для поиска суммы квадратов $\sum a_i^2$. В приложении вычисления должны независимо выполнять четыре потока.

Обсуждение. Разобьем последовательность чисел на четыре части и создадим четыре потока, каждый из которых будет вычислять суммы квадратов элементов в отдельной части последовательности. Главный поток создаст дочерние потоки, соберет данные и вычислит окончательный результат, после того, как отработают четыре дочерних потока (рис. 1.1).

Реализуем задачу, используя библиотеку PTHREAD.

```
#include <pthread.h>
#include <stdio.h>
int A[100] ;  //последовательность чисел a0…a99

//стартовая функция для дочерних потоков
void *func(void *param)
{
   //вычисление суммы квадратов
```

```
    int i, sum = 0, p = 25*(int)param ;
    for(i=p ; i<p+25 ; i++) sum+=A[i]*A[i] ;
    return (void *)sum ;
}
int main()
{
    int rez=0 ; //для записи окончательного результата
    // счиитывание или заполнение массива A
    pthread_t thread[4] ;
    //создание четырех дочерних потоков
    for (int i=0 ; i<4 ; i++) pthread_create(&thread[i], NULL, func, (void *)i) ;
    for (int i=0 ; i<4 ; i++)
    {
        int sum ; //ожидание завершения работы дочерних потоков
        pthread_join(thread[i],(void **)&sum) ;  // и получение результата из вычислений
        rez += sum ;
    }
    fprintf(stdout,"\nСумма квадратов = %d",rez) ; // вывод результата
    return 1;
}
```

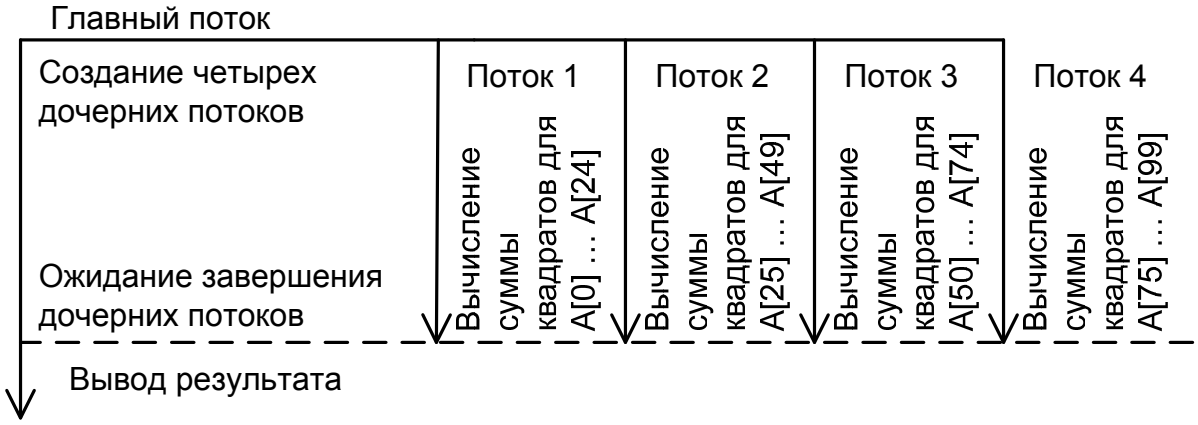

Рис. 1.1. Схема потоков для примера 1

Реализация этого же алгоритма с применением функций WinAPI приведена ниже.

```
#include <stdio.h>
#include <conio.h>
#include <windows.h>
const int n = 4;
int a[100];
DWORD WINAPI ThreadFunc(PVOID pvParam)
{
    int num,sum = 0,i;
    num = 25*(*((int *)pvParam));
```

```
    for(i=num;i<num+25;i++) sum += a[i]*a[i];
    *(int*)pvParam = sum;

    DWORD dwResult = 0;
    return dwResult;
}

int main(int argc, char** argv)
{
    int x[n];
    int i,rez = 0;

    DWORD dwThreadId[n],dw;
    HANDLE hThread[n];

    for (i=0;i<100;i++) a[i] = i;

    //создание n дочерних потоков
    for (i=0;i<n;i++)
    {
        x[i] = i;
        hThread[i] = CreateThread(NULL,0,ThreadFunc,(PVOID)&x[i], 0, &dwThreadId[i]);
        if(!hThread) printf("main process: thread %d not execute!",i);
    }

    // ожидание завершения n потоков
    dw = WaitForMultipleObjects(n,hThread,TRUE,INFINITE);

    for(i=0;i<n;i++) rez+=x[i];
    printf("\nСумма квадратов = %d",rez);

    getch();
    return 0;
}
```

Пример №2

Дана последовательность натуральных чисел $a_0 \ldots a_{99}$. Создать многопоточное приложение для поиска суммы квадратов $\sum a_i^2$. В приложении вычисления должны выполнять два потока, работающие последовательно: первый поток вычисляет квадраты чисел, второй – находит их сумму

Обсуждение. В приложении главный поток создаст первый дочерний поток, который отработав, создаст второй поток, после окончания работы которого результат передается первому потоку, который в свою очередь передает его главному потоку. (рис. 1.2).

```
#include <pthread.h>
#include <stdio.h>
int A[100] ; //последовательность чисел a0…a99

//стартовая функция для второго потока
void *summa(void *param)
{
    int sum=0 ;
    for(int i=0 ; i<100 ; i++) sum+=A[i] ; //вычисление суммы
    return (void *)sum ;
```

```
}


//стартовая функция для первого потока
void *kvadrat(void *param)
{   int rez ;
    for(int i=0 ; i<100 ; i++) A[i]*=A[i] ; //вычисление квадратов
    pthread_t threadS ; //создание второго потока
    pthread_create(&threadS, NULL, summa, NULL) ;
    //ждем завершения второго потока и передаем его результат главному потоку
    pthread_join(threadS,(void **)&rez) ;
    return (void *)rez ;
}


int main()
{
    int rez=0 ;
    for (int i=0 ; i<100 ; i++) A[i] = 5 ;    // заполнение массива A
    pthread_t threadK ; //создание первого потока
    pthread_create(&threadK, NULL, kvadrat, NULL) ;
    pthread_join(threadK,(void **)&rez) ; //ожидание завершения первого потока
    fprintf(stdout,"\nСумма квадратов = %d",rez) ;
    return 1;
}
```

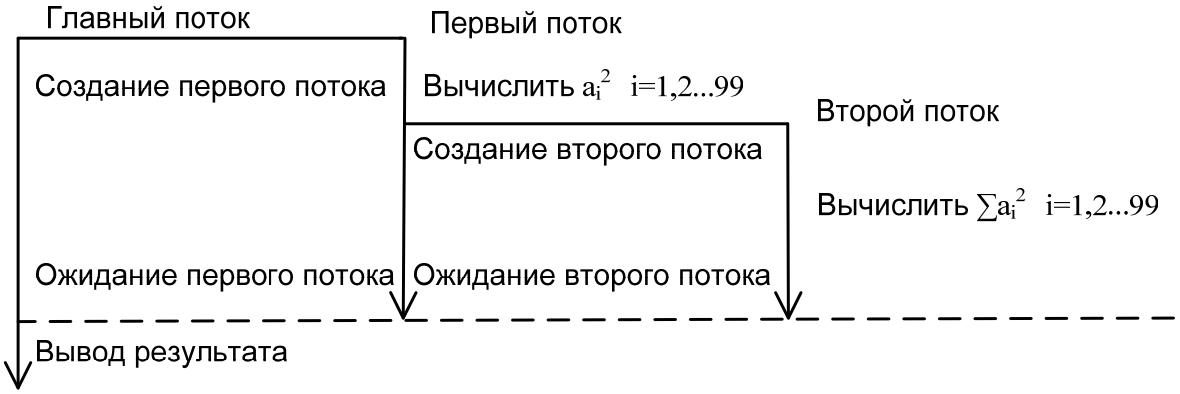

Рисунок 1.2. Схема потоков для примера 2

Реализация этого же алгоритма с применением функций WinAPI приведена ниже.

```
#include <stdio.h>
#include <conio.h>
#include <windows.h>

int a[100];
```

```
// вычисляет сумму элементов массива
DWORD WINAPI ThreadFunc2(PVOID pvParam)
{
    int sum = 0,i;

    for(i=0;i<100;i++)  sum += a[i];

    *(int*)pvParam = sum;
    DWORD dwResult = 0;
    return dwResult;
}
// вычисляет квадрат каждого элемента массива
DWORD WINAPI ThreadFunc1(PVOID pvParam)
{
    HANDLE hThread;
    DWORD dwThreadId;

    int rez = 0,i;

    for(i=0;i<100;i++) a[i] = a[i]*a[i];

    // создание потока ThreadFunc2
    hThread = CreateThread(NULL,0,ThreadFunc2,(PVOID)&rez, 0, &dwThreadId);
    if(!hThread) printf("ThreadFunc1 process: thread not execute!");

    // ожидание завершения потока
    WaitForSingleObject(hThread,INFINITE);

    *(int*)pvParam = rez;
    DWORD dwResult = 0;
    return dwResult;
}
int main(int argc, char** argv)
{
    DWORD dwThreadId;
    HANDLE hThread;
    int i,rez = 0;

    for (i=0;i<100;i++) a[i] = i;

    // создание потока ThreadFunc1
    hThread = CreateThread(NULL,0,ThreadFunc1,(PVOID)&rez, 0, &dwThreadId);
    if(!hThread) printf("main process: thread not execute!");

    // ожидание завершения потока
    WaitForSingleObject(hThread,INFINITE);

    printf("\nСумма квадратов = %d",rez);
    getch();
    return 0;
}
```

Варианты заданий

1. Даны последовательности символов A = $\{a_0 \ldots a_{n-1}\}$ и C = $\{c_0 \ldots c_{k-1}\}$. В общем случае $n \neq k$. Создать многопоточное приложение, определяющее, совпадают ли посимвольно строки A и C. Количество потоков является вход-

ным параметром программы, количество символов в строках может быть не кратно количеству потоков.

2. Дана последовательность символов C = $\{c_0 \ldots c_{n-1}\}$. Дан набор из N пар кодирующих символов (a_i,b_i). Создать многопоточное приложение, кодирующее строку C следующим образом: поток 0 заменяет в строке C все символы a_0 на символы b_0, поток 1 заменяет в строке C все символы a_1 на символы b_1, и т.д. Потоки должны осуществлять кодирование последовательно.

3. Дана последовательность символов C = $\{c_0 \ldots c_{n-1}\}$. Дан набор из N пар кодирующих символов (a_i,b_i), т.е. все символы строки a_i заменяются на b_i. Создать многопоточное приложение, кодирующее строку C следующим образом: строка разделяется на подстроки и каждый поток осуществляет кодирование своей подстроки. Количество символов с последовательности, количество кодирующих пар и потоков являются входными параметрами программы, количество символов в строке может быть не кратно количеству потоков.

4. Дана последовательность символов C = $\{c_0 \ldots c_{n-1}\}$ и символ b. Создать многопоточное приложение для определения количество вхождений символа b в строку C. Количество потоков является входным параметром программы, количество символов в строке может быть не кратно количеству потоков.

5. Дана последовательность натуральных чисел $\{a_0 \ldots a_{n-1}\}$. Создать многопоточное приложение для поиска суммы $\sum a_i$. Количество потоков является входным параметром программы, потоки проводят вычисления независимо друг от друга, количество символов в строке может быть не кратно количеству потоков.

6. Дана последовательность натуральных чисел $\{a_0 \ldots a_{n-1}\}$. Создать многопоточное приложение для поиска произведения чисел $a_0 * a_1 * \ldots * a_{n-1}$. Количество потоков является входным параметром программы, потоки проводят вычисления независимо друг от друга, количество символов в строке может быть не кратно количеству потоков.

7. Дана последовательность натуральных чисел $\{a_0 \ldots a_{n-1}\}$. Создать многопоточное приложение для поиска максимального a_i. Количество потоков является входным параметром программы, потоки проводят вычисления независимо друг от друга, количество символов в строке может быть не кратно количеству потоков.

8. Дана последовательность натуральных чисел $\{a_0 \ldots a_{n-1}\}$. Создать многопоточное приложение для поиска минимального a_i. Количество потоков является входным параметром программы, потоки проводят вычисления независимо друг от друга, количество символов в строке может быть не кратно количеству потоков.

9. Дана последовательность натуральных чисел $\{a_0...a_{n-1}\}$. Создать многопоточное приложение для поиска всех a_i, являющихся простыми числами. Количество потоков является входным параметром программы, потоки проводят вычисления независимо друг от друга, количество символов в строке может быть не кратно количеству потоков.

10. Дана последовательность натуральных чисел $\{a_0...a_{n-1}\}$. Создать многопоточное приложение для поиска всех a_i, являющихся квадратами, любого натурального числа.

11. Дана последовательность натуральных чисел $\{a_0...a_{n-1}\}$. Создать многопоточное приложение для вычисления выражения a_0-a_1+a_2-a_3+a_4-a_5+...

12. Дана последовательность натуральных чисел $\{a_0...a_{n-1}\}$. Создать многопоточное приложение для поиска суммы $\sum a_i$, где a_i – четные числа.

13. Изготовление знаменитого самурайского меча – катаны происходит в три этапа. Сначала младший ученик мастера выковывает заготовку будущего меча. Затем старший ученик мастера закаливает меч в трех водах – кипящей, студеной и теплой. И в конце мастер собственноручно изготавливает рукоять меча и наносит узоры. Требуется создать многопоточное приложение, в котором мастер и его ученики представлены разными потоками. Изготовление меча представить в виде разных арифметических операций над глобальной переменной.

14. Изготовление знаменитого самурайского меча – катаны происходит в три этапа. Сначала младший ученик мастера выковывает заготовку будущего меча. Затем старший ученик мастера закаливает меч в трех водах – кипящей, студеной и теплой. И в конце мастер собственноручно изготавливает рукоять меча и наносит узоры. Требуется создать многопоточное приложение, в котором мастер и его ученики представлены одинаковыми потоками (обработка производится в цикле). Изготовление меча представить в виде разных арифметических операций над глобальной переменной.

15. Командиру N-ской ВЧ полковнику Кузнецову требуется перемножить два секретных числа. Полковник Кузнецов вызывает дежурного по части лейтенанта Смирнова и требует в течение получаса предоставить ему ответ. Лейтенант Смирнов будит старшего по караулу сержанта Петрова и приказывает ему в 15 минут предоставить ответ. Сержант Петров вызывает к себе рядового Иванова, бывшего студента СФУ, и поручает ему ответственное задание по определению произведения. Рядовой Иванов успешно справляется с поставленной задачей и ответ по цепочке передается полковнику Кузнецову. Требуется создать многопоточное приложение, в котором все военнослужащие от полковника до рядового моделируются потоками одного вида.

16. Командиру N-ской ВЧ полковнику Кузнецову требуется перемножить два секретных числа. Полковник Кузнецов вызывает дежурного по час-

ти лейтенанта Смирнова и требует в течение получаса предоставить ему ответ. Лейтенант Смирнов будит старшего по караулу сержанта Петрова и приказывает ему в 15 минут предоставить ему ответ. Сержант Петров вызывает к себе рядового Иванова, бывшего студента СФУ, и поручает ему ответственное задание по определению произведения. Рядовой Иванов успешно справляется с поставленной задачей и ответ по цепочке передается полковнику Кузнецову. Требуется создать многопоточное приложение, с главным потоком – полковником Кузницовым, и двумя видами дочерних потоков (рядовой и командиры).

17. Даны результаты сдачи экзамена по курсу «Средства разработки параллельных программ» по студенческим группам. Требуется создать многопоточное приложение, вычисляющее средний балл. Потоки должны осуществлять вычисления параллельно по группам. Количество потоков является входным параметром программы, потоки проводят вычисления независимо друг от друга, количество групп может быть не кратно количеству потоков.

18. Охранное агентство разработало новую систему управления электронными замками. Для открытия двери клиент обязан произнести произвольную фразу из 25 слов. В этой фразе должно встречаться заранее оговоренное слово, причем только один раз. Требуется создать многопоточное приложение, управляющее замком. Потоки должны осуществлять сравнение параллельно по словам.

19. Дан список студентов по группам. Требуется создать многопоточное приложение для определения количества студентов с фамилией Иванов. Потоки должны осуществлять поиск совпадений по группам параллельно. Количество потоков является входным параметром программы, потоки проводят вычисления независимо друг от друга, количество групп может быть не кратно количеству потоков.

20. Среди студентов СФУ проведен опрос с целью определения процента студентов, знающих точную формулировку правила Буравчика. В результате собраны данные о количестве знатоков на каждом факультете по группам. Известно, что всего в СФУ обучается 10000 студентов. Требуется создать многопоточное приложение для определения процента знающих правило Буравчика студентов. Потоки должны осуществлять поиск количества знатоков по факультету. Искомый процент определяет главный поток. Количество потоков является входным параметром программы, потоки проводят вычисления независимо друг от друга, количество факультетов может быть не кратно количеству потоков.

21. Даны результаты сдачи экзамена по дисциплине «Средства разработки параллельных программ» по группам. Требуется создать многопоточ-

ное приложение, вычисляющее количество двоечников и отличников. Потоки должны осуществлять вычисления параллельно по группам. Количество потоков является входным параметром программы, потоки проводят вычисления независимо друг от друга, количество групп может быть не кратно количеству потоков.

22. Руководство заготовительной компании «Рога и Копыта» проводит соревнование по заготовке рогов среди своих региональных отделений. Все данные по результатам заготовки рогов (заготовитель, его результат) хранятся в общей базе данных по отделениям. Требуется создать многопоточное приложение для поиска лучшего заготовителя. Потоки должны осуществлять поиск победителя параллельно по отделениям. Главный поток определит победителя. Количество потоков является входным параметром программы, потоки проводят вычисления независимо друг от друга, количество отделений может быть не кратно количеству потоков.

Лабораторная работа №2

Практические приемы построения многопоточных приложений

Цели и задачи: Изучить работу с потоками. Научиться разбивать задачу на части, для последующего их выполнения различными потоками. Решение задачи с помощью библиотеки Pthread.

Время: 2 часа

Потоки предоставляют возможность проведения параллельных или псевдопараллельных, в случае одного процессора, вычислений. Потоки могут порождаться во время работы программы, процесса или другого потока. Основное отличие потоков от процессов заключается в том, что различные потоки имеют различные пути выполнения, но при этом пользуются общей памятью. Таким образом, несколько порожденных в программе потоков, могут пользоваться глобальными переменными, и любое изменение данных одним потоком, будет доступно и для всех остальных.

Существует небольшое число моделей построения многопоточных приложений. Опишем кратко основные.

Итеративный параллелизм используется для реализации нескольких потоков (часто идентичных), каждый из которых содержит циклы. Потоки программы, описываются итеративными функциями и работают совместно над решением одной задачи.

Рекурсивный параллелизм используется в программах с одной или несколькими рекурсивными процедурами, вызов которых независим. Это технология «разделяй-и-властвуй» или «перебор-с-возвратами».

Производители и потребители – это парадигма взаимодействующих неравноправных потоков. Одни из потоков «производят» данные, другие их «потребляют». Часто такие потоки организуются в **конвейер**, через который проходит информация. Каждый поток конвейера потребляет выход своего предшественника и производит входные данные для своего последователя. Другой распространенный способ организации потоков – древовидная структура или сети слияния, на этом основан, в частности, принцип **дихотомии**.

Клиенты и серверы – еще один способ взаимодействия неравноправных потоков. Клиентский поток запрашивает сервер и ждет ответа. Серверный поток ожидает запроса от клиента, затем действует в соответствии с поступившим запросом.

Управляющий и рабочие – модель организации вычислений, при которой существует поток, координирующий работу всех остальных потоков. Как правило, управляющий поток распределяет данные, собирает и анализирует результаты.

Взаимодействующие равные – модель, в которой исключен не занимающийся непосредственными вычислениями управляющий поток. Распределение работ в таком приложении либо фиксировано заранее, либо динамически определяется во время выполнения. Одним из распространенных способов динамического распределения работ является **«портфель задач»**. Портфель задач, как правило, реализуется с помощью разделяемой переменной, доступ к которой в один момент времени имеет только один процесс.

Порядок выполнения лабораторной работы

1. Выбрать модель приложения, наиболее точно отвечающую специфике задачи. Разработать алгоритм решения задания, с учетом разделения вычислений между несколькими потоками. Желательно избегать ситуаций изменения одних и тех же общих данных несколькими потоками. Если же избежать этого невозможно, необходимо использовать алгоритмы с активным ожиданием или неделимые операции.

2. Реализовать алгоритм с применением функций библиотеки Pthread и протестировать его на нескольких примерах.

3. *Самостоятельная работа.* Реализовать алгоритм с применением функций WinAPI. Сравнить возможности обоих подходов.

Пример

Найти произведение матриц A*B, где A и B – матрицы 3x3.

Обсуждение. Разделим задачу между тремя потоками. Каждый поток будет вычислять элементы результирующей матрицы, стоящие в определенной строке.

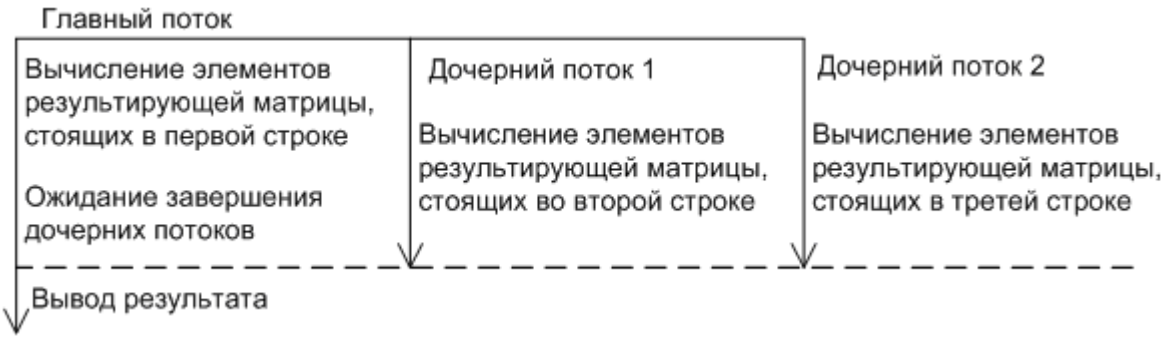

Рис. 2.1. Схема потоков

Реализуем задачу, используя библиотеку PTHREAD.

```
#include <stdio.h>
#include <pthread.h>
int A[3][3] = { 1, -2, 0, 4, 6, 2, -3, 4, -2 } ;
int B[3][3] = {0, 2, 0, 1, 1, 1, 5, -3, 10} ;
int C[3][3] ;  //результирующая матрица

//стартовая функция для дочерних потоков
void* func(void *param) //param – номер строки, заполняемой потоком
{
// вычисление элементов результирующей матрицы,
// стоящих в строке с номером param
    for (int i=0 ; i<3 ; i++)
    {
        C[(int)param][i]=0 ;
        for (int j=0 ; j<3 ; j++)
        C[(int)param][i]+=A[(int)param][j]*B[j][i] ;
    }
}

int main()
{   // создание дочерних потоков
    pthread_t mythread1,  mythread2 ;
    pthread_create(&mythread1, NULL, func, (void *)1) ;
    pthread_create(&mythread2, NULL, func, (void *)2) ;

    // заполнение первой строки результирующей матрицы
    func((void *)0) ;

    // ожидание завершения дочерних потоков
    pthread_join(mythread1,NULL) ;
    pthread_join(mythread2,NULL) ;

    // вывод результата вычислений всех потоков
    for (int i=0 ; i<3 ; i++)
    {   fprintf(stdout,"\n") ;
        for (int j=0 ; j<3 ; j++)
        fprintf(stdout, "%d ",C[i][j]) ;
    }

    return 1;
}
```

В приведенном примере не используются средства синхронизации (мьютексы, семафоры и пр.), поскольку каждый поток осуществляет запись, исключительно в выделенные для него элементы массива, индексы которых не пересекаются для различных потоков.

Реализация этого же алгоритма с применением функций WinAPI приведена ниже.

```
#include <stdio.h>
#include <conio.h>
#include <windows.h>

const int n = 3;
int a[n][n] = { 1,-2, 0, 4, 6, 2,-3, 4,-2};
int b[n][n] = { 0, 2, 0, 1, 1, 1, 5,-3,10};
int c[n][n];

DWORD WINAPI ThreadFunc(PVOID pvParam)
{
    int num,i,j;

    num = (*((int *)pvParam));

    for(i=0;i<n;i++)
    {
        c[num][i]=0 ;
        for(j=0;j<n; j++) c[num][i]+=a[num][j]*b[j][i];
    }

    DWORD dwResult = 0;
    return dwResult;
}

int main(int argc, char** argv)
{
    int i,j;
    int x[n];

    DWORD dwThreadId[n-1],dw;
    HANDLE hThread[n-1];

    // создание дочерних потоков
    for (i=1;i<n;i++)
    {
        x[i] = i;
        hThread[i-1] = CreateThread(NULL,0,ThreadFunc,(PVOID)&x[i], 0, &dwThreadId[i-1]);
        if(!hThread) printf("main process: thread %d not execute!",i);

    // заполнение первой строки результирующей матрицы
    x[0] = 0;
    ThreadFunc((PVOID)&x[0]);

    // ожидание завершения дочерних потоков
    dw = WaitForMultipleObjects(n-1,hThread,TRUE,INFINITE);
```

```
    // вывод результата вычислений всех потоков
    for(i=0;i<n;i++)
    {
        fprintf(stdout,"\n") ;
        for(j=0;j<n;j++) printf("%d ",c[i][j]) ;
    }
    getch();
    return 0;
}
```

Варианты заданий

1. Вычислить произведение матриц A и B. Входные данные: произвольные квадратные матрицы A и B одинаковой размерности. Решить задачу двумя способами: 1) количество потоков является входным параметром, при этом размерность матриц может быть не кратна количеству потоков; 2) количество потоков заранее неизвестно и не является параметром задачи.

2. Найти определитель матрицы A. Входные данные: целое положительное число *n*, произвольная матрица A размерности *n* x *n*. Решить задачу двумя способами: 1) количество потоков является входным параметром, при этом размерность матриц может быть не кратна количеству потоков; 2) количество потоков заранее неизвестно и не является параметром задачи.

3. Найти алгебраическое дополнение для каждого элемента матрицы. целое положительное число *n*, произвольная матрица A размерности *n* x *n*. Решить задачу двумя способами: 1) количество потоков является входным параметром, при этом размерность матриц может быть не кратна количеству потоков; 2) количество потоков заранее неизвестно и не является параметром задачи.

4. Найти обратную матрицу для матрицы A. Входные данные: целое положительное число *n*, произвольная матрица A размерности *n* x *n*. Решить задачу двумя способами: 1) количество потоков является входным параметром, при этом размерность матриц может быть не кратна количеству потоков; 2) количество потоков заранее неизвестно и не является параметром задачи.

5. Определить ранг матрицы. Входные данные: целое положительное число *n*, произвольная матрица A размерности *n* x *n*. Решить задачу двумя способами: 1) количество потоков является входным параметром, при этом размерность матриц может быть не кратна количеству потоков; 2) количество потоков заранее неизвестно и не является параметром задачи.

6. Вычислить прямое произведение множеств A1, A2, A3, A4. Входные данные: множества чисел A1, A2, A3, A4, мощности множеств могут быть не равны между собой и мощность каждого множества больше или равна 1. ко-

личество потоков определяется, исходя из мощностей множеств и не является параметром задачи.

7. Вычислить прямое произведение множеств A1, A2, A3… An. Входные данные: целое положительное число *n*, множества чисел A1, A2, A3… An, мощности множеств равны между собой и мощность каждого множества больше или равна 1. Количество потоков определяется, исходя из мощности множеств, и не является параметром задачи.

8. Используя формулы Крамера, найти решение системы линейных уравнений.

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + a_{14}x_4 = b_1 \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + a_{24}x_4 = b_2 \\ a_{31}x_1 + a_{32}x_2 + a_{33}x_3 + a_{34}x_4 = b_3 \\ a_{41}x_1 + a_{42}x_2 + a_{43}x_3 + a_{44}x_4 = b_4 \end{cases}$$

Предусмотреть возможность деления на ноль. Входные данные: коэффициенты системы. Оптимальное количество потоков выбрать самостоятельно.

9. Представить выражение $(ax + by)^n$ в виде

$$C_1a^nx^n + C_2b^ny^n + C_3a^{n-1}x^{n-1}by + C_4b^{n-1}y^{n-1}ax + C_5a^{n-2}x^{n-2}b^2y^2 + C_6b^{n-2}y^{n-2}a^2x^2 + \text{K}$$

Входные данные: числа *a* и *b*, целое положительное число *n*. Количество потоков выбирается в зависимости от показателя степени *n* и не является параметром задачи.

10. Определить, является ли множество C объединением множеств A и B (A ∪ B), пересечением множеств (A ∩ B), разностью множеств A и B (A\B), разностью множеств B и A (B\A). Входные данные: множества целых положительных чисел A, B, C. Количество потоков определяется, исходя из мощностей множеств, и не является параметром задачи.

11. Найти все возможные тройки компланарных векторов. Входные данные: множество не равных между собой векторов (*x, y, z*), где *x*, *y*, *z* – числа. Количество потоков зависит от мощности множества векторов, и не является параметром задачи.

12. Определить, делится ли целое число A, содержащее от 1 до 1000 значащих цифр, на 2, 3, 4, 5, 6, 7, 8, 9, 10. Входные данные: целое положительное число A, записанное в файле. Количество потоков зависит числа A, и не является параметром задачи.

13. Определить индексы i, j ($i \neq j$), для которых выражение A[i] – A[i+1] + A[i+2] – A[i+3] + … ± A[j] имеет максимальное значение. Входные данные: массив чисел A, произвольной длины большей 10. Количество потоков не является параметром задачи.

14. Определить индексы i, j, для которых существует наиболее длинная последовательность A[i] < A[i+1] < A[i+2] < A[i+3] < … < A[j]. Входные данные: массив чисел A, произвольной длины большей 1000. Количество потоков не является параметром задачи.

15. Определить индексы i1, j1, i2, j2, для которых существует наибольшее количество значений

A[i1,j1], A[i1,j1+1], A[i1,j1+2], … A[i1,j2],
A[i1+1,j1], A[i1+1,j1+1], A[i1+1,j1+2], … A[i1+1,j2],
……..
A[i2,j1], A[i2,j1+1], A[i2,j1+2], … A[i2,j2],

равных между собой. Входные данные: двумерный массив целых чисел A, размерности 5x5. Количество потоков не является параметром задачи.

16. Определить множество индексов i, для которых (A[i] - B[i]) или (A[i] + B[i]) являются простыми числами. Входные данные: массивы целых положительных чисел A и B, произвольной длины ≥ 1000. Количество потоков зависит от размерностей массивов, и не является параметром задачи.

17. Определить множество индексов i, для которых A[i] и B[i] не имеют общих делителей (единицу в роли делителя не рассматривать). Входные данные: массивы целых положительных чисел A и B, произвольной длины ≥ 1000. Количество потоков зависит от размерностей массивов, и не является параметром задачи.

18. Вывести список всех целых чисел, содержащих от 4 до 9 значащих цифр, которые после умножения на n, будут содержать все те же самые цифры в произвольной последовательности и в произвольном количестве. Входные данные: целое положительное число n, больше единицы и меньше десяти. Количество потоков зависит от размерностей массивов, и не является параметром задачи.

19. Вычислить $\int_a^b f(x)dx$, используя метод прямоугольников. Входные данные: числа a и b, функция f(x) определяется с помощью программной функции. При суммировании использовать принцип дихотомии.

20. *Задача об инвентаризации по рядам.* После нового года в библиотеке СФУ обнаружилась пропажа каталога. После поиска и наказания виноватых, ректор дал указание восстановить каталог силами студентов. Фонд библиотека представляет собой прямоугольное помещение, в котором находится M рядов по N шкафов по K книг в каждом шкафу. Требуется создать многопоточное приложение, составляющее каталог. При решении задачи использовать метод «портфель задач», причем в качестве отдельной задачи задается составление каталога одним студентом для одного ряда.

21. Задача об инвентаризации по книгам. После нового года в библиотеке СФУ обнаружилась пропажа каталога. После поиска и наказания, виноватых ректор дал указание восстановить каталог силами студентов. Фонд библиотека представляет собой прямоугольное помещение, в котором находится M рядов по N шкафов по K книг в каждом шкафу. Требуется создать многопоточное приложение, составляющее каталог. При решении задачи использовать метод «портфель задач», причем в качестве отдельной задачи задается внесение в каталог записи об отдельной книге.

22. *Задача для агронома.* Председатель колхоза «Светлый путь» Сидоров В.И. получил из райцентра указание, что в связи с составлением единого земельного кадастра, необходимо представить справку о площади колхозных земель. Известно, что колхозные угодья с запада и востока параллельны меридиану, на севере ограничены параллелью, а с юга выходят к реке, описываемой функцией f(x). Требуется создать многопоточное приложение, вычисляющее площадь угодий методом адаптивной квадратуры (см. лекции). При решении использовать парадигму рекурсивного параллелизма. Замечание: кривизну Земли в силу малости площади угодий можно не учитывать.

23. *Задача о наследстве.* У старого дона Энрике было два сына, у каждого из сыновей – еще по два сына, каждый из которых имел еще по два сына. Умирая, дон Энрике завещал все свое богатство правнукам в разных долях. Адвокат дон Хосе выполнил задачу дележа наследства в меру своих способностей. Правнуки заподозрили адвоката в укрывательстве части наследства. Требуется создать многопоточное приложение, которое при известных сумме завещания дона Энрике и доле каждого наследника, проверяет честность адвоката. При решении использовать принцип дихотомии.

24. У одной очень привлекательной студентки есть N поклонников. Традиционно в день св. Валентина очень привлекательная студентка проводит романтический вечер с одним из поклонников. Счастливый избранник заранее не известен. С утра очень привлекательная студентка получает N «валентинок» с различными вариантами романтического вечера. Выбрав наиболее заманчивое предложение, студентка извещает счастливчика о своем согласии, а остальных – об отказе. Требуется создать многопоточное приложение, моделирующее поведение студентки. При решении использовать парадигму «клиент-сервер» с активным ожиданием.

25. *Задача о производстве булавок.* В цехе по заточке булавок все необходимые операции осуществляются тремя рабочими. Первый из них берет булавку и проверяет ее на предмет кривизны. Если булавка не кривая, то рабочий передает ее своему напарнику. Напарник осуществляет собственно заточку и передает заточенную булавку третьему рабочему, который осуществляет контроль качества операции. Требуется создать многопоточное прило-

жение, моделирующее работу цеха. При решении использовать парадигму «производитель-потребитель» с активным ожиданием.

26. *Задача про экзамен.* Преподаватель проводит экзамен у группы студентов. Каждый студент заранее знает свой билет и готовит по нему ответ. Подготовив ответ, он передает его преподавателю. Преподаватель просматривает ответ и сообщает студенту оценку. Требуется создать многопоточное приложение, моделирующее действия преподавателя и студентов. При решении использовать парадигму «клиент-сервер» с активным ожиданием.

27. *Первая задача о Винни-Пухе, или неправильные пчелы.* Неправильные пчелы, подсчитав в конце месяца убытки от наличия в лесу Винни-Пуха, решили разыскать его и наказать в назидание всем другим любителям сладкого. Для поисков медведя они поделили лес на участки, каждый из которых прочесывает одна стая неправильных пчел. В случае нахождения медведя на своем участке стая проводит показательное наказание и возвращается в улей. Если участок прочесан, а Винни-Пух на нем не обнаружен, стая также возвращается в улей. Требуется создать многопоточное приложение, моделирующее действия пчел. При решении использовать парадигму портфеля задач.

28. *Первая военная задача.* Темной-темной ночью прапорщики Иванов, Петров и Нечепорчук занимаются хищением военного имущества со склада родной военной части. Будучи умными людьми и отличниками боевой и строевой подготовки, прапорщики ввели разделение труда: Иванов выносит имущество со склада, Петров грузит его в грузовик, а Нечепорчук подсчитывает рыночную стоимость добычи. Требуется составить многопоточное приложение, моделирующее деятельность прапорщиков. При решении использовать парадигму «производитель-потребитель» с активным ожиданием.

30. *Задача о Пути Кулака.* На седых склонах Гималаев стоят два древних буддистских монастыря: Гуань-Инь и Гуань-Янь. Каждый год в день сошествия на землю боддисатвы Араватти монахи обоих монастырей собираются на совместное празднество и показывают свое совершенствование на Пути Кулака. Всех соревнующихся монахов разбивают на пары, победители пар бьются затем между собой и так далее, до финального поединка. Монастырь, монах которого победил в финальном бою, забирает себе на хранение статую боддисатвы. Реализовать многопоточное приложение, определяющего победителя. В качестве входных данных используется массив, в котором хранится количество энергии Ци каждого монаха. При решении использовать принцип дихотомии.

31. *Первая задача об Острове Сокровищ.* Шайка пиратов под предводительством Джона Сильвера высадилась на берег Острова Сокровищ. Не

смотря на добытую карту старого Флинта, местоположение сокровищ попрежнему остается загадкой, поэтому искать клад приходится практически на ощупь. Так как Сильвер ходит на деревянной ноге, то самому бродить по джунглям ему не с руки. Джон Сильвер поделил остров на участки, а пиратов на небольшие группы. Каждой группе поручается искать клад на одном из участков, а сам Сильвер ждет на берегу. Пираты, обшарив свою часть острова, возвращаются к Сильверу и докладывают о результатах. Требуется создать многопоточное приложение с управляющим потоком, моделирующее действия Сильвера и пиратов.

32. *Вторая задача об Острове Сокровищ.* Шайка пиратов под предводительством Джона Сильвера высадилась на берег Острова Сокровищ. Не смотря на добытую карту старого Флинта, местоположение сокровищ попрежнему остается загадкой, поэтому искать клад приходится практически на ощупь. Так как Сильвер ходит на деревянной ноге, то самому бродить по джунглям ему не с руки. Джон Сильвер поделил остров на участки, а пиратов на небольшие группы. Каждой группе поручается искать клад на нескольких участках, а сам Сильвер ждет на берегу. Группа пиратов, обшарив одну часть острова, переходит к другой, еще необследованной части. Закончив поиски, пираты возвращаются к Сильверу и докладывают о результатах. Требуется создать многопоточное приложение с управляющим потоком, моделирующее действия Сильвера и пиратов. При решении использовать парадигму портфеля задач.

33. *Пляшущие человечки.* На тайном собрании глав преступного мира города Лондона председатель собрания профессор Мориарти постановил: отныне вся переписка между преступниками должна вестись тайнописью. В качестве стандарта были выбраны «пляшущие человечки», шифр, в котором каждой букве латинского алфавита соответствует хитроумный значок. Реализовать многопоточное приложение, шифрующее исходный текст (в качестве ключа используется кодовая таблица, устанавливающая однозначное соответствие между каждой буквой и каким-нибудь числом). Каждый поток шифрует свои кусочки текста. При решении использовать парадигму портфеля задач.

34. *И снова пляшущие человечки.* Узнав о планах преступников озвученных в задаче 33, Шерлок Холмс предложил лондонской полиции специальную машину для дешифровки сообщений злоумышленников. Реализовать многопоточное приложение, дешифрующее кодированный текст. В качестве ключа используется известная кодовая таблица, устанавливающая однозначное соответствие между каждой буквой и каким-нибудь числом. Процессом узнавания кода в решении задачи пренебречь. Каждый поток дешифрует свои кусочки текста. При решении использовать парадигму портфеля задач.

ЛАБОРАТОРНАЯ РАБОТА №3

Механизмы синхронизации

1. Двоичные семафоры: защита совместного доступа к памяти

Цели и задачи: Изучить работу с двоичными семафорами. Научиться корректно использовать общие данные потоков.

Время: 1 час

Для предотвращения ошибок взаимодействия между потоками, можно воспользоваться двоичными семафорами. Двоичные семафоры (мьютексы) довольно просты в использовании, они могут находиться в двух состояниях, открытом и закрытом. Поток может закрыть только открытый мутекс, если поток пытается закрыть уже закрытый мутекс, то выполнение этого потока приостанавливается, до тех пор, пока мутекс не станет открытым. Таким образом, двоичные семафоры блокируют выполнение нужных потоков, когда это необходимо.

В лабораторной работе двоичные семафоры используются для ограничения доступа к разделяемым всеми потоками переменным.

Порядок выполнения лабораторной работы

1. Переосмыслить алгоритм решения предыдущей задачи с учетом новых требований, описанных в задании.

2. Реализовать алгоритм с применением функций работы с мьютексами библиотеки Pthread и протестировать его на нескольких примерах.

3. *Самостоятельная работа.* Реализовать алгоритм с применением функций работы с критической секцией WinAPI. Сравнить возможности обоих подходов.

Пример

Найти результат произведения матриц A*B, где A и B – матрицы 3x3. Записать в общую очередь промежуточные результаты вычислений потоков, с указанием идентификатора потока или индивидуального номера потока, присваиваемого потоку пользователем. Защитить операции с общей очередью посредством двоичного семафора. Вывести общую очередь, после завершения основных вычислений.

Обсуждение. Поскольку каждый поток записывает результаты вычислений в общую очередь, то одновременная запись в очередь несколькими потоками или передача управления к другому потоку, когда операция записи не закончена, могут привести к разнообразным и малоприятным последствиям, например, таким как потеря записи нескольких результатов или разрушение связей между элементами очереди. Для предотвращения подобных ситуаций

воспользуемся двоичным семафором, с помощью которого будет защищена операция записи в очередь.

Реализуем задачу, используя библиотеку Pthread. Полужирным шрифтом отмечены изменения, внесенные в предыдущий алгоритм вычисления произведения матриц.

```
#include <stdio.h>
#include <pthread.h>

pthread_mutex_t mutex ; //двоичный семафор

int A[3][3] = { 1, -2, 0, 4, 6, 2, -3, 4, -2 } ;
int B[3][3] = {0, 2, 0, 1, 1, 1, 5, -3, 10} ;
int C[3][3] ; // результирующая матрица

struct result  // структура очереди
{
   char str[50] ;  // для записи результата вычислений и дополнительной информации
   rezult *next ;  // указатель на следующий элемент очереди
} ;

rezult *head ;  // указатель на первый элемент очереди
rezult *newrez ; // указатель/для создания новых элементов очереди
rezult *rez ;   // указатель на текущий последний элемент очереди

//стартовая функция для дочерних потоков
void* func(void *param)
{
// вычисление элементов результирующей матрицы,
// стоящих в строке с номером param
   for (int i=0 ; i<3 ; i++)
   {
      C[(int)param][i]=0 ;
      for (int j=0 ; j<3 ; j++)
      C[(int)param][i]+=A[(int)param][j]*B[j][i] ;

      pthread_mutex_lock(&mutex) ; //протокол входа в КС: закрыть двоичный семафор

      //начало критической секции – запись результата в очередь
      newrez = new rezult ;
      sprintf(newrez->str,"Поток %d: вычислен элемент [%d][%d] = %d",
         (int)param, (int)param, i, C[(int)param][i]) ;
      newrez->next = NULL ;
      rez->next = newrez ;
      rez = newrez ;
      //конец критической секции

      pthread_mutex_unlock(&mutex) ;   //протокол выхода из КС:
                                       // открыть двоичный семафор
   }
}

int main()
{
```

```
    head = new rezult ; rez = head ; //создание первого элемента очереди
    pthread_mutex_init(&mutex, NULL) ; //инициализация двоичного семафора

    //создание дочерних потоков
    pthread_t mythread1,  mythread2 ;
    pthread_create(&mythread1, NULL, func, (void *)1) ;
    pthread_create(&mythread2, NULL, func, (void *)2) ;

    //заполнение первой строки результирующей матрицы
    func((void *)0) ;

    //ожидание завершения дочерних потоков
    pthread_join(mythread1,NULL) ;
    pthread_join(mythread2,NULL) ;

    //вывод очереди
    rez = head->next ;
    while (rez!=NULL)
    {   fprintf(stdout,"\n%s",rez->str) ; rez = rez->next ; }

    //вывод результата вычислений всех потоков
    for (int i=0 ; i<3 ; i++)
    {
        fprintf(stdout,"\n") ;
        for (int j=0 ; j<3 ; j++)
        fprintf(stdout, "%d ",C[i][j]) ;
    }
    return 1;
}
```

Задание

Выполнить задачу из предыдущей лабораторной работы со следующими изменениями.

Для потоков, выполняющих вычисления, записать в общую очередь промежуточные результаты, с указанием идентификатора потока или индивидуального номера потока, присваиваемого потоку пользователем. Защитить операции с общей очередью посредством двоичного семафора. Вывести общую очередь, после завершения основных вычислений.

Для задач, использующих парадигму «портфеля задач», защитить неделимое обращение к «портфелю» посредством двоичного семафора.

Для задач, использующих активное ожидание, заменить постоянную проверку разделяемой переменной операциями с двоичным семафором.

Самостоятельная работа. Реализовать алгоритм с применением функций критической секции WinAPI. Сравнить возможности обоих подходов.

2. Семафоры: защита критических секций, условная синхронизация

Цели и задачи: Изучить работу с семафорами. Научиться выделять критические секции алгоритма и защищать их с помощью семафоров.

Время: 4 часа

Каждый семафор содержит неотрицательное целое значение. Любой поток может изменять значение семафора. Когда поток пытается уменьшить значение семафора, происходит следующее: если значение больше нуля, то оно уменьшается, если же значение равно нулю, поток приостанавливается до того момента, когда значение семафора станет положительным, тогда значение уменьшается и поток продолжает работу. Операция увеличения значения семафора является не блокирующей.

Критической секцией многопоточного алгоритма чаще всего являются операции записи в общие данные, которые могут быть изменены или открыты для чтения несколькими потоками. Для защиты критических секций могут быть использованы мьютексы или семафоры.

Синхронизация потоков носит иногда более сложный характер. Например, при наступлении какого-то условия необходимо приостановить поток до изменения ситуации (другим процессом) или, наоборот, при наступлении какого-то условия следует запустить поток. Одним из механизмов условной синхронизации процессов также являются семафоры.

Порядок выполнения лабораторной работы

1. Разработать алгоритм решения задания, с учетом разделения вычислений между несколькими потоками. Определить критические секции алгоритма. Ввести мьютексы или семафоры для защиты критических секций. Если в задании указан конкретный механизм для защиты, использовать только его. Составить схему потоков.

2. Реализовать алгоритм с применением функций работы с семафорами библиотеки Pthread и протестировать его на нескольких примерах.

3. *Самостоятельная работа.* Реализовать алгоритм с применением функций работы с событиями и семафорами WinAPI. Сравнить возможности обоих подходов.

Пример

Задача о кольцевом буфере. Потоки производители и потребители разделяют кольцевой буфер, состоящий из 100 ячеек. Производители передают сообщение потребителям, помещая его в конец очереди буфера. Потребители сообщение извлекают из начала очереди буфера. Создать многопоточное приложение с потоками писателями и читателями. Предотвратить такие си-

туации как, изъятие сообщения из пустой очереди или помещение сообщения в полный буфер. При решении задачи использовать семафоры.

Обсуждение. Пусть для определенности буфер – это целочисленный массив из 100 элементов. Задача обладает двумя критическими секциями. Первая критическая секция связана с операциями чтения-записи нескольких потоков в общий буфер. Вторая критическая секция определяется тем, что буфер являются конечным, запись должна производиться только в те ячейки, которые являются свободными или уже прочитаны потоками-читателями (условная взаимная синхронизация).

Для защиты первой критической секции воспользуемся двумя двоичными семафорами (мьютексами). Один двоичный семафор сделает возможным запись в буфер только для одного потока-писателя. Второй двоичный семафор сделает возможным чтение из буфера только для одного потока-читателя. Операция чтения должна быть защищена, потому что она является и операцией записи тоже, так как поток, прочитавший ячейку буфера, обязан ее как-то пометить. Иначе через определенное время выполнения программы, операция записи может стать невозможной или некорректной, в силу того, что буфер конечен. Операции чтения и записи могут проходить параллельно, так как всегда происходят в разных ячейках.

Для условной синхронизации воспользуемся двумя семафорами. Значение первого семафора показывает, сколько ячеек в буфере свободно. Ячейка свободна, когда в нее еще не осуществлялась запись или ячейка была прочитана. Значение второго семафора показывает, сколько ячеек в буфере занято. Естественно, операция записи не может быть выполнена, пока количество занятых ячеек равно 100 (или количество свободных ячеек равно 0), и операция чтения не может быть выполнена, пока количество свободных ячеек равно 100 (или количество занятых ячеек равно 0). Для блокировки потока воспользуемся условиями, заключенными в скобки, исходя из особенностей поведения семафоров.

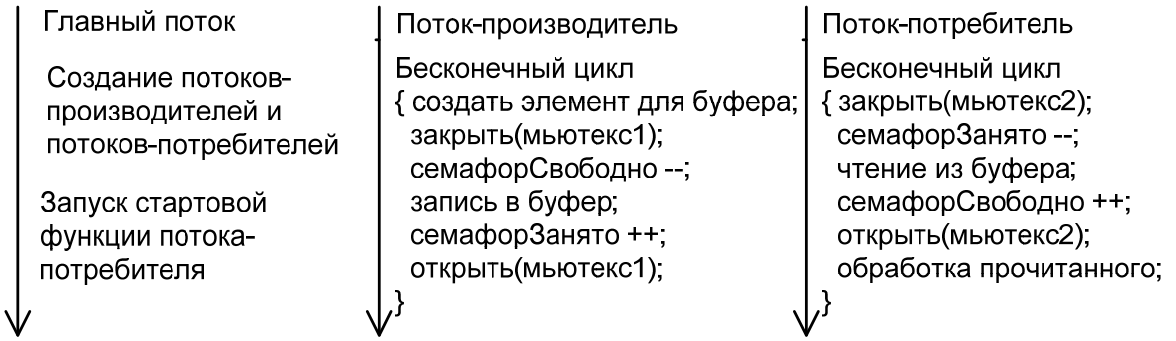

Рис. 3.1. Схема потоков

Реализуем задачу, используя библиотеку Pthread.

```
#include <pthread.h>
#include <semaphore.h>
#include <stdlib.h>
#include <stdio.h>

int buf[100] ; //буфер
int front = 0 ; //индекс для чтения из буфера
int rear = 0 ; //индекс для записи в буфер

sem_t  empty ; //семафор, отображающий насколько  буфер пуст
sem_t  full ; //семафор, отображающий насколько полон буфер

pthread_mutex_t mutexD ; //мутекс для операции записи
pthread_mutex_t mutexF ; //мутекс для операции чтения

//стартовая функция потоков – производителей(писателей)
void *Producer(void *param)
{
    int data, i ;
    while (1)
    {
        //создать элемент для буфера
        for (i=0,data=0 ; i<100 ; i++)
            data += rand()/(RAND_MAX/10) – 5 ;

        //поместить элемент в буфер
        pthread_mutex_lock(&mutexD) ; //защита операции записи

        sem_wait(&empty) ; //количество свободных ячеек уменьшить на единицу
        buf[rear] = data ; rear = (rear+1)%100 ; //критическая секция
        sem_post(&full) ; //количество занятых ячеек увеличилось на единицу

        pthread_mutex_unlock(&mutexD) ;
    }
}

//стартовая функция потоков – потребителей(читателей)
void *Consumer(void *param)
{
    int result ;
    while (1)
    {
        //извлечь элемент из буфера
        pthread_mutex_lock(&mutexF) ; //защита операции чтения

        sem_wait(&full) ; //количество занятых ячеек уменьшить на единицу
        result = buf[front] ; front = (front+1)%100 ; //критическая секция
        sem_post(&empty) ; //количество свободных ячеек увеличилось на единицу

        pthread_mutex_unlock(&mutexF) ;

        //обработать полученный элемент
        fprintf(stdout,” - %d -”,result) ;
    }
}

int main()
{
    int i ;
```

```
    //инициализация мутексов и семафоров
    pthread_mutex_init(&mutexD, NULL) ;
    pthread_mutex_init(&mutexF, NULL) ;
    sem_init(&empty, 0, 100) ; //количество свободных ячеек равно 100
    sem_init(&full, 0, 0) ; //количество занятых ячеек равно 0

    //запуск производителей
    pthread_t threadP[3] ;
    for (i=0 ; i<3 ; i++)
        pthread_create(&threadP[i],NULL,Producer,NULL) ;

    //запуск потребителей
    pthread_t threadC[4] ;
    for (i=0 ; i<4 ; i++)
        pthread_create(&threadC[i],NULL,Consumer,NULL) ;

    //пусть главный поток будет потребителем
    Consumer(NULL) ;
}
```

Ниже приведено решение задачи о кольцевом буфере с помощью функций WinAPI.

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <windows.h>

const int n = 100, \\ длина буфера
          m = 7, \\ количество производителей
          k = 3; \\ количество потребителей

int buf[n], front = 0, rear = 0;

HANDLE hSemFull,hSemEmpty, \\ семафоры для условной синхронизации
          hMutexD, hMutexF; \\ мьютексы для исключительного доступа

// процесс, пополняющий буфер
DWORD WINAPI Producer(PVOID pvParam)
{
    int num;
    long prev;

    num = *((int *)pvParam);
    printf("thread %d (producer): start!\n",num);

    while(true)
    {
        WaitForSingleObject(hSemEmpty,INFINITE);
        WaitForSingleObject(hMutexD,INFINITE);
        buf[rear] = rand()%n;
        printf("\nproducer %d: data = %d to %d",num,buf[rear],rear);
        rear = (rear+1)%n;
        Sleep(1000);
        ReleaseMutex(hMutexD);
        ReleaseSemaphore(hSemFull,1,&prev);
    }
```

```
    return 0;
}
// процесс, берущий данные из буфера
DWORD WINAPI Consumer(PVOID pvParam)
{
    int num,data;
    long prev;

    num = *((int *)pvParam);
    printf("thread %d (consumer): start!\n",num);

    while(true)
    {
        WaitForSingleObject(hSemFull,INFINITE);
        WaitForSingleObject(hMutexF,INFINITE);
        data = buf[front];
        printf("\nconsumer %d: data = %d from %d",num,data,front);
        front = (front+1)%n;
        Sleep(1000);
        ReleaseMutex(hMutexF);
        ReleaseSemaphore(hSemEmpty,1,&prev);
    }

    return 0;
}
int main(int argc, char** argv)
{
    int i, x[k+m];

    DWORD dwThreadId[k+m],dw;
    HANDLE hThread[k+m];

    hSemEmpty = CreateSemaphore(NULL,n,n,"Empty");
    hSemFull = CreateSemaphore(NULL,0,n,"Full");

    hMutexD = CreateMutex(NULL,FALSE,"MutexD");
    hMutexF = CreateMutex(NULL,FALSE,"MutexF");


    for(i=0;i<k;i++)
    {
        x[i] = i;

        hThread[i] = CreateThread(NULL,0,Producer,(PVOID)&x[i], 0, &dwThreadId[i]);

        if(!hThread) printf("main process: thread %d not execute!",i);
    }

    for(i=k;i<k+m;i++)
    {
        x[i] = i;

        hThread[i] = CreateThread(NULL,0,Consumer,(PVOID)&x[i], 0, &dwThreadId[i]);
        if(!hThread) printf("main process: thread %d not execute!",i);
    }
```

```
    WaitForMultipleObjects(k+m,hThread,TRUE,INFINITE);
    // закрытие описателей событий
    CloseHandle(hSemFull);
    CloseHandle(hSemEmpty);
    CloseHandle(hMutexF);
    CloseHandle(hMutexD);
    return 0;
}
```

Варианты заданий

1. *Задача о парикмахере.* В тихом городке есть парикмахерская. Салон парикмахерской мал, ходить там может только парикмахер и один посетитель. Парикмахер всю жизнь обслуживает посетителей. Когда в салоне никого нет, он спит в кресле. Когда посетитель приходит и видит спящего парикмахера, он будет его, садится в кресло и спит, пока парикмахер занят стрижкой. Если посетитель приходит, а парикмахер занят, то он встает в очередь и засыпает. После стрижки парикмахер сам провожает посетителя. Если есть ожидающие посетители, то парикмахер будит одного из них и ждет пока тот сядет в кресло парикмахера и начинает стрижку. Если никого нет, он снова садится в свое кресло и засыпает до прихода посетителя. Создать многопоточное приложение, моделирующее рабочий день парикмахерской.

2. *Задача о Винни-Пухе или правильные пчелы.* В одном лесу живут n пчел и один медведь, которые используют один горшок меда, вместимостью H глотков. Сначала горшок пустой. Пока горшок не наполнится, медведь спит. Как только горшок заполняется, медведь просыпается и съедает весь мед, после чего снова засыпает. Каждая пчела многократно собирает по одному глотку меда и кладет его в горшок. Пчела, которая приносит последнюю порцию меда, будит медведя. Создать многопоточное приложение, моделирующее поведение пчел и медведя.

3. *Задача о читателях и писателях.* Базу данных разделяют два типа процессов – читатели и писатели. Читатели выполняют транзакции, которые просматривают записи базы данных, транзакции писателей и просматривают и изменяют записи. Предполагается, что в начале БД находится в непротиворечивом состоянии (т.е. отношения между данными имеют смысл). Каждая отдельная транзакция переводит БД из одного непротиворечивого состояния в другое. Для предотвращения взаимного влияния транзакций процесс-писатель должен иметь исключительный доступ к БД. Если к БД не обращается ни один из процессов-писателей, то выполнять транзакции могут одновременно сколько угодно читателей. Создать многопоточное приложение с потоками-писателями и потоками-читателями. Реализовать решение, используя семафоры.

4. *Задача об обедающих философах.* Пять философов сидят возле круглого стола. Они проводят жизнь, чередуя приемы пищи и размышления. В центре стола находится большое блюдо спагетти. Спагетти длинные и запутанные, философам тяжело управляться с ними, поэтому каждый из них, что бы съесть порцию, должен пользоваться двумя вилками. К несчастью, философам дали только пять вилок. Между каждой парой философов лежит одна вилка, поэтому эти высококультурные и предельно вежливые люди договорились, что каждый будет пользоваться только теми вилками, которые лежат рядом с ним (слева и справа). Написать многопоточную программу, моделирующую поведение философов с помощью семафоров. Программа должна избегать фатальной ситуации, в которой все философы голодны, но ни один из них не может взять обе вилки (например, каждый из философов держит по одной вилки и не хочет отдавать ее). Решение должно быть симметричным, то есть все потоки-философы должны выполнять один и тот же код.

5. *Задача о каннибалах.* Племя из n дикарей ест вместе из большого горшка, который вмещает m кусков тушеного миссионера. Когда дикарь хочет обедать, он ест из горшка один кусок, если только горшок не пуст, иначе дикарь будит повара и ждет, пока тот не наполнит горшок. Повар, сварив обед, засыпает. Создать многопоточное приложение, моделирующее обед дикарей. При решении задачи пользоваться семафорами.

6. *Задача о курильщиках.* Есть три процесса-курильщика и один процесс-посредник. Курильщик непрерывно скручивает сигареты и курит их. Чтобы скрутить сигарету, нужны табак, бумага и спички. У одного процесса-курильщика есть табак, у второго – бумага, а у третьего – спички. Посредник кладет на стол по два разных случайных компонента. Тот процесс-курильщик, у которого есть третий компонент, забирает компоненты со стола, скручивает сигарету и курит. Посредник дожидается, пока курильщик закончит, затем процесс повторяется. Создать многопоточное приложение, моделирующее поведение курильщиков и посредника. При решении задачи использовать семафоры.

7. *Военная задача.* Анчуария и Тарантерия – два крохотных латиноамериканских государства, затерянных в южных Андах. Диктатор Анчуарии, дон Федерико, объявил войну диктатору Тарантерии, дону Эрнандо. У обоих диктаторов очень мало солдат, но очень много снарядов для минометов, привезенных с последней американской гуманитарной помощью. Поэтому армии обеих сторон просто обстреливают наугад территорию противника, надеясь поразить что-нибудь ценное. Стрельба ведется по очереди до тех пор, пока либо не будут уничтожены все цели, либо стоимость потраченных снарядов не превысит суммарную стоимость всего того, что ими можно уничтожить. Создать многопоточное приложение, моделирующее военные действия.

8. *Задача о читателях и писателях-2 («грязное чтение»).* Базу данных разделяют два типа потоков – читатели и писатели. Читатели выполняют транзакции, которые просматривают записи базы данных, транзакции писателей и просматривают и изменяют записи. Предполагается, что в начале БД находится в непротиворечивом состоянии (т.е. отношения между данными имеют смысл). Транзакции выполняются в режиме «грязного чтения», то есть процесс-писатель не может получить доступ к БД только в том случае, если ее занял другой процесс-писатель, а процессы-читатели ему не мешают. Создать многопоточное приложение с потоками-писателями и потоками-читателями. Реализовать решение, используя семафоры, и не используя блокировки чтения-записи.

9. *Задача о читателях и писателях-3 («подтвержденное чтение»).* Базу данных разделяют два типа процессов – читатели и писатели. Читатели выполняют транзакции, которые просматривают записи базы данных, транзакции писателей и просматривают и изменяют записи. Предполагается, что в начале БД находится в непротиворечивом состоянии (т.е. отношения между данными имеют смысл). Каждая отдельная транзакция переводит БД из одного непротиворечивого состояния в другое. Транзакции выполняются в режиме «подтвержденного чтения», то есть процесс-писатель не может получить доступ к БД в том случае, если ее занял другой процесс-писатель или процесс-читатель. К БД может обратиться одновременно сколько угодно процессов-читателей. Процесс читатель получает доступ к БД, даже если ее занял процесс-писатель. Создать многопоточное приложение с потоками-писателями и потоками-читателями. Реализовать решение, используя семафоры, и не используя блокировки чтения-записи.

10. *Задача о супермаркете.* В супермаркете работают два кассира, покупатели заходят в супермаркет, делают покупки и становятся в очередь к случайному кассиру. Пока очередь пуста, кассир спит, как только появляется покупатель, кассир просыпается. Покупатель спит в очереди, пока не подойдет к кассиру. Создать многопоточное приложение, моделирующее рабочий день супермаркета.

11. *Задача о магазине.* В магазине работают три отдела, каждый отдел обслуживает один продавец. Покупатель, зайдя в магазин, делает покупки в произвольных отделах, и если в выбранном отделе продавец не свободен, покупатель становится в очередь и засыпает, пока продавец не освободится. Создать многопоточное приложение, моделирующее рабочий день магазина.

12. *Задача о больнице.* В больнице два врача принимают пациентов, выслушивают их жалобы и отправляют их или к стоматологу или к хирургу или к терапевту. Стоматолог, хирург и терапевт лечат пациента. Каждый врач может принять только одного пациента за раз. Пациенты стоят в очере-

ди к врачам и никогда их не покидают. Создать многопоточное приложение, моделирующее рабочий день клиники.

13. *Задача о гостинице.* В гостинице 30 номеров, клиенты гостиницы снимают номер на одну ночь, если в гостинице нет свободных номеров, клиенты устраиваются на ночлег рядом с гостиницей и ждут, пока любой номер не освободится. Создать многопоточное приложение, моделирующее работу гостиницы.

14. *Задача о гостинице-2 (умные клиенты).* В гостинице 10 номеров с ценой 200 рублей, 10 номеров с ценой 400 рублей и 5 номеров с ценой 600 руб. Клиент, зашедший в гостиницу, обладает некоторой суммой и получает номер по своим финансовым возможностям, если тот свободен. Если среди доступных клиенту номеров нет свободных, клиент уходит искать ночлег в другое место. Создать многопоточное приложение, моделирующее работу гостиницы.

15. *Задача о гостинице - 3 (дамы и джентльмены).* В гостинице 10 номеров рассчитаны на одного человека и 15 номеров рассчитаны на двух человек. В гостиницу приходят клиенты дамы и клиенты джентльмены, и конечно они могут провести ночь в номере только с представителем своего пола. Если для клиента не находится подходящего номера, он уходит искать ночлег в другое место. Создать многопоточное приложение, моделирующее работу гостиницы.

16. *Задача о клумбе.* На клумбе растет 40 цветов, за ними непрерывно следят два садовника и поливают увядшие цветы, при этом оба садовника очень боятся полить один и тот же цветок. Создать многопоточное приложение, моделирующее состояния клумбы и действия садовников. Для изменения состояния цветов создать отдельный поток.

17. *Задача о нелюдимых садовниках.* Имеется пустой участок земли (двумерный массив) и план сада, который необходимо реализовать. Эту задачу выполняют два садовника, которые не хотят встречаться друг с другом. Первый садовник начинает работу с верхнего левого угла сада и перемещается слева направо, сделав ряд, он спускается вниз. Второй садовник начинает работу с нижнего правого угла сада и перемещается снизу вверх, сделав ряд, он перемещается влево. Если садовник видит, что участок сада уже выполнен другим садовником, он идет дальше. Садовники должны работать параллельно. Создать многопоточное приложение, моделирующее работу садовников. При решении задачи использовать мутексы.

18. *Задача о картинной галерее.* Вахтер следит за тем, чтобы в картинной галерее было не более 50 посетителей. Для обозрения представлены 5 картин. Посетитель ходит от картины к картине, и если на картину любуются более чем десять посетителей, он стоит в стороне и ждет, пока число желаю-

щих увидеть картину не станет меньше. Посетитель может покинуть галерею. Создать многопоточное приложение, моделирующее работу картинной галереи.

19. *Задача о Винни-Пухе - 3 или неправильные пчелы - 2.* N пчел живет в улье, каждая пчела может собирать мед и сторожить улей (N>3). Ни одна пчела не покинет улей, если кроме нее в нем нет других пчел. Каждая пчела приносит за раз одну порцию меда. Всего в улей может войти тридцать порций меда. Вини-Пух спит пока меда в улье меньше половины, но как только его становится достаточно, он просыпается и пытается достать весь мед из улья. Если в улье находится менее чем три пчелы, Вини-Пух забирает мед, убегает, съедает мед и снова засыпает. Если в улье пчел больше, они кусают Вини-Пуха, он убегает, лечит укус, и снова бежит за медом. Создать многопоточное приложение, моделирующее поведение пчел и медведя.

20. *Задача о болтунах.* N болтунов имеют телефоны, ждут звонков и звонят друг другу, чтобы побеседовать. Если телефон занят, болтун будет звонить, пока ему кто-нибудь не ответит. Побеседовав, болтун не унимается и или ждет звонка или звонит на другой номер. Создать многопоточное приложение, моделирующее поведение болтунов. Для решения задачи использовать мутексы.

21. *Задача о магазине - 2 (забывчивые покупатели).* В магазине работают два отдела, каждый отдел обладает уникальным ассортиментом. В каждом отделе работает один продавец. В магазин ходят исключительно забывчивые покупатели, поэтому каждый покупатель носит с собой список товаров, которые желает купить. Покупатель приобретает товары точно в том порядке, в каком они записаны в его списке. Продавец может обслужить только одного покупателя за раз. Покупатель, вставший в очередь, засыпает пока не дойдет до продавца. Продавец засыпает, если в его отделе нет покупателей, и просыпается, если появится хотя бы один. Создать многопоточное приложение, моделирующее работу магазина.

22. *Задача о программистах.* В отделе работают три программиста. Каждый программист пишет свою программу и отдает ее на проверку другому программисту. Программист проверяет чужую программу, когда его собственная уже написана. По завершении проверки, программист дает ответ: программа написана правильно или написана неправильно. Программист спит, если не пишет свою программу и не проверяет чужую программу. Программист просыпается, когда получает заключение от другого программиста. Если программа признана правильной, программист пишет другую программу, если программа признана неправильной, программист исправляет ее и отправляет на проверку тому же программисту, который ее проверял. Создать многопоточное приложение, моделирующее работу программистов.

3. Условные переменные

Цели и задачи: Изучить работу с условными переменными. Научиться защищать критические секции алгоритма с помощью условных переменных.

Время: 1 час

При условной синхронизации потоков естественно использовать вместо семафоров условные переменные, которые, как и семафоры, осуществляют блокировку потоков и их пробуждение. Используя условную переменную, можно приостанавливать работу потока, оставляя его ожидать сигнала к пробуждению; можно разбудить один или все потоки, ожидающие сигнала от данной условной переменной. Условные переменные, в отличие от семафоров, не обладают внутренними состояниями. Описание условий блокирования или пробуждения потока – обязанность программиста.

Порядок выполнения лабораторной работы

1. Переосмыслить алгоритм решения предыдущей задачи с учетом новых требований, описанных в задании. Составить схему потоков.

2. Реализовать алгоритм с применением функций работы с семафорами и условными переменными библиотеки Pthread и протестировать его на нескольких примерах.

Пример

Выполнить предыдущую задачу, используя для условной синхронизации потоков условные переменные.

Семафоры full и empty, отображающие насколько пуст или полон буфер, могут быть легко заменены условными переменными. Для описания условий к вызовам функций условных переменных воспользуемся глобальной переменной count, показывающей, сколько ячеек буфера заняты.

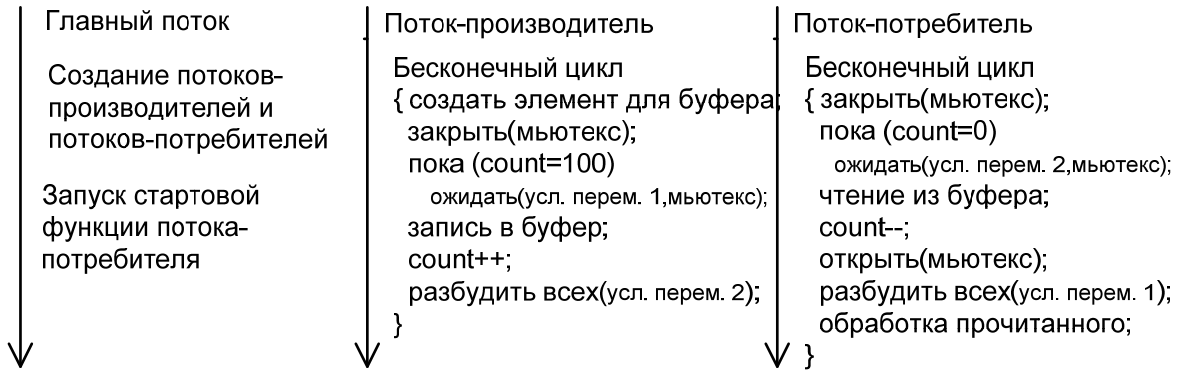

Рис. 3.2. Схема потоков

Реализуем задачу, используя библиотеку Pthread. Изменения, внесенные в программу, выделены жирным шрифтом.

```
#include <pthread.h>
#include <stdlib.h>
#include <stdio.h>

int buf[100] ; //буфер
int front = 0 ; //индекс для чтения из буфера
int rear = 0 ; //индекс для записи в буфер
int count = 0 ; //количество занятых ячеек буфера

pthread_mutex_t mutex ; // мьютекс для условных переменных

    // поток-писатель блокируется этой условной переменной,
    // когда количество занятых ячеек становится равно 100
pthread_cond_t not_full ;

    // поток-читатель блокируется этой условной переменной,
    // когда количество занятых ячеек становится равно 0
pthread_cond_t not_empty ;


//стартовая функция потоков – производителей (писателей)
void *Producer(void *param)
{
    int data, i ;

    while (1)
    {
        //создать элемент для буфера
        for (i=0,data=0 ; i<100 ; i++)
        data += rand()/(RAND_MAX/10) – 5 ;

        //поместить элемент в буфер – начало критической секции
        pthread_mutex_lock(&mutex) ;

        //заснуть, если количество занятых ячеек равно сто
        while (count == 100 )  pthread_cond_wait(&not_full, &mutex) ;

        //запись в общий буфер      buf[rear] = data ; rear = (rear+1)%100 ;
        count++ ; //появилась занятая ячейка

        //конец критической секции
        pthread_mutex_unlock(&mutex) ;

        //разбудить потоки-читатели после добавления элемента в буфер
        pthread_cond_broadcast(&not_empty) ;
    }
}

//стартовая функция потоков – потребителей(читателей)
void *Consumer(void *param)
{
    int result ;

    while (1)
    {

        //извлечь элемент из буфера – начало критической секции
```

```
        pthread_mutex_lock(&mutex) ;
        //заснуть, если количество занятых ячеек равно нулю
        while (count == 0 )  pthread_cond_wait(&not_empty, &mutex) ;
        //изъятие из общего буфера – начало критической секции
        result = buf[front] ; front = (front+1)%100 ;
        count-- ; //занятая ячейка стала свободной
        //конец критической секции
        pthread_mutex_unlock(&mutex) ;
        //разбудить потоки-писатели после получения элемента из буфера
        pthread_cond_broadcast(&not_full) ;
        //обработать полученный элемент
        fprintf(stdout,” - %d -”,result) ;
    }
}
int main()
{
    int i ;
    //инициализация семафоров и усл. переменных
    pthread_mutex_init(&mutex, NULL) ;
    pthread_cond_init(&not_full, NULL) ;
    pthread_cond_init(&not_empty, NULL) ;
    //запуск производителей
    pthread_t threadP[3] ;
    for (i=0 ; i<3 ; i++)
        pthread_create(&threadP[i],NULL,Producer,NULL) ;
    //запуск потребителей
    pthread_t threadC[4] ;
    for (i=0 ; i<4 ; i++)
        pthread_create(&threadC[i],NULL,Consumer,NULL) ;
    //пусть главный поток будет потребителем
    Consumer(NULL) ;
}
```

Условные переменные используют один и тот же мутекс, так как критические секции, защищенные этими переменными, изменяют глобальную переменную count.

Задание

Выполнить предыдущую задачу, используя для защиты критических секций и синхронизации условные переменные.

ЛАБОРАТОРНАЯ РАБОТА №4

Блокировки чтения-записи

Цели и задачи: Изучить работу с блокировками чтения-записи. Научиться создавать с помощью библиотеки Pthread многопоточные приложения, работающие с разделяемым ресурсом.

Время: 2 часа

Блокировка чтения-записи, также как и двоичный семафор, может находиться или в открытом или в закрытом состоянии. Различие состоит в том, что блокировка может быть закрыта для записи или закрыта для чтения. Если блокировка закрыта для записи, то никакой другой поток не может закрыть блокировку. Если блокировка закрыта для чтения, то никакой поток не может закрыть блокировку для записи. Если закрытие блокировки в данный момент не возможно, поток останавливается до того момента, когда блокировка станет открытой. Таким образом, блокировки чтения-записи предоставляют возможность читать общие данные, сразу многим потокам, или изменять общие данные только одному потоку, при этом операции чтения и операция записи не могут быть выполнены одновременно.

Использование блокировок чтения-записи актуально, когда одновременно работает несколько потоков – читателей. В противном случае, правильнее пользоваться двоичными семафорами, так как операции закрытия и открытия двоичного семафора происходят быстрее, чем аналогичные операции блокировки чтения-записи.

Порядок выполнения лабораторной работы

1. Разработать алгоритм решения задания, с учетом разделения вычислений между несколькими потоками. Определить критические секции алгоритма. Воспользоваться блокировками чтения-записи для защиты критических секций. Составить схему потоков.

2. Реализовать алгоритм с применением функций работы блокировками библиотеки Pthread и протестировать его на нескольких примерах.

Пример

Создать многопоточное приложение с потоками-писателями и потоками-читателями, работающими с общим одномерным массивом. Потоки-писатели изменяют произвольный элемент массива. Потоки-читатели получают значение произвольного элемента массива и находят факториал прочитанного значения. Для защиты операций с общими данными использовать блокировки чтения-записи.

Для защиты критических секций – операций с общим массивом введем блокировку чтения-записи. Потоки-писатели перед изменением элемента

массива будут закрывать блокировку для записи, потоки-читатели перед получением значения из массива будут закрывать блокировку для чтения. Благодаря этому, потоки-читатели смогут работать параллельно, потоки-писатели смогут изменять данные только последовательно, и операции чтения и операции записи не будут происходить одновременно.

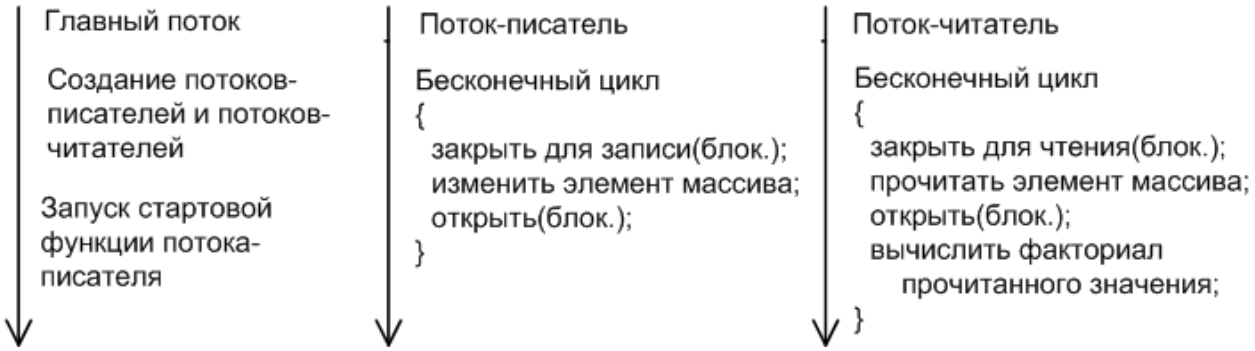

Рис. 4.1. Схема потоков

Для решения задачи достаточно одной блокировки чтения-записи, так как необходимо защитить операции с одним массивом. Однако существует и другой путь (в описанном примере он не рассматривается), ввести свою собственную блокировку чтения-записи для каждого элемента массива, т.е. использовать массив блокировок чтения-записи.

Реализуем задачу, используя библиотеку Pthread.

```
#include <pthread.h>
#include <stdlib.h>
#include <stdio.h>

int A[100] ;  //общий массив

pthread_rwlock_t rwlock ;  //блокировка чтения-записи

//стартовая функция потоков-читателей
void *funcRead(void *param)
{
    int number ;
    while (1)
    {
        //получить случайный индекс
        number = random()/(RAND_MAX/100) ;

        //закрыть блокировку для чтения
        pthread_rwlock_rdlock(&rwlock) ;

        //прочитать данные из общего массива – критическая секция
        int a = A[number] ;

        //открыть блокировку
        pthread_rwlock_unlock(&rwlock) ;
```

```
        long int F = 1 ;  //вычислить факториал
        for (int i = 2 ; i <= a ; i++)  F=F*i ;
        fprintf(stdout,"\nФакториал %d = %d", a,F) ;
    }
}
//стартовая функция потоков-писателей
void *funcWrite(void *param)
{
    int number ;
    while (1)
    {
        //получить случайный индекс
        number = random()/(RAND_MAX/100) ;
        //закрыть блокировку для записи
        pthread_rwlock_wrlock(&rwlock) ;
        //изменить элемент общего массива – критическая секция
        A[number] = random()/(RAND_MAX/10) ;
        //открыть блокировку
        pthread_rwlock_unlock(&rwlock) ;
        fprintf(stdout,"\nЭлемент %d изменен", number) ;
    }
}

int main()
{
    //инициализация блокировки чтения-записи
    pthread_rwlock_init(&rwlock, NULL) ;
    //заполнение общего массива
    for (int i=0 ; i<100 ; i++)  A[100] = random()/(RAND_MAX/10) ;
    //создание четырех потоков-читателей
    pthread_t threadR[4] ;
    for (int i=0 ; i<4 ; i++)
        pthread_create(&threadR[i], NULL, funcRead, NULL) ;
    //создание трех потоков-писателей
    pthread_t threadW[3] ;
    for (int i=0 ; i<3 ; i++)
        pthread_create(&threadW[i], NULL, funcWrite, NULL) ;
    //пусть главный поток будет потоком-писателем
    funcWrite(NULL) ;
    return 1;
}
```

Варианты заданий

1. Создать многопоточное приложение, работающее с общим файлом. Для защиты операций с файлом использовать блокировки чтения-записи. В приложении должны работать следующие потоки: 1) поток, полностью изменяющий файл; 2) поток, выводящий содержимое файла на экран; 3) поток,

определяющий количество вхождений в файле каждого отдельного символа; 4) потоки, добавляющие произвольные записи в файл.

2. Создать многопоточное приложение, работающее с общим двумерным массивом. Для защиты операций с общим массивом использовать массив блокировок чтения-записи для охраны строк общего массива и массив блокировок чтения-записи для охраны столбцов общего массива. В приложении должны работать следующие потоки: 1) потоки, изменяющие произвольный элемент массива; 2) потоки, изменяющие произвольную строку массива; 3) потоки, изменяющие произвольный столбец массива; 4) потоки, находящие произведение двух произвольных элементов массива; 5) потоки, находящие произведение двух произвольных строк массива; 6) потоки, находящие произведение двух произвольных столбцов массива.

3. Создать многопоточное приложение, работающее с общим двусвязным списком. Для защиты операций с двусвязным списком использовать блокировки чтения-записи. В приложении должны работать следующие потоки: 1) потоки, изменяющие произвольный элемент двусвязного списка; 2) потоки, читающие произвольный элемент двусвязного списка и выводящие его на экран; 3) потоки, определяющие количество элементов в списке; 4) потоки, добавляющие элемент в начало или конец списка; 5) потоки, удаляющие произвольный элемент из списка.

4. Создать многопоточное приложение, работающее с общим графом. Для защиты операций с графом использовать блокировки чтения-записи. Граф описывает множество городов и множество рейсов автобусов от города А к городу Б с указанием цены билета (по умолчанию, если рейс от А к Б, он идет и от Б к А, с одинаковой ценой). В приложении должны работать следующие потоки: 1) поток, изменяющий цену билета; 2) поток, удаляющий и добавляющий рейсы между городами; 3) поток, удаляющий старые города и добавляющий новые; 4) потоки, определяющие есть ли путь от произвольного города А до произвольного города Б, и какова цена такой поездки (если прямого пути нет, то найти любой путь из существующих).

5. Создать многопоточное приложение, работающее с общим файлом. Для защиты операций с общим файлом использовать блокировки чтения-записи. Файл содержит последовательность записей вида: Ф.И.О.1 – телефон1, Ф.И.О.2 – телефон2… В приложении должны работать следующие потоки: 1) потоки, находящие телефоны по указанной фамилии; 2) потоки, находящие Ф.И.О. по указанному телефону; 3) потоки, удаляющие и добавляющие записи в файл.

6. Создать многопоточное приложение, работающее с общим двумерным массивом. Для защиты операций с общим массивом использовать блокировки чтения-записи. Двумерный массив описывает сад. В приложении

должны работать следующие потоки: 1) поток-садовник следит за садом и поливает увядшие растения; 2) поток-природа может произвольно изменять состояние растений; 3) поток-монитор1 периодически выводит состояние сада в файл (не стирая предыдущее состояние); 4) поток-монитор2 выводит состояние сада на экран.

7. Создать многопоточное приложение, работающее с общим односвязным списком. Для защиты операций с односвязным списком использовать блокировки чтения-записи. В приложении должны работать следующие потоки: 1) потоки, изменяющие произвольный элемент списка; 2) потоки, читающие произвольный элемент списка и выводящие его на экран; 3) потоки, добавляющие элемент в начало или конец или середину списка; 4) потоки, удаляющие произвольный элемент из списка.

8. Создать многопоточное приложение, работающее с общим графом. Для защиты операций с графом использовать блокировки чтения-записи. Граф описывает множество состояний системы и множество переходов из состояния А в состояние Б, причем наличие перехода из А в Б, не означает наличия перехода из Б в А, кроме того переход из А в А может как существовать, так и не существовать. В приложении должны работать следующие потоки: 1) поток, добавляющий новые состояния системы; 2) поток, удаляющий старые состояния системы; 3) поток, изменяющий множество переходов; 4) потоки, определяющие может ли система перейти из одного состояния в другое.

9. Создать многопоточное приложение, работающее с общим файлом. Для защиты операций с общим файлом использовать блокировки чтения-записи. Файл содержит последовательность записей вида: Организация1 – Адрес1 – Телефон1, Организация2 – Адрес2 – Телефон2… В приложении должны работать следующие потоки: 1) потоки, находящие телефон и адрес указанной организации; 2) потоки, находящие организацию по указанному телефону; 3) поток, добавляющий записи в файл; 4) поток, удаляющий записи из файла; 5) поток, изменяющий существующую запись.

10. Создать многопоточное приложение, работающее с общим динамическим списком.

Группа 1 →	Студент 1 →	Список оценок
↓	Студент 2 →	Список оценок
↓	↓ …	
Группа 2 →	Студент 1 →	Список оценок
↓ …	↓ …	

Для защиты операций с динамическим списком использовать блокировки чтения-записи. В приложении должны работать следующие потоки: 1) поток, удаляющий группу; 2) поток, добавляющий новую группу и составляющий для нее список студентов; 3) потоки, удаляющие студента из груп-

пы; 4) потоки, добавляющие студента в группу; 5) потоки, добавляющие оценку в список оценок произвольного студента; 6) потоки, определяющие средний балл произвольного студента.

11. Создать многопоточное приложение, работающее с общим двусвязным списком. Каждый элемент списка содержит Ф.И.О., адрес и телефон. Для защиты операций с двусвязным списком использовать блокировки чтения-записи. В приложении должны работать следующие потоки: 1) потоки, добавляющие элемент в список; 2) потоки, удаляющие элемент из списка; 3) потоки, изменяющие запись в произвольном узле списка; 4) потоки, находящие информацию по известному Ф.И.О. или адресу или телефону.

12. Создать многопоточное приложение, работающее с общей таблицей и файлом. Таблица содержит записи вида символ1 – символ2, символ3 – символ4 и т.д., и определяет однозначный код. Для защиты операций с таблицей и файлом использовать блокировки чтения-записи. В приложении должны работать следующие потоки: 1) потоки, изменяющие записи в таблице (при этом код должен оставаться однозначным); 2) поток, изменяющий содержимое файла, в соответствие с кодом в таблице и сохраняющий все изменения в специальном файле; 3) поток, выводящий содержимое файла на экран; 4) поток, выводящий таблицу на экран.

13. Создать многопоточное приложение, работающее с общим двумерным массивом. Для защиты операций с общим массивом использовать блокировки чтения-записи. Двумерный массив состоит из нулей и единиц. В приложении должны работать следующие потоки: 1) поток, заменяющий единицу нулем, если в клетках, окружающих единицу нет других единиц; 2) поток, заменяющий ноль единицей, если в клетках, окружающих ноль, есть хотя бы две единицы; 3) поток, заменяющий единицу нулем, если в клетках, окружающих единицу, есть хотя бы пять единиц; 4) потоки, подсчитывающие количество нулей; 5) потоки, подсчитывающие количество единиц.

14. Создать многопоточное приложение, работающее с общим динамическим списком.

Склад 1 →	Товар 1 →	Количество → Цена за единицу товара
↓	Товар 2 →	Количество → Цена за единицу товара
↓	↓ …	
Склад 2 →	Товар 1 →	Количество → Цена за единицу товара
↓ …	↓ …	

Для защиты операций с динамическим списком использовать блокировки чтения-записи. В приложении должны работать следующие потоки: 1) потоки, добавляющие новый товар на произвольный склад; 2) потоки, изменяющие количество или цену товара; 3) поток, выводящий список всех товаров на складе; 4) поток, выводящий список товаров, не имеющихся на

складе (количество равно нулю); 5) поток, подсчитывающий общую стоимость всех товаров на складе.

15. Создать многопоточное приложение, работающее с общим двусвязным списком. Каждый элемент списка содержит матрицу размерности 3x3. Для защиты операций с двусвязным списком использовать блокировки чтения-записи. В приложении должны работать следующие потоки: 1) поток, добавляющий элементы в список; 2) поток, удаляющий элементы из списка; 3) поток, находящий определитель для случайного элемента списка; 4) поток, находящий произведение двух случайных элементов списка.

16. Создать многопоточное приложение, работающее с общим файлом. Для защиты операций с общим файлом использовать блокировки чтения-записи. Файл содержит последовательность записей вида: Автор – Название – Год издания. В приложении должны работать следующие потоки: 1) поток, удаляющий запись из файла; 2) поток, добавляющий запись в файл; 3) поток, находящий информацию, по известному автору; 4) поток, находящий информацию, по известному названию; 5) поток, составляющий список всех авторов, записанных в файле (без повторений).

17. Создать многопоточное приложение, работающее с общим динамическим списком.

Должность 1 →	Ф.И.О. 1 →	Заработная плата
↓	Ф.И.О. 2 →	Заработная плата
↓	↓ …	
Должность 2 →	Ф.И.О. 1 →	Заработная плата
↓ …	↓ …	

Для защиты операций с динамическим списком использовать блокировки чтения-записи. В приложении должны работать следующие потоки: 1) потоки, добавляющие нового работника; 2) потоки, увольняющие работников 3) потоки, изменяющие заработную плату работников; 4) поток, выводящий список всех работников по известной должности; 5) поток, подсчитывающий общую заработную плату по известной должности.

18. Создать многопоточное приложение, работающее с общим двумерным массивом целых чисел. Для защиты операций с общим массивом использовать массив блокировок чтения-записи для охраны строк общего массива и массив блокировок чтения-записи для охраны столбцов общего массива. В приложении должны работать следующие потоки: 1) потоки, изменяющие произвольный элемент массива; 2) потоки, находящие сумму элементов произвольного столбца; 3) потоки, находящие сумму элементов произвольной строки; 4) потоки, находящие сумму трех произвольных элементов массива; 5) поток, находящий определитель, составленный из элементов, лежащих на пересечении трех случайных строк и трех случайных столбцов.

19. Создать многопоточное приложение, работающее с общим односвязным списком. Каждый элемент списка содержит вектор (x, y, z). Для защиты операций с односвязным списком использовать блокировки чтения-записи. В приложении должны работать следующие потоки: 1) поток, добавляющий элементы в список; 2) поток, удаляющий элементы из списка; 3) поток, находящий скалярные произведения случайных элементов списка; 4) поток, находящий векторные произведения случайных элементов списка; 5) поток, находящий пары компланарных векторов 6) поток, находящий тройки компланарных векторов.

20. Создать многопоточное приложение, работающее с общим динамическим списком.

Группа 1 →	Понедельник →	Расписание (аудитория, время, предмет)
↓	Вторник →	Расписание
↓	↓ …	
Группа 2 →	Понедельник →	Расписание
↓ …	↓ …	

Для защиты операций с динамическим списком использовать блокировки чтения-записи. В приложении должны работать следующие потоки: 1) потоки, изменяющие расписание; 2) поток, определяющий общие дисциплины для двух произвольных групп; 3) поток, определяющий аудитории, используемые одновременно несколькими группами; 4) поток, выводящий на экран расписание по известной группе и известному дню.

21. Создать многопоточное приложение, работающее с общим графом. Для защиты операций с графом использовать блокировки чтения-записи. Граф описывает множество состояний системы и множество переходов из состояния А в состояние Б, причем наличие перехода из А в Б, не означает наличия перехода из Б в А, переход из А в А существует всегда. Мощность множества состояний системы является входным параметром функции main. В приложении должны работать следующие потоки: 1) поток, изменяющий множество переходов; 2) потоки, начинающие с указанного состояния и осуществляющие переходы по случайному принципу, вся история переходов записывается потоком в отдельный файл (свой для каждого потока). Количество таких потоков должно быть равно трем для каждого начального состояния.

22. Создать многопоточное приложение, работающее с общим двусвязным списком. Каждый элемент списка содержит множество неповторяющихся натуральных чисел произвольной мощности. Для защиты операций с двусвязным списком использовать блокировки чтения-записи. В приложении должны работать следующие потоки: 1) поток, удаляющий элементы из списка; 2) поток, добавляющий элементы в список; 3) поток, добавляющий элементы в произвольное множество; 4) поток, удаляющий элементы из произ-

вольного множества; 5) поток, находящий пересечение двух случайных множеств; 6) поток, находящий объединение двух случайных множеств; 7) поток, находящий разность двух случайных множеств, и добавляющий полученный результат в список множеств.

ЛАБОРАТОРНАЯ РАБОТА №5

Барьеры

Цели и задачи: Изучить работу с барьерами. Научиться использовать барьеры для синхронизации потоков и защиты критических секций.

Время: 2 часа.

Барьер – механизм синхронизации, позволяющий останавливать выполнение группы потоков, пока все потоки из этой группы не дойдут до определенной точки выполнения, помечаемой вызовом функции ожидания барьера. При создании барьер получает целое положительное число, оно показывает количество потоков в этой группе. Поток, ожидающий на барьере, останавливается, пока общее количество потоков, ожидающих на этом же барьере, не станет равно числу, указанному при инициализации барьера. Как только, число потоков, ожидающих на барьере, становиться достаточным, все эти потоки возвращаются к работе, и барьер может быть использован снова.

Порядок выполнения лабораторной работы

1. Разработать алгоритм решения задания, с учетом разделения вычислений между несколькими потоками. Для синхронизации потоков и защиты критических секций использовать барьеры. Составить схему потоков.

2. Реализовать алгоритм с применением функций работы с барьерами библиотеки Pthread и протестировать его на нескольких примерах.

Пример

Создать многопоточное приложение, в котором работают два потока, обладающие собственными массивами одинаковой размерности. Каждый поток должен заменить нулями, те элементы своего массива, которые равны соответственным элементам массива другого потока. При решении задачи не пользоваться общими массивами.

Потоки работают с собственными массивами и не могут напрямую обратиться к массиву другого потока. Для решения задачи используем следующую схему. Потоки будет обмениваться информацией о каждом соответственном элементе массива, и заменять его нулем, по выполнении условия

равенства элементов массива. Чтобы потоки работали согласованно, используем барьер.

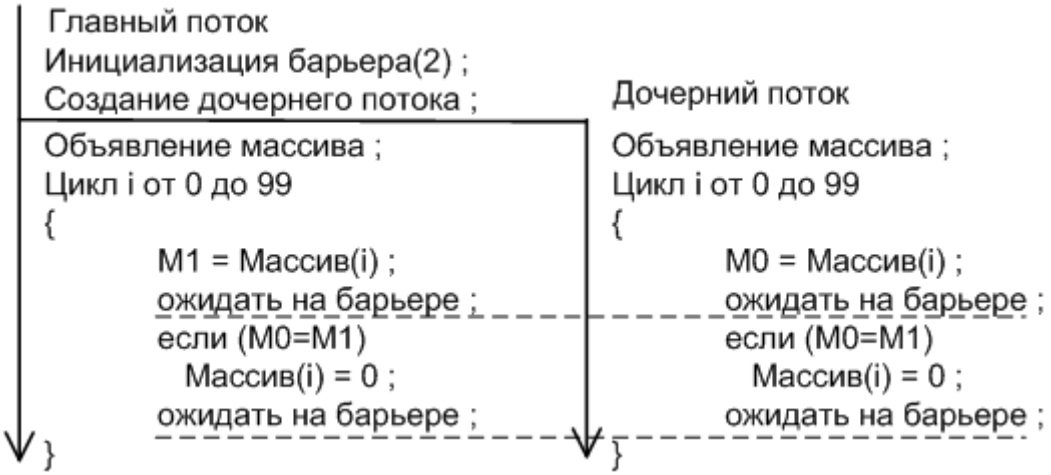

Рис. 5.1. Схема потоков

В описанной схеме барьер служит как для синхронизации, так и для предотвращения логических ошибок. Синхронизация потоков осуществляется ожиданием на барьере на каждом шаге цикла. Это позволяет потокам сравнивать элементы массивов с соответственными индексами. Для предотвращения появления логических ошибок необходимо двойное ожидание на барьере на каждом шаге цикла. Рассмотрим, к чему может привести снятие одного из ожиданий.

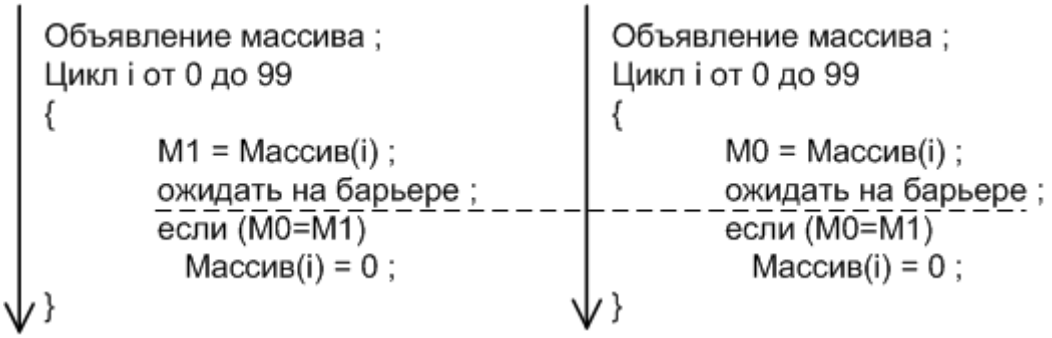

Рис. 5.2. Неверная схема потоков

По выполнении первой пары команд потоки выделяют элементы с одинаковыми индексами i. Пусть затем выполняется первый поток, он вычисляет команду «если» и записывает в M1 элемент с индексом i+1, затем он ожидает на барьере. Начинает работать второй поток, он выполняет команду «если», но сравнивает не элементы с одинаковыми индексами i, а i–й элемент с i+1-м элементом.

Реализуем задачу, используя библиотеку Pthread.

```
#include <pthread.h>

//для обмена информацией между потоками
//первый поток записывает результат в M0, второй в M1
int M0, M1 ;

pthread_barrier_t barr ; //барьер

pthread_t mythread ;  //идентификатор для дочернего потока

//стартовая функция
void* func(void *p)
{
    int Mas[100] ; //собственный массив потока

    for (int i=0 ; i<100 ; i++)
    {
        //первый поток записывает элемент массива в M0, второй в M1
        if ( (int)p == 0 ) M0=Mas[i] ;
        if ( (int)p == 1 ) M1=Mas[i] ;

        pthread_barrier_wait(&barr) ; //ожидание на барьере

        //если соответственные элементы массивов равны, заменим их нулями
        if (M0==M1) Mas[i]=0 ;

        pthread_barrier_wait(&barr) ; //ожидание на барьере
    }
}

int main()
{
    pthread_barrier_init(&barr, NULL, 2) ; //инициализация барьера со значением 2
    pthread_create(&mythread, NULL, func, (void *)0) ; //создание дочернего потока
    func((void *)1) ; return 1 ;
}
```

Варианты заданий

Для всех вариантов: Потоки не могут обращаться напрямую к данным других потоков. При решении задачи не использовать предварительную запись данных потоков в общие данные.

1. *Задача о новобранцах.* Строю новобранцев дается команда «налево» или «направо». Все новобранцы стараются исполнить приказ, но проблема в том, что они не знают где право, а где лево. Следовательно, каждый новобранец поворачивается либо направо, либо налево. Если новобранец повернулся и видит, что его сосед стоит к нему спиной, он считает, что все сделал правильно. Если же они сталкиваются лицом к лицу, то оба считают, что ошиблись, и разворачиваются на 180 градусов. Создать многопоточное приложение, моделирующее поведение строя новобранцев, пока он не придет к стационарному состоянию. Количество новобранцев ≥ 100. Отдельный поток отвечает за часть строя не менее 50 новобранцев.

2. Создать приложение с тремя потоками. Каждый поток работает со своим массивом, потоки проверяют сумму элементов своего массива с сум-

мами элементов других потоков и останавливаются, когда все три суммы равны между собой. Если суммы не равны, каждый поток прибавляет единицу к одному элементу массива или отнимает единицу от одного элемента массива, затем снова проверяет условие равенства сумм. На момент остановки всех трех потоков, суммы элементов массивов должны быть одинаковы.

3. Создать приложение с двумя потоками. Каждый поток работает с собственной строкой, строки имеют одинаковую длину. Дан набор из N пар кодирующих символов (a_i,b_i), известный первому потоку. Первый поток заменяет i-й элемент строки (заменяя a_i на b_i), если i-1-й или i-й или i+1-й элемент строки второго потока является кодирующим символом.

4. Создать приложение с двумя потоками. Каждый поток открывает на чтение собственный файл. Потоки считывают по слову из файла, и подсчитывают количество пар слов, начинающихся с одной и той же буквы.

5. Создать многопоточное приложение. Количество потоков > 1 и задается как входной параметр функции main. Потоки обладают собственными двумерными массивами одинаковой размерности. Найти множество индексов тех строк, которые содержат хотя бы один элемент, присутствующий во всех соответственных строках остальных массивов.

6. Создать приложение с двумя потоками. Оба потока случайно заполняют двумерные массивы нулями и единицами, затем проверяют условие равенства суммы элементов двух диагоналей первого массива с аналогичной суммой второго массива. Если условие не выполняется, потоки снова заполняют массивы, и опять проверяют условие.

7. Создать приложение с тремя потоками. Каждый поток работает с собственной очередью, каждый элемент очереди содержит число и символ. Первый поток составляет слово из тех символов, для которых равны между собой хотя бы два числа их трех соответственных чисел в разных потоках.

8. Создать приложение с двумя потоками. Каждый поток использует собственный массив. Необходимо записать в массив первого потока элементы, меньше либо равные любому элементу массива второго потока, при этом объединение начальных массивов должно совпадать с объединением массивов, полученных при решении. Для решения задачи использовать следующий алгоритм. Первый поток находит максимум, второй поток находит минимум, потоки меняются найденными значениями, пока максимум больше минимума.

9. Создать приложение с тремя потоками. Потоки работают с массивами одинаковой размерности. Массивы содержат положительные числа. Потоки производят одно действие над массивом, находят произведение и сумму элементов массива, затем проверяют условие равенства сумм трех потоков или равенства произведений трех потоков, если условие выполняется, потоки

завершают работу. В противном случае, все описанные шаги повторяются. Поток может производить над массивом следующие действия; заменить один произвольный четный элемент любым нечетным числом или заменить один произвольный нечетный элемент любым четным числом.

10. Создать приложение с двумя потоками. Каждый поток работает со строкой произвольной длины. Длина строк в разных потоках может быть не одинаковой. Существует общая строка произвольной длины, содержащая неповторяющиеся символы. Потоки меняются символами с одинаковыми индексами, если хотя бы один из этих символов есть в общей строке.

11. Создать приложение с тремя потоками. Два потока открывают на чтение различные файлы. Третий поток создает новый файл, записывая в него первый символ из файла первого потока, затем первый символ из файла второго потока, затем второй символ из файла первого потока, затем второй символ из файла второго потока и т.д., пока не будет достигнут конец одного из файлов.

12. Создать многопоточное приложение. Количество потоков > 1 и задается как входной параметр функции main. Потоки обладают собственными двумерными массивами одинаковой размерности. Найти индексы тех строк, суммы элементов которых равны между собой и существуют хотя бы два не одинаковых произведения элементов строк различных массивов.

14. Создать приложение с тремя потоками. Первый поток заполняет матрицу A, случайным образом, параллельно второй поток заполняет матрицу B, случайным образом. Матрицы A и B квадратные и имеют одинаковую размерность. Третий поток находит произведение матриц AB. Потоки прекращают работу, если определитель AB равен нулю, иначе все операции повторяются.

15. Создать приложение с тремя потоками. Два потока работают с очередями, содержащими равное количество элементов. Потоки сравнивают соответственные элементы очереди. Третий поток составляет собственную очередь, добавляя в нее не равные пары элементов.

16. Создать приложение с двумя потоками. Каждый поток работает с массивом, содержащим нули и единицы. Потоки обмениваются элементами: первый поток заменяет ноль единицей из массива второго потока, второй поток заменяет единицу нулем из массива первого потока. Обмен происходит столько раз, сколько это возможно.

17. Создать приложение с двумя потоками. Каждый поток работает с собственным массивом целых чисел. Поток может умножить один произвольный элемент массива на любое целое число или обнулить его. Потоки работают до тех пор, пока суммы элементов их массивов не станут равными.

18. Создать приложение с четырьмя потоками. Каждый поток работает с собственной строкой. Строки могут содержать только символы A, B, C, D. Поток может поменять символ A на C или C на A или B на D или D на B. Потоки останавливаются когда общее количество символов A и B становится равным хотя бы для трех строк.

19. Создать приложение с тремя потоками. Каждый поток работает с собственным двумерным массивом. Первый поток находит сумму элементов своего массива с индексом ij, которые равны элементу с индексом ij массива второго потока или элементу с индексом ji массива третьего потока.

20. Создать приложение с двумя потоками. Каждый поток работает с собственным двумерным массивом. Необходимо найти все пары (максимум i-й строки массива первого потока, минимум i-й строки массива второго потока), для которых минимум i-й строки массива первого потока меньше либо равен максимуму i-й строки массива второго потока.

21. Создать приложение с тремя потоками. Каждый поток работает с собственным массивом. Первый поток находит сумму элементов своего массива с индексом i, для которых i-й элемент массива второго потока является делителем i-го элемента массива третьего потока, либо наоборот.

22. Создать приложение с двумя потоками. Каждый поток работает с массивом, содержащим натуральные числа. Потоки обмениваются элементами: первый поток заменяет четное число нечетным из массива второго потока, второй поток заменяет нечетное число четным из массива первого потока. Обмен происходит столько раз, сколько это возможно.

23. Решить задачу о новобранцах (вариант 1). Количество новобранцев больше одного и меньше одиннадцати. Отдельный поток отвечает за одного новобранца.

Лабораторная работа № 6

Создание простого приложения в среде OpenMP

Цели и задачи: Научиться использовать OpenMP для разработки параллельных программ и распараллеливания уже написанных программ.

Время: 2 часа

Стандарт OpenMP предназначен для параллельного программирования для систем с общей памятью. Управление параллелизмом явно задается программистом соответствующими директивами препроцессора.

Порядок выполнения лабораторной работы

1. Создать и откомпилировать простое приложение в среде OpenMP (пример № 1). Доработать его следующим образом:

1) перед запуском параллельных нитей приложение должно определять максимально возможное количество нитей, которое одновременно может работать в системе;

2) если максимальное количество нитей меньше четырех, то параллельная секция должна выполняться с максимальным количеством нитей, иначе установить количество нитей для параллельной секции равным 4;

3) синхронизировать вывод потоков, что бы доступ к окну консоли был исключающим.

2. Разработать алгоритм решения задания.

3. Реализовать алгоритм в последовательной программе.

4. Распараллелить программу при помощи директив OpenMP, откомпилировать и отладить в среде OpenMP.

5. Составить схему потоков.

Пример № 1

```
#include <omp.h>
#include <stdio.h>
int main(int argc, char** argv)
{
    #pragma omp parallel
    {
        count=omp_get_thread_num();
        ItsMe=omp_get_num_threads();
        printf("Hello, OpenMP! I am %d of %d\n", count,ItsMe);
    }
    return 0;
}
```

Комиляция и запуск примера:

```
icc -openmp test.c
./a.out
```

Результат работы примера:

```
Hello, OpenMP! I am 0 of 4
Hello, OpenMP! I am 1 of 4
Hello, OpenMP! I am 2 of 4
Hello, OpenMP! I am 3 of 4
```

Пример № 2

Создать приложение, заполняющее таблицу значений $i^{3/2}$, $0 \le i \le 99$. Распараллелить его с помощью директив OpenMP.

Последовательная программа

```
#include <stdio.h>
#include <math.h>
#define N 100

int main(int argc, char** argv)
{
    int b[N];
    for (int i=0; i<N; i++)
        b[i]=i*sqrt(i);

    for (int i=0; i<N; i++)
        printf("%d ",b[i]);

    return 0;
}
```

Параллельная программа

```
#include <stdio.h>
#include <math.h>
#include <omp.h>
#define N 10

int main(int argc, char** argv)
{
    double b[N];
    #pragma omp parallel for
        for (int i=0; i<N; i++)  // этот цикл выполняется параллельно,
            b[i]=i*sqrt(i);        // итерации распределяются между нитями по умолчанию

    for (int i=0; i<N; i++) // этот цикл выполняется последовательно
        printf("%f ",b[i]);

    return 0;
}
```

Комиляция и запуск примера:

```
icc -openmp test.c
./a.out
```

Пример № 3

Дана последовательность натуральных чисел $a_0, \ldots, a_{99}$. Создать многопоточное приложение для поиска суммы квадратов $\sum a_i^2$. В приложении вычисления должны независимо выполнять четыре потока.

```
#include<omp.h>
#include<stdio.h>

int A[100] ;  //последовательность чисел a0…a99

main()
{
    int i;
    double sum = 0.0;
```

```
    if (omp_get_max_threads ()<4) { printf("мало нитей"); getch(); return(0);}
       else omp_set_num_threads(4);
    #pragma omp parallel for schedule(static,250) private(i,a2) \
       shared(A) reduction(+:sum)
       for(i=0; i < n; i++)
       {
          a2 = A[i]*A[i];
          sum = sum + a2;
       }
    printf("sum = %f\n",sum);
    return(0);
}
```

Для компиляции и запуска необходимо выполнить команды:

```
icc test.c -openmp
/a.out
```

Варианты заданий

1. Даны последовательности символов A = $\{a_0 \ldots a_{n-1}\}$ и C = $\{c_0 \ldots c_{к-1}\}$. В общем случае $n \neq k$. Создать OpenMP-приложение, определяющее, совпадают ли посимвольно строки A и C. Количество потоков является входным параметром программы, количество символов в строках может быть не кратно количеству потоков.

2. Дана последовательность символов C = $\{c_0 \ldots c_{n-1}\}$. Дан набор из N пар кодирующих символов (a_i, b_i). Создать OpenMP-приложение, кодирующее строку C следующим образом: поток 0 заменяет в строке C все символы a_0 на символы b_0, поток 1 заменяет в строке C все символы a_1 на символы b_1, и т.д. Количество потоков является входным параметром программы, количество символов в строке может быть не кратно количеству потоков.

3. Дана последовательность натуральных чисел $\{a_0 \ldots a_{n-1}\}$. Создать OpenMP-приложение для вычисления общей суммы и всех промежуточных сумм простых чисел последовательности.

4. Дана последовательность символов C = $\{c_0 \ldots c_{n-1}\}$ и символ b. Создать OpenMP-приложение для определения количество вхождений символа b в строку C. Количество символов и потоков являются входными параметрами программы, количество символов в строке может быть не кратно количеству потоков.

5. Дана последовательность натуральных чисел $\{a_0 \ldots a_{n-1}\}$. Создать OpenMP-приложение для поиска всех a_i, являющихся простыми числами. Количество потоков является входным параметром программы, потоки проводят вычисления независимо друг от друга, количество символов в строке может быть не кратно количеству потоков.

6. Дана последовательность натуральных чисел $\{a_0 \ldots a_{n-1}\}$. Создать OpenMP-приложение для поиска всех a_i, являющихся квадратами, любого натурального числа.

7. Дана последовательность натуральных чисел $\{a_0 \ldots a_{n-1}\}$. Создать OpenMP-приложение для вычисления выражения $a_0 - a_1 + a_2 - a_3 + a_4 - a_5 + \ldots$

8. Дана последовательность натуральных чисел $\{a_0 \ldots a_{n-1}\}$. Создать OpenMP-приложение для поиска суммы $\sum a_i$, где a_i – четные числа.

9. Дана последовательность натуральных чисел $\{a_0 \ldots a_{n-1}\}$. Создать OpenMP-приложение для поиска суммы $\sum a_i$, где a_i – простые числа.

10. Дана последовательность символов $C = \{c_0 \ldots c_{n-1}\}$. Создать OpenMP-приложение, определяющее является ли строка полиндромом (полиндром – фраза, читающаяся с права на лево и с лева на право одинаково, без учета пробелов). Количество символов потоков являются входными параметрами программы, количество символов в строке может быть не кратно количеству символов.

11. Даны результаты сдачи экзамена по курсу «Средства разработки параллельных программ» по студенческим группам. Требуется создать OpenMP-приложение, вычисляющее средний балл на курсе и средний балл каждой группы. Количество потоков и групп являются входными параметрами программы, количество групп может быть не кратно количеству потоков.

12. Охранное агентство разработало новую систему управления электронными замками. Для открытия двери клиент обязан произнести произвольную фразу из 25 слов. В этой фразе должно встречаться заранее оговоренное слово, причем только один раз. Требуется создать OpenMP-приложение, управляющее замком. Потоки должны осуществлять сравнение параллельно по 5 слов.

13. Дан список студентов по группам. Требуется создать OpenMP-приложение для определения количества студентов с фамилией Иванов. Потоки должны осуществлять поиск совпадений по группам параллельно. Количество потоков является входным параметром программы, потоки проводят вычисления независимо друг от друга, количество групп может быть не кратно количеству потоков.

14. Среди студентов СФУ проведен опрос с целью определения процента студентов, знающих точную формулировку правила Буравчика. В результате собраны данные о количестве знатоков на каждом факультете по группам. Известно, что всего в СФУ обучается 10000 студентов. Требуется создать OpenMP-приложение для определения процента знающих правило Буравчика студентов на факультете и во всем университете. Количество потоков является входным параметром программы, потоки проводят вычисле-

ния независимо друг от друга, количество факультетов может быть не кратно количеству потоков.

15. Даны результаты сдачи экзамена по дисциплине «Средства разработки параллельных программ» по группам. Требуется создать OpenMP-приложение, вычисляющее количество двоечников и отличников в каждой группе и по всем сдававшим экзамен. Количество потоков является входным параметром программы, количество групп может быть не кратно количеству потоков.

16. Руководство заготовительной компании «Рога и Копыта» проводит соревнование по заготовке рогов среди своих региональных отделений. Все данные по результатам заготовки рогов (заготовитель, его результат) хранятся в общей базе данных по отделениям. Требуется создать OpenMP-приложение для поиска лучшего заготовителя в каждом отделении и во всей компании. Количество потоков является входным параметром программы, количество отделений может быть не кратно количеству потоков.

17. Дана последовательность натуральных чисел $\{a_0 \ldots a_{n-1}\}$. Создать OpenMP-приложение для вычисления общей суммы и всех промежуточных сумм.

Лабораторная работа № 7

Отладка OpenMP –приложения: поиск ошибок, оптимизация

Цели и задачи: Изучить базовые методы и средства отладки многопоточных программ, в том числе и написанных с применением OpenMP.

Время: 4 часа

1. Работа с отладчиком Intel Thread Checker

Программы OpenMP являются многопоточными и могут подвергаться тем же ошибкам и испытывать те же неполадки производительности, что и приложения с явной многопоточностью. Отладка многопоточных программ может оказаться очень сложной, и в некоторых случаях практически невозможной.

Для определения многих ошибок и потенциальных проблемных мест в программе удобно использовать специализированные отладчики, такие как Intel Thread Checker. Этот отладчик встраивает свои функции в код программы и при ее запуске анализирует состояние потоков, переключателей и доступ к памяти. Отладчик способен выявить широкий спектр ошибок, таких как: неверный доступ к памяти; дедлоки; состояния гонок; ошибки при работе с семафорами и мьютексами.

Единственный минус такого способа отладки – это замедление исполнения программы в сотни раз.

При компиляции программы для последующей отладки необходимо указать ключ -tcheck компилятору:

icc -tcheck -o <имя исполняемого файла> имя исходного файла

После запуска созданной программы в текущем каталоге появится файл threadchecker.thr – он содержит информацию во внутреннем формате представления. Для получения информации в удобном виде необходимо использовать следующую команду:

tcheck_cl -f txt -w 80 threadchecker.thr

Порядок выполнения лабораторной работы

1. Запустить программу с использованием OpenMP для различных наборов входных данных. Отметить ошибки в работе программы.

2. Запустить программу с отладчиком Intel Thread Checker. Составить список потенциально проблемных мест и ошибок при синхронизации нескольких потоков.

3. Устранить ошибки в программе.

4. Вновь запустить Intel Thread Checker и убедиться в правильности работы программы.

Пример

```
1  #include <stdio.h>
2
3  #ifdef _OPENMP
4  #include <omp.h>
5  #endif
6
7  #define NMAX 11
8
9  double a[NMAX];
10 double b[NMAX];
11
12
13 int main(void){
14 int i;
15 a[0]=0.0;
16
17 #pragma omp parallel
18 {
19     #pragma omp for
20         for(i=1; i<NMAX; i++){
21             a[i]=1.0/i;
22             b[i]=a[i]+a[i-1];
23         }
24  }
25
26  for(i=1;i<NMAX; i++){
27      printf("%g\n",b[i]);
```

```
28  }
29
30  return 0;
31 }
```

Комиляция, запуск примера и запуск отладчика:

```
icc -tcheck -openmp -g -o conflict conflict.c
./conflict
tcheck_cl -f txt -w 80 threadchecker.thr
```

Результат работы отладчика отображен на рис. 7.1.

```
|ID|Short Des|Sever|Co|Contex|Description                          |1st Acc|2nd Acc|
|  |cription |ity  |un|t[Best|                                       |ess[Bes|ess[Bes|
|  |         |Name |t |]     |                                       |t]     |t]     |
|1 |Write -> |Error|9 |omp   |Memory read of a[] at                  |"confli|"confli|
|  |Read     |     |  |for   |"conflict.c":22 conflicts with a       |ct.c": |ct.c": |
|  |data-race|     |  |      |prior memory write of a[] at           |21     |22     |
|  |         |     |  |      |"conflict.c":21 (flow dependence)      |       |       |
|2 |Thread te|Infor|1 |WholeP|Thread termination at                  |"confli|"confli|
|  |rmination|matio|  |rogram|"conflict.c":13 - includes stack       |ct.c":1|ct.c":1|
|  |         |n    |  |1     |allocation of 8 MB and use of          |3      |3      |
|  |         |     |  |      |4.547 KB                               |       |       |
```

Рис. 7.1. Результат работы отладчика Intel Thread Checker для программы из примера

Внимательно изучите выдачу Intel Thread Checker. Отладчик сообщает о наличии «гонок» в строках 21 – 22 при доступе к массиву a. Для разрешения проблемы гонок в данном случае следует изменить вычисление элементов массива b так, чтобы элементы массива a не участвовали в вычислениях. Правильный программный код этого участка приведен ниже.

```
...
17 #pragma omp parallel
18  {
19 #pragma omp for
20  for(i=1; i<NMAX; i++){
21     a[i]=1.0/i;
22     b[i] = a[i]+1.0/(i-1);
23  }
24  }
...
```

Варианты заданий

1. openmp_itc_1.c

```
...
  double a[N];
  a[0]=1;
  double f = 2.0;

  # pragma omp parallel for
    for (i=1; i<N; i++)
    {
```

2. openmp_itc_2.c

```
...
   double a[N],b[N];
   a[0] = 0;

  # pragma omp parallel for nowait
  for (i=1; i<N; i++)
  {
      a[i] = a[i-1]+2.0;
```

```
    a[i] = f*a[i-1];
  }
...
```

3. openmp_itc_3.c

```
#include <stdio.h>
#include <math.h>
#include <omp.h>
#define N 1000

int main(int argc, char** argv)
{
  double b[N];
  double s=0;

  #pragma omp parallel for
    for (int i=0; i<N; i++)
    {
      b[i]=i*tan(i*3.14/N);
      s+=b[i];
    }

  printf("%f ",s);
  return 0;
}
```

4. openmp_itc_4.c

```
#include <stdio.h>
#include <math.h>
#include <omp.h>
#define N 1000

int main(int argc, char** argv)
{
  double b[N];
  double s=0;

  #pragma omp parallel for
   for (int i=0; i<N; i++)
   {
     b[i]=i*sqrt(i);
     s+=b[i];
   }

   printf("%f ",s);
   return 0;
}
```

5. openmp_itc_5.c

```
...
 double x[N],y[N];
 x[0] = 0;

 # pragma omp parallel for nowait
   for (i=1; i<N; i++)
   {
     x[i] = x[i-1]*x[i-1];
     y[i] = x[i] /y[i-1];
   }

  y[0]=x[N-1];
...
```

6. openmp_itc_6.c

```
...
  double z[N];
  z[0]=100;
  double h;

  # pragma omp parallel for
    for (i=0; i<N; i++)
    {
      h = z[i-1]*sqrt(i);
      z[i] = h*(i+h);
    }
...
```

7. openmp_itc_7.c

```
...
  double a[N];
  double y,x;

  # pragma omp parallel for
    for (i=0; i<N; i++)
    {
      y = i*sin(i/N*3.14);
      x = i*cos(i/N*3.14);
      a[i] = y+x;
    }
...
```

8. openmp_itc_8.c

```
...
 double a[N],b[N];
 a[0] = 2;

 # pragma omp parallel for nowait
 for (i=1; i<N; i++)
 {
    a[i] = a[i-1]*a[i-1];
    b[i] = a[i] + 1;
 }
  b[0]=a[N-1];
...
```

9. openmp_itc_9.c

```
...
  double x[N],y[N];
  x[0] = 0;
  # pragma omp parallel for nowait
  for (i=1; i<N; i++)
  {
     x[i] = x[i-1]*x[i-1];
     y[i] = x[i] /y[i-1];
   }
   y[0]=x[N-1];
...
```

10. openmp_itc_10.c

```
...
   double a[N];
   a[0]=1;
   double x;
   # pragma omp parallel for
     for (i=1; i<N; i++)
     {
        x = sqrt(i);
        a[i] = x*(i+2*x+1);
     }
...
```

11. openmp_itc_11.c

```
...
   double a[N];
   a[0]=1;
   double w;
   # pragma omp parallel for
     for (i=1; i<N; i++)
     {
        w = sin(i/N*3.14);
        a[i] = w*a[i-1];
     }
...
```

12. openmp_itc_12.c

```
...
   double x[N],y[N];
   x[0] = 0;
   # pragma omp parallel for nowait
   for (i=1; i<N; i++)
   {
     x[i] = x[i-1]*log(i);
     y[i] = x[i] /y[i-1];
    }
    y[0]=x[N-1];
...
```

13. openmp_itc_13.c

```
...
   double a[N];
   a[0]=1;
   double f = 3.14;
   # pragma omp parallel for
     for (i=1; i<N; i++)
     {  a[i] = f*a[i-1]*a[i-1]; }
...
```

14. openmp_itc_14.c

```
...
   double a[N];
   a[0]=1;
   double w;
   # pragma omp parallel for
     for (i=1; i<N; i++)
     {
        w = sin(i/N*3.14);
        a[i] = w*a[i-1];
     }
...
```

2. Разработка и отладка OpenMP-приложения

Задания этой лабораторной работы повторяют задачи, которые предлагались к решению в лабораторной работе № 2 с помощью библиотеки Pthread. Вспомните и проанализируйте программный код, созданный при выполнении этой лабораторной работы. Решите эту же задачу с помощью OpenMP-технологии. Сравните объемы программного кода, сложность программирования многопоточности, отладки, производительности.

Порядок выполнения лабораторной работы

1. Разработать и реализовать алгоритм решения задания с помощью последовательной программы и протестировать его на нескольких примерах.

2. Разработать алгоритм решения задания, с учетом разделения вычислений между несколькими потоками. Избегать ситуаций изменения одних и тех же общих данных несколькими потоками. Составить схему потоков.

3. Реализовать алгоритм в среде OpenMP и протестировать его на нескольких примерах.

Пример

Найти все простые числа в промежутке от нуля до 10000.

Сначала рассмотрим код, написанный с помощью библиотеки Pthread. Для взаимоисключающего доступа к счетчику простых чисел, разделяемой переменной primeCount используется мьютекс cs.

```
#include <stdio.h>
#include <pthread.h>

#define NUM_THREADS 4
#define MAX_NUMBERS 10000
/* Note:  BLOCKSIZE is assumed to be an even number below. */
#define BLOCKSIZE (MAX_NUMBERS / NUM_THREADS)

long primes[MAX_NUMBERS];
int primeCount;

pthread_mutex_t cs;

void * findPrimes ( void * arg )
{
   long tnum   = *(long *)arg;
   long start  = tnum * BLOCKSIZE + 1;
   long end    = tnum * BLOCKSIZE + BLOCKSIZE;
   long stride = 2;
   long number, factor;

   if(start == 1) start += stride;

   for (number = start; number < end; number += stride )
   {
      factor = 3;
      while ( (number % factor) != 0 ) factor += 2;
      if ( factor == number )
      {
         pthread_mutex_lock (&cs);
         primes[ primeCount ] = number;
         primeCount++;
         pthread_mutex_unlock (&cs);
      }
   }
```

```
    return 0;
}
int main()
{
    int i, rc;
    long tnums[NUM_THREADS];
    pthread_t h[NUM_THREADS];

    pthread_mutex_init( &cs, 0);
    primeCount = 0;
    primes[primeCount++] = 2; // handle special case
    printf( "Determining primes from 1 - %d \n", MAX_NUMBERS);
    for ( i = 0; i < NUM_THREADS; ++i)
    {
        tnums[i] = i;
        rc = pthread_create (&h[i], 0, findPrimes, (void *) &tnums[i]);
    }
    for ( i = 0; i < NUM_THREADS; ++i)
    {
        rc = pthread_join (h[i], 0);
    }
    pthread_mutex_destroy (&cs);
    printf( "Found %d primes\n", primeCount );
    return 0;
}
```

Теперь рассмотрим решение этой же задачи с помощью технологии OpenMP. Цикл определения факта, является ли число простым, распараллелен с помощью директивы OpenMP.

```
#include <stdio.h>
#include <math.h>
#include <omp.h>
#define MAX_NUMBERS 10000
int primes[MAX_NUMBERS];
int primeCount;
int main()
{
    primeCount = 0;
    primes[primeCount++] = 2;
    printf( "Determining primes from 1 - %d \n", MAX_NUMBERS);
    #pragma omp parallel for
        for (int i = 3; i < MAX_NUMBERS; i+=2)
        {
            int factor = 3;
            int maxfactor = sqrt(i);
```

```
            while ( ((i % factor) != 0) && (factor<=maxfactor) ) factor+=2;
                if ( factor > maxfactor )
                {
                    #pragma omp critical
                    {
                        primes[ primeCount ] = i;
                        primeCount++;
                    }
                }
        }
    printf( "Found %d primes\n", primeCount );
    return 0;
}
```

Варианты заданий

1. Вычислить произведение матриц A и B. Входные данные: произвольные квадратные матрицы A и B одинаковой размерности. Решить задачу двумя способами: 1) количество потоков является входным параметром, при этом размерность матриц может быть не кратна количеству потоков; 2) количество потоков заранее неизвестно и не является параметром задачи.

2. Найти определитель матрицы A. Входные данные: целое положительное число *n*, произвольная матрица A размерности *n* x *n*. Решить задачу двумя способами: 1) количество потоков является входным параметром, при этом размерность матриц может быть не кратна количеству потоков; 2) количество потоков заранее неизвестно и не является параметром задачи.

3. Найти алгебраическое дополнение для каждого элемента матрицы. целое положительное число *n*, произвольная матрица A размерности *n* x *n*. Решить задачу двумя способами: 1) количество потоков является входным параметром, при этом размерность матриц может быть не кратна количеству потоков; 2) количество потоков заранее неизвестно и не является параметром задачи.

4. Найти обратную матрицу для матрицы A. Входные данные: целое положительное число *n*, произвольная матрица A размерности *n* x *n*. Решить задачу двумя способами: 1) количество потоков является входным параметром, при этом размерность матриц может быть не кратна количеству потоков; 2) количество потоков заранее неизвестно и не является параметром задачи.

5. Определить ранг матрицы. Входные данные: целое положительное число *n*, произвольная матрица A размерности *n* x *n*. Решить задачу двумя способами: 1) количество потоков является входным параметром, при этом размерность матриц может быть не кратна количеству потоков; 2) количество потоков заранее неизвестно и не является параметром задачи.

6. Вычислить прямое произведение множеств A1, A2, A3, A4. Входные данные: множества чисел A1, A2, A3, A4, мощности множеств могут быть не равны между собой и мощность каждого множества больше или равна 1. количество потоков определяется, исходя из мощностей множеств и не является параметром задачи.

7. Вычислить прямое произведение множеств A1, A2, A3… An. Входные данные: целое положительное число *n*, множества чисел A1, A2, A3… An, мощности множеств равны между собой и мощность каждого множества больше или равна 1. Количество потоков определяется, исходя из мощности множеств, и не является параметром задачи.

8. Используя формулы Крамера, найти решение системы линейных уравнений.

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + a_{14}x_4 = b_1 \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + a_{24}x_4 = b_2 \\ a_{31}x_1 + a_{32}x_2 + a_{33}x_3 + a_{34}x_4 = b_3 \\ a_{41}x_1 + a_{42}x_2 + a_{43}x_3 + a_{44}x_4 = b_4 \end{cases}$$

Предусмотреть возможность деления на ноль. Входные данные: коэффициенты системы. Оптимальное количество потоков выбрать самостоятельно.

9. Представить выражение $(ax + by)^n$ в виде

$$C_1a^nx^n + C_2b^ny^n + C_3a^{n-1}x^{n-1}by + C_4b^{n-1}y^{n-1}ax + C_5a^{n-2}x^{n-2}b^2y^2 + C_6b^{n-2}y^{n-2}a^2x^2 + \text{K}$$

Входные данные: числа *a* и *b*, целое положительное число *n*. Количество потоков выбирается в зависимости от показателя степени *n* и не является параметром задачи.

10. Определить, является ли множество C объединением множеств A и B (A U B), пересечением множеств (A I B), разностью множеств A и B (A\B), разностью множеств B и A (B\A). Входные данные: множества целых положительных чисел A, B, C. Количество потоков определяется, исходя из мощностей множеств, и не является параметром задачи.

11. Найти все возможные тройки компланарных векторов. Входные данные: множество не равных между собой векторов (*x, y, z*), где *x*, *y*, *z* – числа. Количество потоков зависит от мощности множества векторов, и не является параметром задачи.

12. Определить, делится ли целое число A, содержащее от 1 до 1000 значащих цифр, на 2, 3, 4, 5, 6, 7, 8, 9, 10. Входные данные: целое положительное число A, записанное в файле. Количество потоков зависит числа A, и не является параметром задачи.

13. Определить индексы i, j ($i \neq j$), для которых выражение A[i] – A[i+1] + A[i+2] – A[i+3] + … ± A[j] имеет максимальное значение. Входные данные:

массив чисел A, произвольной длины большей 10. Количество потоков не является параметром задачи.

14. Определить индексы i, j, для которых существует наиболее длинная последовательность A[i] < A[i+1] < A[i+2] < A[i+3] < … < A[j]. Входные данные: массив чисел A, произвольной длины большей 1000. Количество потоков не является параметром задачи.

15. Определить индексы i1, j1, i2, j2, для которых существует наибольшее количество значений

A[i1,j1], A[i1,j1+1], A[i1,j1+2], … A[i1,j2],
A[i1+1,j1], A[i1+1,j1+1], A[i1+1,j1+2], … A[i1+1,j2],
……..
A[i2,j1], A[i2,j1+1], A[i2,j1+2], … A[i2,j2],

равных между собой. Входные данные: двумерный массив целых чисел A, размерности 5x5. Количество потоков не является параметром задачи.

16. Определить множество индексов i, для которых (A[i] - B[i]) или (A[i] + B[i]) являются простыми числами. Входные данные: массивы целых положительных чисел A и B, произвольной длины ≥ 1000. Количество потоков зависит от размерностей массивов, и не является параметром задачи.

17. Определить множество индексов i, для которых A[i] и B[i] не имеют общих делителей (единицу в роли делителя не рассматривать). Входные данные: массивы целых положительных чисел A и B, произвольной длины ≥ 1000. Количество потоков зависит от размерностей массивов, и не является параметром задачи.

18. Вывести список всех целых чисел, содержащих от 4 до 9 значащих цифр, которые после умножения на *n*, будут содержать все те же самые цифры в произвольной последовательности и в произвольном количестве. Входные данные: целое положительное число *n*, больше единицы и меньше десяти. Количество потоков зависит от размерностей массивов, и не является параметром задачи.

19. Вычислить $\int_a^b f(x)dx$, используя метод трапеций. Входные данные: числа a и b, функция f(x) определяется с помощью программной функции. При суммировании использовать принцип дихотомии.

20. **Задача о восьми ферзях.** Требуется расставить на шахматной доске восемь ферзей таким образом, чтобы ни один ферзь не мог атаковать другого в соответствии с шахматными правилами (т.е. ни один ферзь не должен находиться на одной вертикали, горизонтали или диагонали с другим ферзем). Найти все возможные комбинации.

ЛАБОРАТОРНАЯ РАБОТА № 8

Работа на кластере рабочих станций, работа в среде МВС-1000

Цели и задачи: Работа на кластере рабочих станций, работа в среде МВС-1000. Компиляция и запуск готового приложения, использующего библиотеку MPI.

Время: 2 часа

Для работы в среде МВС-1000 на компьютере, с которого будет осуществляться доступ к кластеру, должна быть установлена telnet-программа с поддержкой протокола ssh. Этот протокол позволяет безопасно передавать данные на удаленную систему по незащищенным каналам передачи данных. Существуют несколько таких программ, например, SecureCRT и PuTTY.

Для трансляции MPI-программы prog.c[1]:

```
mpicc  –o prog  prog.c  <параметры>
```

В качестве параметров этой команды могут использоваться те же ключи, что и в команде запуска стандартного компилятора gcc (GNU C) и имена исходных файлов, подлежащих трансляции.

Рекомендуется использовать опции оптимизации компилятора, например, ключ «-O2». Для удачной линковки, если использовались математические функции из модуля math.h, может потребоваться ключ «-lm».

Не следует создавать большое количество локальных переменных в функции main. Система Linux, имеющая динамический стек, очень необычно ведет себя при запуске программы, требующей большой объем памяти под локальные переменные. Лучше использовать глобальные переменные.

Запуск программы на исполнение с использованием MPICH производится с помощью команды:

```
mpirun [параметры_mpirun...] <имя_программы> [параметры_программы...]
```

Часто используемые параметры команды mpirun:

-h – интерактивная подсказка по параметрам команды mpirun.

-np <число_процессоров> – **ч**исло процессоров, требуемое программе.

-maxtime <максимальное_время> – **м**аксимальное время счета в минутах. По умолчанию – пять часов. От этого времени зависит положение задачи в очереди. По истечении этого времени задача принудительно заканчивается.

-quantum <значение_кванта_времени> – параметр указывает, что задача является фоновой, и задает размер кванта в минутах для фоновой задачи. Для

[1] Двум поддерживаемым вариантам MPI (MPICH и LAM) соответствуют различные варианты команды. По умолчанию программа готовится с использованием MPICH. Если надо подготовить программу для LAM, следует явно указывать путь к соответствующим командам: /usr/bin/mpicc …,

фоновой задачи нужно обязательно указывать и параметр maxtime, иначе она не проживет в системе дольше подразумеваемых пяти часов.

Например, команда

```
mpirun -np 5 -maxtime 30 prog
```

запускает задачу prog в пакетном режиме на пяти процессорах на срок не более получаса, а команда

```
mpirun -np 5 -maxtime 6000 -quantum 200 prog
```

запускает задачу prog на пяти процессорах в фоновом режиме с квантом 3 часа, на общий срок не более 100 часов.

Для запуска программы с использованием LAM служит команда lamrun с теми же параметрами, что и mpirun.

Удачно запущенная задача получает определенный номер, который добавляется к имени задачи (например, prog.1). Это позволяет пользователю запускать одновременно несколько экземпляров задачи с одним и тем же именем – система присвоит каждому экземпляру задачи уникальный номер. По завершении задачи ее номер «освобождается» и будет использован повторно.

При удачном старте система выдаст пользователю некоторую информацию:

- имена свободных узлов в системе на момент запуска задачи;
- имена выделенных под задачу узлов;
- сведения о принятых системой установках по умолчанию.

Завершает выдачу сообщение об удачном старте задачи, причем в сообщении указывается присвоенный задаче номер, например: «Task “prog.1” started successfully». После этого в каталоге, откуда был произведен запуск, появится папка с именем «prog.1», которая будет содержать файлы output и error для данной задачи, а также служебные файлы.

Если задача не запущена, а поставлена системой в очередь, то сообщение будет следующим: «Task “prog.1” was queued»

Если программа ведет себя не так, как предполагалось, может потребоваться отладка. Специальный параллельный отладчик в системе отсутствует. Если программа в состоянии запуститься на одном процессоре, можно поработать с ней с помощью отладчика gdb или rhide. В противном случае единственное средство отладки – это расстановка по программе точек контрольной печати вида:

```
fprintf(stdout, “I am %d, param 1 = %f, param2 = %f”, rank, a,b);
fflush(stdout);
```

где a, b – это интересующие параметры (в данном случае типа float), а rank – переменная, отображающая номер процесса. Последний оператор ну-

жен для того, чтобы принудительно вывести эту строку в выходной файл output задачи. В противном случае весь поток вывода попадет в этот файл только по завершении работы программы.

Все задачи пользователей делятся на три категории:

- *Отладочные задачи* короткие по времени, они запускаются исключительно в целях отладки.
- *Пакетные задачи* средние по времени задачи, они производят реальные расчеты и выполняются, не прерываясь.
- *Фоновые задачи* с большим временем счета, они могут прерываться системой. Для фоновой задачи пользователь должен явно указать *квант* – минимальное время счета фоновой задачи, в течение которого задачу прерывать нельзя.

Планирование очередей в каждый момент времени производится в соответствии с параметрами текущего *режима планирования,* который определяет максимальное время, отведенное для отладочных задач; максимальное время, отведенное для пакетных задач; число процессоров, которые «резервируются» для отладочных задач; шкала приоритетов пользователей.

Командой mfree можно узнать, **с**колько на данный момент времени в системе имеется свободных процессоров

При работе на кластере пользователь имеет возможность просматривать очередь задач, видеть, какие задачи находятся в счете, какие – стоят в очереди. Пользователь имеет возможность удалить свои задачи из очереди или принудительно прервать их счет.

Постановка в очередь задачи осуществляется командами запуска задачи mpirun. По команде mps можно посмотреть все запущенные задачи, при этом стоящие в очереди задачи будут помечены атрибутом queued.

Удаление задачи из очереди производится командой

```
mqdel <имя_задачи.номер_задачи>
```

Безусловное завершение запущенной задачи производится командой

```
mterm [имя_задачи.номер_задачи]
```

Параметром для команды служит имя задачи и – через точку – ее номер. Данная команда допускает задание в качестве параметра маски Unix-формата с использованием символов-джокеров. По этой команде будут завершены все задачи, имена которых удовлетворят заданной маске. Завершить все свои задачи можно командой:

```
mterm ‘*’
```

При отсутствии параметра пользователю будет выдан список всех запущенных задач и предложено ввести номер (по списку) той задачи, которую

нужно завершить. Перед завершением задачи в этом случае будет задан вопрос о полном завершении задачи.

Просмотр очереди осуществляется по команде mqinfo. Данная команда не имеет параметров.

Пользователю предоставляется следующая информация о состоянии системы в целом: время, на которое последний раз было перезаписано состояние очереди, номер текущего расписания, количество отключенных процессоров (если таковые имеются), учетный период (**Accumulation period**), за который производится суммирование времени счета завершившихся задач одного пользователя.

Пользователю предоставляется информация о включенных режимах: дата и время включения режима, общее число планируемых процессоров, количество процессоров, отведенное под отладочные задачи, максимальное время (в минутах) для отладочных задач, максимальное время (в минутах) для пакетных задач, шкала приоритетов.

Пользователю предоставляется информация о выполняющихся в системе задачах имя задачи, имя пользователя, количество оставшихся повторов счета, количество процессоров, занимаемых задачей, остаток времени счета в минутах, квант счета в минутах, количество минут до предполагаемого завершения задачи, предполагаемое время завершения. Информация о стоящих в очереди задачах отличается от информации о считающихся задачах тем, что вместо количества минут до завершения выдается количество минут до предполагаемого старта, а вместо времени завершения — время постановки в очередь.

Проверка нахождения задачи в очереди осуществляется командой

```
mqtest <имя_задачи>
```

Порядок выполнения лабораторной работы

1. Зайти на кластер. Определить сколько в настоящий момент доступно процессоров, сколько и каких задач выполняется или стоят в очереди на выполнение.

2. Скопировать с помощью ftp предложенный файл с программой на кластер. Запустить программу несколько раз для различного числа процессоров. Не забывайте указывать максимальное время порядка минуты, иначе задача будет считаться фоновой и запускаться с низким приоритетом!

3. Получить сведения о запуске задач или постановке их в очередь, при необходимости удалить не прошедшие по каким либо причинам задачи из очереди или системы.

4. Изучить выходной файл и файл ошибок для каждой из выполнившихся задач. Какую информацию к командной строке добавляет загрузчик?

5. Добавьте в программу операторы, позволяющие определить время, затраченное главным процессом на выполнение.

Пример простейшей программы с использованием MPI

```
#include <mpi.h>
#include <stdio.h>

int main( int argc, char **argv )

{
    int size, rank, i;

    MPI_Init( &argc, &argv ); // инициализация MPI-библиотеки
    MPI_Comm_size( MPI_COMM_WORLD, &size );        // определение количества
                                                   // запущенных ветвей
    MPI_Comm_rank( MPI_COMM_WORLD, &rank );        // определение своего ранга

    // ветвь с нулевым рангом сообщает количество запущенных процессов
    if( rank==0 ) printf("Total processes count = %d\n", size );

    // все ветви сообщают свой ранг
    printf("Hello! My rank in MPI_COMM_WORLD = %d\n", rank );

    // Точка синхронизации
    MPI_Barrier( MPI_COMM_WORLD );

    // ветвь с рангом 0 сообщает аргументы командной строки,
    // которая может содержать параметры, добавляемые загрузчиком MPIRUN
    if( rank == 0 )
        for( puts("Command line of process 0:"), i=0; i<argc; i++ )
        printf( "%d: \"%s\"\n", i, argv[i] );

    MPI_Finalize(); // закрытие MPI-библиотеки
    return 0;
}
```

ЛАБОРАТОРНАЯ РАБОТА № 9

Создание простого приложения с помощью библиотеки MPI

Цели и задачи: Изучить различные способы двухточечного обмена в MPI, производные типы данных и операции упаковки и распаковки данных.

Время: 4 часа

Порядок выполнения лабораторной работы

1. Изучить рассмотренный ниже пример. Запустить программу на двух процессорах. Поменять местами строки, помеченные в комментариях (1) и (2), и запустить программу еще раз. Просмотреть выходной файл и файл ошибок.

2. Разработать алгоритм решения задания, с учетом разделения вычислений между несколькими процессорами. Составить схему взаимодействия процессов.

3. Реализовать алгоритм с применением функций двухточечного обмена библиотеки MPI и протестировать его на нескольких примерах. Засечь время выполнения программы для различных наборов данных и различного количества процессоров. Вычислить ускорение и эффективность написанной программы.

4. Заменить все операции обменов в программе на обмены с упакованными данными, с последующей их распаковкой на стороне приема. Выяснить, улучшилась ли эффективность программы. Подумать почему, при необходимости изменить алгоритм или заменить тип обмена.

Пример

Ниже приведен пример программы, которую следует запускать на двух процессорах. Она осуществляет обмен сообщениями в автоматическом режиме. Если система будет иметь достаточно ресурсов (что для такой простой программы высоко ожидаемо), то обмены выполняться без проблем, поскольку **с**ообщение первого дошедшего до оператора MPI_Send() процесса будет с большой долей вероятности буферизовано по инициативе системы, и процесс перейдет к выполнению операции получения MPI_Recv(). Однако такой обмен не совсем безопасен. Если операторы посылки (в коде помечена комментарием (1)) и получения (в коде помечена комментарием (2)) сообщения выполнять в разных циклах длиной, например, в 100000 итераций, то вполне вероятно исчерпание системных ресурсов и, как следствие, зависание программы. Более того, программа гарантированно зависнет, если операции посылки и приема сообщения поменять местами.

```
#include <mpi.h>
#include <stdio.h>

#define leng 20   //length of WR-string

int main( int argc, char **argv )
{
   double t,t2;
   int i,rank,size;
   char WR[leng];

   MPI_Status status;

   MPI_Init( &argc, &argv );
   MPI_Comm_rank( MPI_COMM_WORLD, &rank );
   MPI_Comm_size(MPI_COMM_WORLD,&size);

   sprintf(WR,"Hello from %d",rank); // формирование сообщения

   t=MPI_Wtime();  // фиксация времени «начала посылки»,
                   // локально для каждого процесса
   MPI_Send(WR,leng,MPI_CHAR,size-(rank+1),rank,MPI_COMM_WORLD); // (1)
   MPI_Recv(WR,leng,MPI_CHAR,rank,size-(rank+1),MPI_COMM_WORLD,&status); // (2)
   t2=MPI_Wtime(); // фиксация времени «окончания приема»,
                   // локально для каждого процесса
```

```
    printf("\n From processor %d\n WR=%s\n",rank,WR);    // вывод сообщения
    printf("\n From processor %d\n Time=%le\n",rank,(t2-t)/100); // вывод времени,
                                // затраченного на обмен данным процессором
    MPI_Finalize();
    return 0;
}
```

Варианты заданий

1. **Остров Сокровищ.** Шайка пиратов под предводительством Джона Сильвера высадилась на берег Острова Сокровищ. Не смотря на добытую карту старого Флинта, местоположение сокровищ по-прежнему остается загадкой, поэтому искать клад приходится практически на ощупь. Так как Сильвер ходит на деревянной ноге, то самому бродить по джунглям ему не с руки. Джон Сильвер поделил остров на участки, а пиратов на небольшие группы. Каждой группе поручается искать клад на одном из участков, а сам Сильвер ждет на берегу. Пираты, обшарив свою часть острова, возвращаются к Сильверу и докладывают о результатах. Написать программу, моделирующую поведение пиратской шайки, используя метод передачи информации «точка-точка».

2. **Первая задача про Китайский банк.** На маленькой улице Чжуань-Сю в городе Гонконг живут двести тысяч китайцев и находятся три банка. Каждый из этих банков принимает деньги от вкладчиков в трех валютах – китайских юанях, американских долларах и английских фунтах стерлингов. При этом если вкладчик хочет взять деньги в одном банке на улице Чжуань-Сю и положить в другой, то ему в первом банке выдается только расписка, которую он и относит во второй банк. В пятницу вечером банки подсчитывают, сколько денег и в какой валюте они должны соседям и отправляют инкассаторов отнести эти деньги. Написать программу, моделирующую обмен деньгами в пятницу вечером на улице Чжуань-Сю, используя метод передачи информации «точка-точка».

3. **Вторая задача про Китайский банк.** Решить задачу о китайских банках с дополнительным условием, что все пятничные расчеты банки проводят через Большой Банк, находящийся на улице Чжуань-Го.

4. **Криптономикон.** В секретном институте № 127, находящемся где-то в Сибири, разработана новейшая система шифровки данных для малого бизнеса. Система представляет собой ряд модулей, каждый из которых принимает на вход часть сообщения и шифрует его по определенной схеме. Какую именно часть текста шифрует каждый из модулей, задается специальным ключом, который меняется каждый день. Написать программу, моделирующую работу шифровальной системы. В качестве шифров использовать прямые подстановки (вид шифра, когда каждому символу исходного текста ставится в соответствие какой-то символ шифрованного текста, причем всегда

один и тот же). В качестве ключа использовать подстановку, нижняя строка которой задает номер модуля, текст делится на равные части, весь остаток получает тот модуль, который шифрует последнюю часть. Использовать метод передачи информации «точка-точка».

5. **Криптономикон-2.** Дружественный секретный институт № 721, расположенный где-то на Урале разработал новейшую систему дешифрования новейшей системы шифрования, разработанной в секретном институте № 127, находящемся где-то в Сибири. Решить задачу, обратную задаче 4, написав программу-дешифровщик. Взяв в качестве входного сообщения шифрованный текст и ключ, она должна восстанавливать исходный текст.

6. **Первая задача про деньги.** Десять бухгалтеров сводят годовой баланс. Каждый из них отвечает за свою сторону жизни предприятия – зарплату, материальные расходы и т.п., и точно знает, сколько доходов и расходов числится за ним. Бухгалтера заполняют баланс по очереди – сперва первый из них заполняет свой раздел, затем это делает второй и т.д. В конце концов все попадает на стол главному бухгалтеру и он объявляет результат и везет отчет в налоговую инспекцию. Написать программу, моделирующую поведение бухгалтеров, используя метод передачи информации «точка-точка».

7. **Вторая задача про деньги.** Решить задачу 6, но при условии, что все данные в отчет заносит главный бухгалтер, а сотрудники просто подходят к нему по одному и сообщают свои цифры.

8. **Аукцион.** Известный предприниматель и филантроп Джон Смит непосильным трудом на ниве экономических преступлений нажил огромное состояние и умер. Имущество досталось его сыну Джону Смиту младшему, который, в отличие от отца, отличается отменным здоровьем и любовью к рулетке. Проиграв за полгода заводы, газеты и пароходы, приобретенные отцом, он для поправки дел решает распродать с аукциона вещи из отцовского особняка. Аукцион проводится следующим образом: каждый из участников в тайне от остальных пишет свою цену на специальной карточке и отдает ее распорядителю. Просмотрев карточки, распорядитель объявляет победителя. Написать программу, моделирующую проведение аукциона, используя метод передачи информации «точка-точка».

9. **Масонский заговор.** Темной-претемной ночью в тайном-претайном месте магистры масонских лож собираются для выборов верховного магистра. Исполнив необходимые ритуалы, они приступают к голосованию. Голосование проводится явным методом – каждый из магистров встает и объявляет всем свой выбор. Далее председатель собрания подсчитывает голоса и объявляет победителя. Написать программу, моделирующую проведение выборов, используя метод передачи информации «точка-точка». Проведением ритуалов при решении задачи пренебречь.

10. **Распределенное казино.** Два человека, находящихся в разных городах играют в кости по телефону. Чтобы избежать обмана, они сообщают результаты бросков своему товарищу, который определяет победителя и сообщает каждому из них о результате. Написать программу, моделирующую поведение игроков и посредника. Каждый процесс должен предоставить в виде файла протокол игры с указанием номеров партий, результатов бросков и победителя.

11. **Произведение с дихотомией.** Дан вектор чисел A[n], где n – произвольное число. Требуется найти значение выражения:

$$C = A_1 \times A_2 \times \ldots \times A_n,$$

где A_i – i-й элемент вектора A. Решить задачу, используя метод дихотомии и синхронный обмен сообщениями. Число процессов p, используемых для решении задачи, должно быть не кратно числу n. Свою часть вектора A каждый процесс читает из файла.

12. **Произведение с дихотомией с накоплением.** Дан вектор чисел A[n], где n – произвольное число. Требуется найти значение выражения:

$$C = A_1 \times A_2 \times \ldots \times A_n,$$

где A_i – i-й элемент вектора A. Решить задачу, используя метод дихотомии и синхронный обмен сообщениями. Число процессов p, используемых для решении задачи, должно быть не кратно числу n. Свою часть вектора A каждый процесс читает из файла. Выходной файл должен содержать значение выражения C и все промежуточные произведения $C_i = A_1 \times A_2 \times \ldots \times A_i$.

13. **Магическое число.** Запущено n процессов, каждый из которых знает некоторое магическое число. Требуется обнаружить максимальное из магических чисел и сообщить его пользователю с указанием номера процесса-владельца. Магическое число определить как некоторую произвольную функцию от номера процесса. Решить задачу, используя метод дихотомии и синхронный обмен сообщениями. Максимальное магическое число с номером процесса-владельца в результате должен знать каждый процесс.

14. **Магическое число-2.** Решить задачу 13, используя парадигму «управляющий-рабочий». Максимальное магическое число с номером процесса-владельца в результате должен знать каждый процесс.

15. **Магическое число-3.** Решить задачу 13, используя парадигму «взаимодействующие равные», с симметричным кодом для всех процессов. Максимальное магическое число с номером процесса-владельца в результате должен знать каждый процесс.

16. **Магическое число-4.** Решить задачу 13, используя топологию связи процессов «кольцо». Максимальное магическое число с номером процесса-владельца в результате должен знать каждый процесс.

17. **Гонки.** Несколько процессов независимо друг от друга складывают по 1000000 чисел. Требуется определить номер самого быстрого процесса и время, затраченное каждым процессом на решение задачи. При решении задачи использовать барьерную синхронизацию. Данные считать из общего для всех процессов файла. Можно использовать процесс - посредник.

18. **Охота на медведя.** Племя кроманьонцев охотится на пещерного медведя, который спит в своем логове в одной из пещер. Для этого они разделяются и начинают заглядывать во все пещеры подряд до тех пор, пока один из них не найдет медведя. Так как крики могут разбудить медведя раньше времени, нашедший медведя находит каждого из своих товарищей и шепотом сообщает ему, что медведь найден. Написать программу, моделирующую поведение племени, исключающую коллективные обмены.

ЛАБОРАТОРНАЯ РАБОТА № 10 – 11

Коллективные обмены MPI. Отладка параллельных программ

Цели и задачи: Изучить различные групповые функции MPI. Сравнить реализацию коллективных обменов с помощью групповых функций и с помощью двухточечных обменов.

Время: 4 часа

Порядок выполнения лабораторной работы:

1. Реализовать алгоритм решения задания с использованием функций коллективного обмена в MPI, замерить время выполнения программы, если возможно, вычислить эффективность.

2. Реализовать алгоритм решения задания с использованием функций двухточечного обмена в MPI. Отладить программу. Замерить время выполнения программы, если возможно, вычислить эффективность.

3. Сравнить эффективность реализаций, построить графики, сделать выводы.

Пример

В примере на основе исходного коммуникатора MPI_COMM_WORLD создается два новых коммуникатора, в первый из них входят все четные процессы, а во второй – все нечетные. Затем 0-ой процесс нового коммуникатора, объединяющего все четные процессы рассылает всем процессам своего коммуникатора число 777, а 0-ой процесс нового коммуникатора, объединяющего все нечетные процессы рассылает всем процессам своего коммуникатора число 666.

```
#include <stdio.h>
#include <mpi.h>

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

{
   int rank1, size1, rank2, size2;
   int value;

   MPI_Comm comm1;
   MPI_Status status;

   MPI_Init (&argc, &argv);
   MPI_Comm_rank (MPI_COMM_WORLD, &rank1);
   MPI_Comm_size (MPI_COMM_WORLD, &size1);
   printf ("Мой ранг в  MPI_COMM_WORLD %d \n", rank1);
   printf ("Всего  в  MPI_COMM_WORLD %d процесоов\n", size1);

   MPI_Comm_split (MPI_COMM_WORLD, (rank1%2==0) ? 0 : 1, 0, &comm1);

   MPI_Comm_rank (comm1, &rank2);
   MPI_Comm_size (comm1, &size2);

   if ((rank2==0) && (rank1%2)) value=777;
   if ((rank2==0) && !(rank1%2)) value=666;

   MPI_Bcast (&value, 1, MPI_INT, 0, comm1);

   printf ("Мой ранг в  MPI_COMM_WORLD  %d,
            Мой ранг в  MPI_COMM_WORLD %d,
            broadcasted message %d", rank1, rank2,value);

   MPI_Finalize();
}
```

Варианты заданий

Схемы выполнения групповых операций приведены на рисунках 10.1 и 10.2.

1. Реализовать барьер для n процессов по принципу дихотомии. Эффективность реализации сравнить с функцией MPI_Barrier().

2. Реализовать симметричный барьер для n процессов. Эффективность реализации сравнить с функцией MPI_Barrier().

3. Реализовать рассылку значения n процессам с помощью двухточечных обменов. Эффективность реализации сравнить с функцией MPI_Bcast().

4. Реализовать сбор значений с n процессов с помощью двухточечных обменов (каждый процесс посылает одно значение). Эффективность реализации сравнить с функцией MPI_Gather().

5. Реализовать рассылку массива значений на n процессов по принципу i-ое значение i-му процессу с помощью двухточечных обменов. Эффективность реализации сравнить с функцией MPI_Scatter().

6. Реализовать сбор данных с n процессов с помощью двухточечных обменов (каждый процесс посылает заранее заданное количество значений). Эффективность реализации сравнить с функцией MPI_Gatherv().

7. Реализовать рассылку массива значений на n процессов с помощью двухточечных обменов по принципу: каждому процессу отправляется заранее заданное число значений. Эффективность реализации сравнить с функцией MPI_Scatterv().

8. Реализовать рассылку значений со всех процессов на все с помощью двухточечных обменов. Эффективность реализации сравнить с функцией MPI_Allgather().

9. Реализовать рассылку массивов данных со всех процессов на все с помощью двухточечных обменов. Эффективность реализации сравнить с функцией MPI_Alltoall().

10. Реализовать сбор массива значений от n процессов на 0-ой с помощью двухточечных обменов с выполнением какой-либо операции приведения (сумма, минимум, среднее и пр.). Эффективность реализации сравнить с функцией MPI_Reduce().

10. Реализовать сбор массива значений от n процессов на каждом процессе с помощью двухточечных обменов с выполнением какой-либо операции приведения (сумма, минимум, среднее и пр.) над каждым элементом массива. Эффективность реализации сравнить с функцией MPI_Allreduce().

11. Реализовать распределение массива значений от n процессов с помощью двухточечных обменов с выполнением какой-либо операции приведения (сумма, минимум, среднее и пр.) таким образом, что бы на i-ом процессе оказались приведенные i-ые элементы массива. Эффективность реализации сравнить с функцией MPI_Reduce_scatter().

12. Реализовать процедуру сканирования массива с приведением. Эффективность реализации сравнить с функцией MPI_Scan().

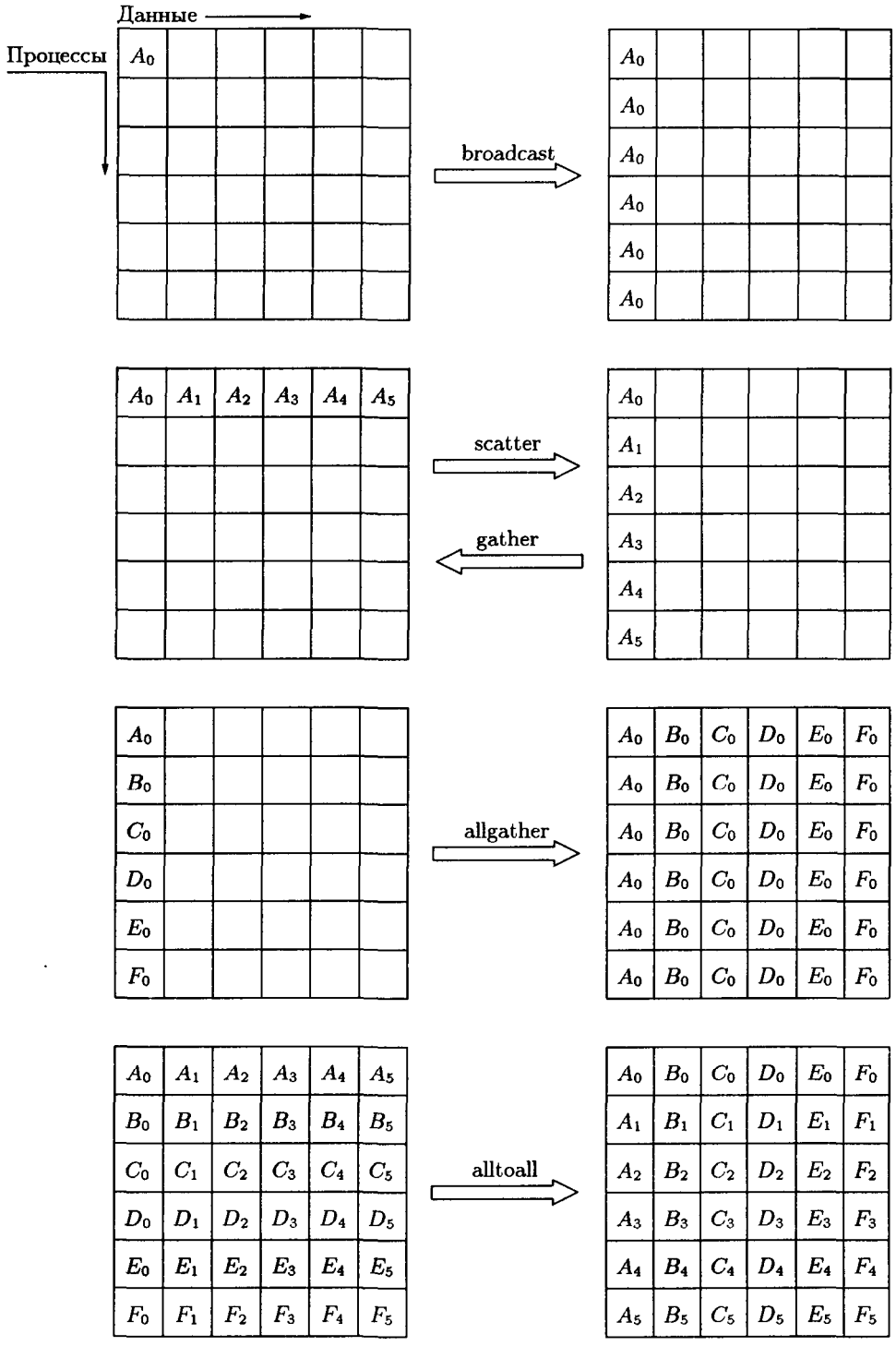

Рис. 10.1. Глобальные функции связи для группы из шести процессов

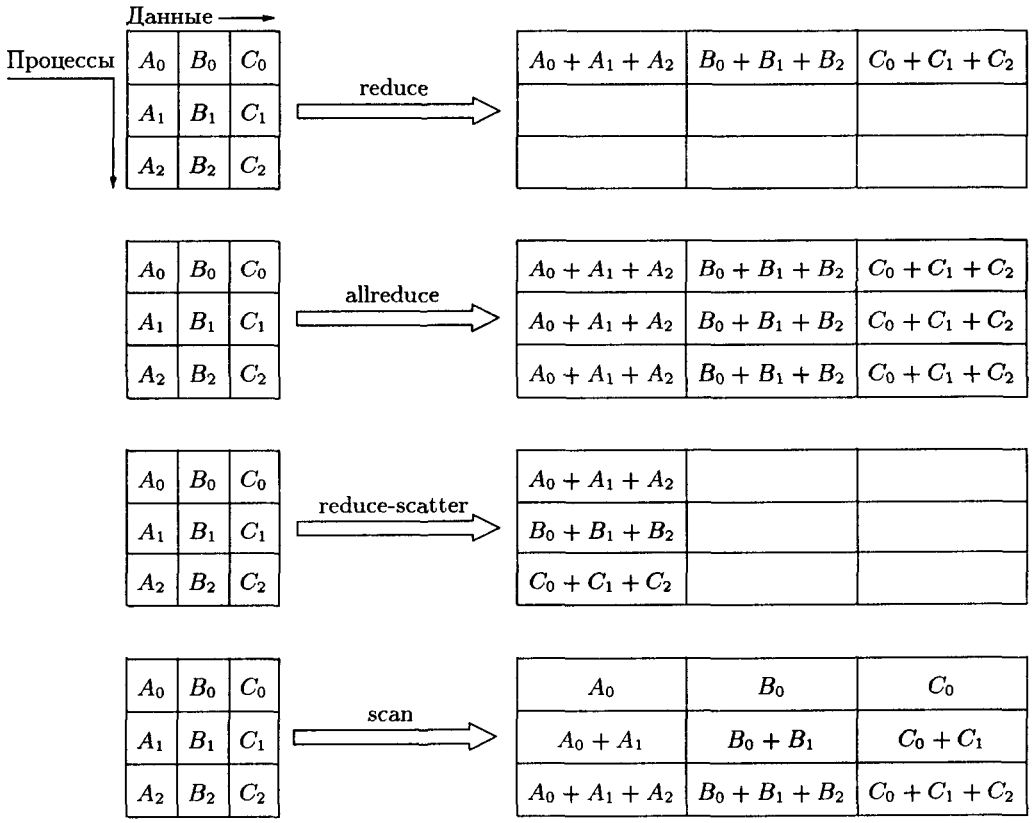

Рис. 10.2. Глобальные операции приведения для группы из трех процессов

13. **Управляемое произведение.** Дан вектор чисел A[n], где n – произвольное число. Требуется найти значение выражения:

$$C = A[1] \times A[2] \times \ldots \times A[n],$$

где A[i] – i-й элемент вектора A. Решить задачу, используя парадигму «управляющий – рабочие» и синхронный обмен сообщениями. Число процессов p, используемых для решении задачи, должно быть не кратно числу n. Свою часть вектора A каждый процесс читает из файла. Решить задачу двумя способами: 1) используя только обмены типа «точка-точка»; 2) используя только коллективные обмены. Проанализировать эффективность алгоритмов.

14. **Управляемое произведение с накоплением.** Дан вектор чисел A[n], где n – произвольное число. Требуется найти значение выражения:

$$C = A_1 \times A_2 \times \ldots \times A_n,$$

где A_i – i-й элемент вектора A. Решить задачу, используя парадигму «управляющий – рабочие» и синхронный обмен сообщениями. Число процессов p, используемых для решении задачи, должно быть не кратно числу n. Свою часть вектора A каждый процесс читает из файла. Выходной файл должен содержать значение выражения C и все промежуточные произведения $C_i = A_1 \times A_2 \times \ldots \times A_i$. Решить задачу двумя способами: 1) используя только обмены

типа «точка-точка»; 2) используя только коллективные обмены. Проанализировать эффективность алгоритмов.

15. **Магическое число.** Запущено n процессов, каждый из которых знает некоторое магическое число. Требуется обнаружить максимальное из магических чисел и сообщить его пользователю с указанием номера процесса-владельца. Магическое число определить как некоторую произвольную функцию от номера процесса. Решить задачу двумя способами: 1) используя только обмены типа «точка-точка»; 2) используя только коллективные обмены. Проанализировать эффективность алгоритмов.

16. **Охота на медведя.** Племя кроманьонцев охотится на пещерного медведя, который спит в своем логове в одной из пещер. Для этого они разделяются и начинают заглядывать во все пещеры подряд до тех пор, пока один из них не найдет медведя. Так как крики могут разбудить медведя раньше времени, нашедший медведя находит каждого из своих товарищей и шепотом сообщает ему, что медведь найден. Решить задачу двумя способами: 1) используя только обмены типа «точка-точка»; 2) используя только коллективные обмены. Проанализировать эффективность алгоритмов.

17. **Распределенное казино.** Два человека, находящихся в разных городах играют в кости по телефону. Чтобы избежать обмана, они сообщают результаты бросков своему товарищу, который определяет победителя и сообщает каждому из них о результате. Написать программу, моделирующую поведение игроков и посредника. Каждый процесс должен предоставить в виде файла протокол игры с указанием номеров партий, результатов бросков и победителя.

Лабораторная работа № 12

Решение прикладных задач с помощью MPI

Цели и задачи: разработать алгоритм, реализовать и отладить параллельную программу для одной из задач линейной алгебры, вычислительной математики, обработки изображений или комбинаторики с помощью библиотеки MPI.

Время: 6 часов

Порядок выполнения лабораторной работы:

1. Разработать алгоритм решения задания параллельный по данным.

2. Реализовать алгоритм решения задания с использованием функций MPI. Отладить программу на нескольких примерах.

3. Замерить время выполнения программы, для разного количества процессоров и разной размерности данных. Вычислить эффективность, построить графики, сделать выводы.

Варианты заданий

1. **Задача о вычислении значения многочлена.** Пусть

$$q(x)=a_0+q_1(x)+x^r q_2(x)+x^{2r} q_3(x)+\ldots+x^{(s-1)r} q_s(x),$$

где $s=2^r$ и

$$q_i(x)=a_k+ \ldots + a_{k+r-1}x^r,\ k=(i-1)r+1.$$

Организовать процесс вычислений следующим образом:

1. Вычислить $x^2, \ldots, x^r$ (последовательно?).
2. Вычислить $q_1(x), \ldots, q_s(x)$ (параллельно).
3. Вычислить $x^r, x^{2r}, \ldots, x^{(s-1)r}$ (последовательно?).
4. Вычислить произведения $x^r q_2(x),\ x^{2r} q_2(x), \ldots, x^{(s-1)r} q_s(x)$ (параллельно).
5. Вычислить сумму $q(x)=a_0+q_1(x)+x^r q_2(x)+x^{2r} q_3(x)+\ldots+x^{(s-1)r} q_s(x)$.

Для простоты считать s также степенью двойки. Распределить нагрузку на процессоры с помощью блочного метода.

2. **Задача о вычислении многочлена-2.** Решить задачу 1, распределяя нагрузку на процессоры с помощью цикличной слоистой схемы.

3. **Задача о вычислении произведения матрицы на вектор.** Дана A – матрица размерностью m строк на n столбцов и дан вектор **x** размерностью n. Исходные данные хранятся в файлах на диске, подготовленные для каждого процессора (написать последовательную программу подготовки данных, входным параметром которой является число процессоров). Написать программу для p процессоров ($2<=p<<m,n$), вычисляющую произведение A**x**, как m скалярных произведений. Распределение нагрузки на процессоры провести наиболее удобным способом.

4. **Задача о вычислении произведения матрицы на вектор-2.** Дана A – матрица размерностью m строк на n столбцов и дан вектор **x** размерностью n. Исходные данные хранятся в файлах на диске, подготовленные для каждого процессора (написать последовательную программу подготовки данных, входным параметром которой является число процессоров). Написать программу для p процессоров ($2<=p<<m,n,\ p\%n,\ p\%m\ !=\ 0$), вычисляющую произведение A**x**, как линейную комбинацию n векторов. Распределение нагрузки на процессоры провести наиболее удобным способом.

5. **Задача о вычислении произведения матриц.** Даны матрицы A размерностью $n1$ строк на $n2$ столбцов и матрица B размерностью $n2$ строк на $n3$ столбцов. Исходные данные хранятся в файлах на диске, подготовленные для каждого процессора (написать последовательную программу подготовки данных, входным параметром которой является число процессоров). Написать программу для p процессоров ($2<=p<<n1,n2,n3,\ p\%n1,\ p\%n2,\ p\%n3\ !=$

0), вычисляющую произведение матриц *AB*, собирающую результат в 0-м процессоре, который сбрасывает его в файл. Распределение нагрузки на процессоры провести наиболее удобным способом. Использовать алгоритм с топологией «кольцо» и парадигмой взаимодействующие равные.

6. **Задача о вычислении произведения матриц-2.** Даны матрицы *A* размерностью *n1* строк на *n2* столбцов и матрица B размерностью *n2* строк на *n3* столбцов. Исходные данные хранятся в файлах на диске, подготовленные для каждого процессора (написать последовательную программу подготовки данных, входным параметром которой является число процессоров). Написать программу для p процессоров (*2<=p<<n1,n2,n3, p%n1, p%n2, p%n3 != 0*), вычисляющую произведение матриц *AB*, собирающую результат в 0-м процессоре, который сбрасывает его в файл. Распределение нагрузки на процессоры провести наиболее удобным способом. Использовать алгоритм с топологией «2D-решетка» и парадигмой управляющий-рабочий (в начале программы 0-й процессор считывает все данные и распределяет по процессорам).

7. **Задача о вычислении произведения матриц-3.** Даны матрицы *A* размерностью *n1* строк на *n2* столбцов и матрица B размерностью *n2* строк на *n3* столбцов. Исходные данные хранятся в файлах на диске, подготовленные для каждого процессора (написать последовательную программу подготовки данных, входным параметром которой является число процессоров). Написать программу для p процессоров (*2<=p<<n1,n2,n3 , p%n1, p%n2, p%n3 = 0*), вычисляющую произведение матриц *AB*, собирающую результат в 0-м процессоре, который сбрасывает его в файл. Распределение нагрузки на процессоры провести наиболее удобным способом. Использовать алгоритм с топологией «3D-решетка» и парадигмой управляющий-рабочий (в начале программы 0-й процессор считывает все данные и распределяет по процессорам).

8. **Решение задачи Дирихле для уравнения Лапласа.** В прямоугольной области 0<=x<=a, 0<=y<=b требуется найти решение уравнения

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = f(x, y)$$

при заданных значениях функции *u* на границах. При аппроксимации производных использовать центральные разности на равномерной квадратной сетке, для численного интегрирования использовать метод Якоби.

Написать программу, решающую задачу на *p* процессорах. Выполнить вычисления для сеток различных размерностей и различного количества процессоров. Исследовать зависимость чистого времени выполнения расчетов от количества процессоров при фиксированной сетке и чистого времени выполнения расчетов от размерности сетки при фиксированном количестве

процессоров $p>=3$. Построить соответствующие графики. Пояснить результаты.

9. **Решение СЛАУ методом Гаусса.** Дана СЛАУ $A\mathbf{x}=F$. Написать программу для p процессоров ($2<=p$), решающую систему методом Гаусса. Размерность системы считать не кратным числу процессоров. Распределить данные по процессорам наиболее удобным способом.

Выполнить вычисления для различных размерностей системы и различного количества процессоров. Исследовать зависимость чистого времени выполнения расчетов от количества процессоров при фиксированной размерности и чистого времени выполнения расчетов от размерности системы при фиксированном количестве процессоров $p>=3$. Построить соответствующие графики. Пояснить результаты.

10. **Решение СЛАУ методом простой итерации.** Дана СЛАУ $A\mathbf{x}=F$. Написать программу для p процессоров ($2<=p$), решающую систему методом простой итерации. Размерность системы считать не кратным числу процессоров. Распределить данные по процессорам наиболее удобным способом. Критерием остановки итерационного процесса является малость в равномерной норме разности решений с двух соседних итераций.

Выполнить вычисления для различных размерностей системы и различного количества процессоров. Исследовать зависимость чистого времени выполнения расчетов от количества процессоров при фиксированной размерности и чистого времени выполнения расчетов от размерности системы при фиксированном количестве процессоров $p>=3$. Построить соответствующие графики. Пояснить результаты.

11. **Решение СЛАУ методом простой итерации-2.** Дана СЛАУ $A\mathbf{x}=F$. Написать программу для p процессоров ($2<=p$), решающую систему методом простой итерации. Размерность системы считать не кратным числу процессоров. Распределить данные по процессорам наиболее удобным способом. Критерием остановки итерационного процесса является малость в равномерной норме невязки численного решения на текущем слое.

Выполнить вычисления для различных размерностей системы и различного количества процессоров. Исследовать зависимость чистого времени выполнения расчетов от количества процессоров при фиксированной размерности и чистого времени выполнения расчетов от размерности системы при фиксированном количестве процессоров $p>=3$. Построить соответствующие графики. Пояснить результаты.

12. **Задача о коммивояжере.** Имеется N городов, которые должен обойти коммивояжер с минимальными затратами. При этом на его маршрут накладывается два ограничения:

- Маршрут должен быть замкнутым, то есть коммивояжер должен вернуться в тот город, из которого он начал движение;
- В каждом из городов коммивояжер должен побывать точно один раз, то есть надо обязательно обойти все города, при этом не побывав ни в одном городе дважды.

Для расчета затрат существует матрица условий, содержащая затраты на переход из каждого города в каждый, при этом считается, что можно перейти из любого города в любой, кроме того же самого (в матрице как бы вычеркивается диагональ). Целью решения является нахождения маршрута, удовлетворяющего всем условиям и при этом имеющего минимальную сумму затрат.

12. **Задача о восьми ферзях.** Требуется расставить на шахматной доске восемь ферзей таким образом, чтобы ни один ферзь не мог атаковать другого в соответствии с шахматными правилами (т.е. ни один ферзь не должен находиться на одной вертикали, горизонтали или диагонали с другим ферзем).

13. **Задача о сглаживании изображения.** Дано черно-белое изображение в виде матрицы чисел-пикселей m X n. Каждый пиксель имеет значение 1 (освещен) или 0 (не освещен). Изображение требуется сгладить, убрав «пики» и «зазубрины». Для этого применяется итеративный алгоритм. Начав с исходного изображения, затемняют все пиксели, у которых как минимум d соседей не освещены. Каждый пиксель рассматривается независимо. Затем эта процедура проделывается с полученным изображением, и повторяется до тех пор, пока изображение не перестанет меняться. Число d выбирается произвольно. Написать программу, реализующий параллельный алгоритм решения для p процессоров. Распределение исходного изображения по процессорам выбрать самостоятельно. Исходное изображение хранится в одном файле. Написать последовательную программу, разрезающую изображение и формирующее файлы с данными для каждого процессора.

14. **Задача о туристе и калориях.** Максимальная выкладка, которую турист может нести на себе 40 кг. Турист Иванов, собираясь в поход, имеет N банок консервов, каждая из которых обладает определенным весом и калорийностью. Турист должен взять рюкзак, вес которого не превышает 40 кг, набитый консервами с максимальной суммарной калорийностью. Написать программу для p процессоров, помогающую туристу укомплектовать рюкзак. Вкусовыми предпочтениями и другими возможными ограничениями пренебречь.