Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования

Томский государственный университет
систем управления и радиоэлектроники

Кафедра автоматизированных систем управления

С.М. Алфёров

ПАРАЛЛЕЛЬНОЕ ПРОГРАММИРОВАНИЕ

Методические указания по самостоятельной и индивидуальной работе студентов всех форм обучения и направлений бакалавриата

Томск
2024

УДК 004.4
ББК 32.97
А-91

Рецензент:
Лукьянов А.К., доцент кафедры Автоматизированных систем управления ТУСУР, к.т.н.

Алфёров, Сергей Михайлович
А-91 Параллельное программирование: методические указания по самостоятельной и индивидуальной работе студентов всех форм обучения направлений бакалавриата 090301 – Информатика и вычислительная техника/ – Томск: ТУСУР, 2024. – 40 с.

Методические указания содержат задания к лабораторным работам, краткий теоретический материал для выполнения заданий и варианты.

Одобрено на заседании каф. Автоматизированных систем управления, протокол №11 от 23.11.2023.

УДК 004.4
ББК 32.97

Оглавление

Лабораторная работа 1. Основные функции MPI....4
Цель работы....4
Лабораторное задание....4
Содержание отчета....4
Методические указания....4
Пример программы вычисления интеграла....5
Варианты заданий. Функции для индивидуальных заданий....7
Лабораторная работа 2. Коллективные функции в MPI....12
Цель работы....12
Лабораторное задание....12
Содержание отчета....12
Методические указания....12
Индивидуальные задания....14
Пример программ. Вычисление максимального и минимального значений каждой строки матрицы....15
Лабораторная работа 3. Производные типы данных в MPI....21
Цель работы....21
Лабораторное задание....21
Содержание отчета....21
Методические указания....21
Индивидуальные задания....22
Лабораторная работа 4. Обработка и выполнение программ в среде OpenMP....24
Цель работы....24
Лабораторное задание....24
Содержание отчета....24
Методические указания....25
Лабораторная работа 5. Синхронизация потоков в OpenMP....26
Цель работы....26
Лабораторное задание....26
Содержание отчета....26
Исполнение параллельных процессов с синхронизацией....26
1 Задача производитель-потребитель....26
2 Методы синхронизации. Задача Производитель–Потребитель....27
3 Синхронизация в задаче Читатели-Писатели....31
4 Обедающие философы....35
5 Спящий брадобрей....35
Средства синхронизации потоков....36
1 Семафоры в потоках....36
2 Взаимоисключение с помощью мьютексов....37
3. Условная переменная....37
4 Замки в OpenMP....38
Обработка программы с использованием потоков....39
Индивидуальные задания....39

Лабораторная работа 1. Основные функции MPI.

Цель работы

Освоить применение основных функций MPI на примере параллельной программы численного интегрирования.

Лабораторное задание

1 Создать в домашнем каталоге папку для лабораторной работы 1. Скопировать в него из каталога pub (ссылка в домашнем каталоге) папку INTEG, содержащую две программы: integi.c, integn.c и файл конфигурации кластера nodes.

2 Для указанной преподавателем программы получить загрузочный модуль и проверить работоспособность программы в режиме последовательного и параллельного выполнения на разном числе процессов. Какой метод численного интегрирования используется в программе?.

3 Разобраться в MPI функциях, используемых в программе. Определить назначение и типы параметров функций. Какие дополнительные функции, кроме шести основных используются в программе?

4 Скопировать программу в новый файл. Произвести замену подынтегральной и первообразной функции на указанные преподавателем функции двух переменных. В программе задать значения переменных для пределов интегрирования и параметра математической функции.

5 Получить загрузочный модуль программы и выполнить его на различном числе процессов. Результаты программы направлять переназначением стандартного вывода в файлы результатов.

6 Скопировать программу с индивидуальной функцией в новый файл. Заменить индивидуальные функции передачи и приема на коллективные функции MPI_Bcast и MPI_Reduce. Выполнить программу с числом процессов, выбранных в п. 5.

7 Скопировать программу п. 6 в другой файл. Заменить в программе назначения пределов интегрирования и параметра функции на ввод их с терминала. Ввод провести в нулевом процессе. Упаковать введенные с терминала пределы интегрирования и параметр функции в буфер, разослать буфер всем процессам и распаковать (функции MPI_Pack и MPI_Unpack). Отладить и выполнить новую программу с тем же набором числа процессов и вводимых данных как в п.5.

8 Провести анализ времен выполнения всех трех программ. Какие выводы и рекомендации можно сделать из этого анализа?

Содержание отчета

В отчет включить:

– цель работы;

– индивидуальное задание (функции);

– полный текст последнего варианта программы для индивидуальной математической функции, коллективными функциями передачи, вводом исходных данных и рассылкой их всем процессам с применением операций упаковки-распаковки (последний вариант);

– перечень использованных в программе функций MPI;

– результаты работы программ п. 5, 6, 7. Для п. 6, 7 для можно привести таблицу значений времен выполнения программ с указанием числа процессов.

Методические указания

В программу с использованием функций MPI необходимо включить директиву – **#include <mpi.h>.**

Для обработки программы используется компилятор **gcc** с оболчкой для MPI – **mpicc**.

Так для программы **myprog.c** следует выполнить команду

mpicc –o myprog –lm myprog.c

Ключи и режимы программы такие же, как в программе gcc.

Для запуска программы с использованием MPI достаточно воспользоваться командой

mpirun –np <число процессов> [-hostfile <имя_файла>] ./<имя программы>

Ключ **hostfile** передает имя файла конфигурации кластера. В лабораторной работе это файл **nodes**. Если ключ –**hostfile** не использовать, все процессы будут выполняться на главном узле кластера.

Для проверки алгоритма программы ее можно выполнить в однопроцессорном режиме обычной командой – **./<имя программы>.**

Пример программы вычисления интеграла

```
/* Parallel calculate of integral for function in MPI
*  Compiling - mpicc -o <name> -lm <name.c>
*  Extcute   -
*  mpirun -np <numb.procs> [-hostfile <name hostfile>] ./<name>
*  Last modification - 19.09.2015
*  Author - Fefelov N.P. TUSUR kaf ASU
*/

#include "mpi.h"
#include <stdio.h>
#include <math.h>

static double f(double a);
static double fi(double a);

void main(int argc, char *argv[])
{
    int done = 0, n, myid, numprocs, i;
    double myfunk, funk, h, sum, x;
    double xl = -0.5,    // low  border
           xh =  0.8;    // high border
    double startwtime, endwtime;
    int  namelen;
    char processor_name[MPI_MAX_PROCESSOR_NAME];
    MPI_Status stats;

    MPI_Init(&argc,&argv);
    MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
    MPI_Comm_rank(MPI_COMM_WORLD,&myid);
    MPI_Get_processor_name(processor_name,&namelen);

    fprintf(stderr,"Process %d on %s\n", myid, processor_name);
    fflush(stderr);

    n = 0;
    while (!done)
    {
```

```
    if (myid == 0)
    {
     printf("Enter the number of intervals (0 quit) ");
     fflush(stdout);
     scanf("%d",&n);

     startwtime = MPI_Wtime();

/* Sending the number of intervals to other nodes */

for (i=1; i < numprocs; i++) {
MPI_Send (&n,                /* buffer               */
          1,                 /* one data             */
        MPI_INT,             /* type                 */
        i,                   /* to which node        */
        1,                   /* tag of message       */
        MPI_COMM_WORLD);     /* common communicator */
     }
 }

 else
 {
MPI_Recv  (&n,               /* buffer               */
          1,                 /* one data             */
        MPI_INT,             /* type                 */
        0,                   /* from which node      */
        1,                   /* tag of message       */
        MPI_COMM_WORLD,      /* common communicator */
        &stats);             /* status of errors     */
 }
    if (n == 0)
        done = 1;
    else
    {
      /* Calculate of integral  */
        h   = (xh-xl) / (double) n;
        sum = 0.0;
        for (i = myid + 1; i <= n; i += numprocs) {
            x = xl + h * ((double)i - 0.5);
            sum += f(x);
        }
        myfunk = h * sum;
        printf("Process %d SUMM %.16f\n", myid, myfunk);

/* Sending the local sum to node 0 */
     if (myid !=0) {
       MPI_Send (&myfunk,    /* buffer               */
                 1,          /* one data             */
                 MPI_DOUBLE, /* type                 */
                 0,          /* to which node        */
                 1,          /* tag of message       */
```

```
                    MPI_COMM_WORLD);  /* common communicator */
          }

          if (myid == 0) {
            funk = myfunk;
            for (i=1; i< numprocs; i++) {
             MPI_Recv (&myfunk,
                       1,
                       MPI_DOUBLE,
                       i,
                       1,
                       MPI_COMM_WORLD,
                       &stats);
             funk += myfunk;
          }

         printf("Integral is approximately  %.16f, Error   %.16f\
n",
           funk, funk - fi(xh) + fi(xl));
      endwtime = MPI_Wtime();
      printf("Time of calculation = %f\n", endwtime-startwtime);
          }
         }
     }
     MPI_Finalize();
}

           /*   Functions   */
static double f(double a){ return cos(a);  }

static double fi(double a){ return sin(a); }
```

Варианты заданий. Функции для индивидуальных заданий

1	$\int_{-0.5}^{2.0} ch(c*x)\,dx = \frac{1}{c} sh(c*x); \quad c = 2.0$
2	$\int_{-0.5}^{1.5} sh(c*x)\,dx = \frac{1}{c} ch(c*x); \quad c = 1.5$
3	$\int_{-0.2}^{1.5} \frac{dx}{cos^2(c*x)} = \frac{1}{c} tg(c*x); \quad c = 0.5$

4	$\int_{-0.2}^{1.5} sin(c*x)\,dx = -\frac{1}{c}cos(c*x);\quad c = 1.2$
5	$\int_{-0.2}^{1.5} cos(c*x)\,dx = \frac{1}{c}sin(c*x);\quad c = 1.7$
6	$\int_{-0.2}^{1.0} \frac{dx}{1+(c*x)^2} = \frac{1}{c}arctg(c*x);\quad c = 0.9$
7	$\int_{-0.2}^{1.0} \frac{dx}{c^2+x^2} = \frac{1}{c}arctg\left(\frac{x}{c}\right);\quad c = 1.5$
8	$\int_{-0.5}^{1.0} \frac{dx}{x^2-c^2} = -\frac{1}{c}arcth\left(\frac{x}{c}\right);\quad c = 1.5$
9	$\int_{-0.5}^{1.5} sin(c*x)*cos(c*x)\,dx = \frac{1}{2c}sin^2(c*x);\quad c = 0.8$
10	$\int_{-0.2}^{2.0} sin^2(c*x)\,dx = 0.5x - \frac{0.25}{c}sin(2c*x);\quad c = 1.2$
11	$\int_{-0.5}^{1.5} cos^2(c*x)\,dx = 0.5x + \frac{0.25}{c}sin(2c*x);\quad c = 0.8$
12	$\int_{-0.5}^{1.0} x*sin(c*x)\,dx = \frac{\sin(c*x)}{c^2} - \frac{x*\cos(c*x)}{c};\quad c = 0.7$

13	$\int_{-1.0}^{0.8} x * cos(c * x)\,dx = \frac{cos(c * x)}{c^2} + \frac{x * sin(c * x)}{c};\quad c = 0.7$
14	$\int_{-0.8}^{2.0} sh^2(c * x)\,dx = \frac{1}{2c} sh(c * x) * ch(c * x) - 0.5x;\quad c = 0.9$
15	$\int_{-0.5}^{1.5} ch^2(c * x)\,dx = \frac{1}{2c} sh(c * x) * ch(c * x) + 0.5x;\quad c = 1.2$
16	$\int_{-0.5}^{2.0} th^2(c * x)\,dx = x - \frac{th(c * x)}{c};\quad c = 0.9$
17	$\int_{0.3}^{1.2} \frac{\sqrt{c^2 - x^2}}{x^2} dx = -\frac{\sqrt{c^2 - x^2}}{x} - \arcsin\left(\frac{x}{c}\right);\ c = 2.5$
18	$\int_{0.4}^{1.5} \frac{x^2}{\sqrt{c^2 - x^2}} dx = -0.5x\sqrt{c^2 - x^2} + 0.5c^2 * \arcsin\left(\frac{x}{c}\right);\ c = 1.6$
19	$\int_{-0.5}^{1.8} tg(c * x)dx = -\frac{1}{c}\ln(\cos(c * x));\ c = 0.7$
20	$\int_{-0.5}^{1.0} x * sh(x * c)\,dx = \frac{x * ch(c * x)}{c} - \frac{sh(c * x)}{c^2};\quad c = 0.8$
21	$\int_{-1.0}^{0,8} x * ch(x * c)\,dx = \frac{x * sh(c * x)}{c} - \frac{ch(c * x)}{c^2};\quad c = 1.2$

22	$\int_{-1.0}^{0.8} tg^2(c*x)dx = \frac{tg(c*x)}{c} - x;\ \ c = 1.5$
23	$\int_{-1.5}^{1.1} x * \exp(c*x)\,dx = \frac{\exp(c*x)}{c^2} * (c*x-1);\ \ c = 0.9$
24	$\int_{-0.5}^{1.5} \frac{dx}{1+\exp(c*x)} = \frac{1}{c}\ln\left(\frac{\exp(c*x)}{1+\exp(c*x)}\right);\ \ c = 1.8$
25	$\int_{0.5}^{2.0} \frac{dx}{x(c*x+1)} = -ln\left(\frac{c*x+1}{x}\right);\ \ c = 1.5$
26	$\int_{-0.8}^{1.5} arcsin\left(\frac{x}{c}\right)dx = x * arcsin\left(\frac{x}{c}\right) + \sqrt{c^2 - x^2};\ c = 2.5$
27	$\int_{-1.2}^{0.8} arccos\left(\frac{x}{c}\right)dx = x * arccos\left(\frac{x}{c}\right) - \sqrt{c^2 - x^2};\ c = 2.8$
28	$\int_{-1.2}^{0.8} \frac{dx}{(c^2+x^2)^2} = \frac{x}{2c^2(c^2+x^2)} + \frac{1}{2c^3} * arctg\left(\frac{x}{c}\right);\ c = 1.4$
29	$\int_{-0.5}^{1.4} \frac{x^2}{\sqrt{c^2-x^2}}dx = -\frac{x}{2}\sqrt{c^2-x^2} + \frac{c^2}{2}arcsin\left(\frac{x}{c}\right);\ c = 2.8$
30	$\int_{0.8}^{2.2} \frac{x}{(cx+1)^2}dx = \frac{1}{c^2(cx+1)} + \frac{\ln(cx+1)}{c^2};\ c = 1.5$

31	$\int_{-0.4}^{0.6} sh(cx) * \sin(cx)\,dx = \frac{ch(cx) * \sin(cx) - sh(cx) * \cos(cx)}{2c};\ c = 1.2$
32	$\int_{-1.2}^{0.8} ch(cx) * \cos(cx)\,dx = \frac{sh(cx) * \cos(cx) + ch(cx) * \sin(cx)}{2c};\ c = 0.9$
33	$\int_{0.5}^{2.5} \frac{\sqrt{c^2 - x^2}}{x} dx = \sqrt{c^2 - x^2} - c * \ln\left(\frac{c + \sqrt{c^2 - x^2}}{x}\right);\ c = 3.2$
34	$\int_{-1.1}^{0.9} x^2 \sin(c * x)\,dx = \frac{2x}{c^2}\sin(c * x) - \left(\frac{x^2}{c} - \frac{2}{c^3}\right)\cos(c * x);\ c = 1.5$
35	$\int_{-1.3}^{1.1} x^2 \cos(c * x)\,dx = \frac{2x}{c^2}\cos(c * x) + \left(\frac{x^2}{c} - \frac{2}{c^3}\right)\sin(c * x);\ c = 1.2$
36	$\int_{-0.4}^{0.6} sh(cx) * \cos(cx)\,dx = \frac{ch(cx) * \cos(cx) + sh(cx) * \sin(cx)}{2c};\ c = 1.2$
37	$\int_{-1.5}^{1.2} ch(cx) * \sin(cx)\,dx = \frac{sh(cx) * \sin(cx) - ch(cx) * \cos(cx)}{2c};\ c = 0.9$
38	$\int_{-0.3}^{2.3} x * \exp(c * x)\,dx = \frac{\exp(c * x)}{c^2}(c * x - 1);\ c = 1.8$

Лабораторная работа 2. Коллективные функции в MPI.

Цель работы

Освоить применение коллективных функций MPI для рассылки и сборки фрагментов массивов по процессам и параллельной их обработки по заданному алгоритму.

Лабораторное задание

1 Для предложенного алгоритма составить и отладить последовательную программу обработки числовых массивов индивидуального задания. Использовать динамическое выделение памяти для массивов (функции malloc(), calloc(), free()). Входные массивы заполнить случайными числами (см. программу genarg, в подразделе далее).

2 Алгоритм обработки оформить внешней функцией.

3 Для параллельной обработки определить размер порции массива для каждого процесса и смещение порции от начала полного массива.

4 В каждом процессе выделить память для размещения порции массива. Функцией MPI_Scatter или MPI_Scatterv распределить исходный массив(ы) на число процессов, выбранных при запуске программы.

5 В каждом процессе выполнить обработку части массива составленной внешней функцией и разместить результаты в массиве порции или в выходных переменных.

6 Собрать в главном процессе окончательные результаты: функции MPI_Gather (MPI_Gatherv) или MPI_Reduce.

7 Вывести окончательные результаты. Для больших выходных массивов достаточно вывести первый и последний элементы.

Содержание отчета

В отчет включить:

- цель работы;
- индивидуальное задание;
- описание применяемого способа конструирования порции данных процесса;
- полный текст параллельной программы;
- результаты работы программы.

Методические указания

Под коллективными операциями в MPI понимаются операции данных, в которых принимают участие все процессы используемого коммуникатора. Причем гарантировано, что эти операции будут выполняться гораздо эффективнее, поскольку MPI-функция реализована с использованием внутренних возможностей коммуникационной среды.

Обобщенная передача данных от одного процесса всем процессам, ведущий процесс (root) передает процессам различающиеся данные:

```
int MPI_Scatter(void *sbuf,int scount,MPI_Datatype stype,
                void *rbuf,int rcount,MPI_Datatype rtype,
                int root, MPI_Comm comm),
```

где

sbuf, scount, stype - параметры передаваемого сообщения (**scount,** определяет количество элементов, передаваемых на каждый процесс),

- **rbuf**, **rcount**, **rtype** - параметры сообщения, принимаемого в процессах,
- **root** – ранг процесса, выполняющего рассылку данных,
- **comm** - коммуникатор, в рамках которого выполняется передача данных.

Обобщенная передача данных от одного процесса всем процессам. MPI_Scatter передает всем процессам сообщения одинакового размера. Если размеры сообщений для процессов могут быть разными, следует использовать функцию MPI_Scatterv:

```
int MPI_Scatterv(void *sbuf, int *scounts, int *displs,
                 MPI_Datatype stype,
                 void *rbuf, int rcount,
                 MPI_Datatype rtype,
                 int root, MPI_Comm comm)
```

sbuf - адрес рассылаемого массива данных.
Начало рассылаемых порций задает массив **displs**, количество элементов в порции задает массив **scounts**.
scounts – целочисленный массив, содержащий количество элементов, передаваемых каждому процессу (индекс равен рангу адресата, длина равна числу процессов в коммуникаторе).
displs – целочисленный массив, содержащий смещения относительно начала массива sbuf (индекс равен рангу адресата, длина равна числу процессов в коммуникаторе).
rbuf - адрес массива, принимающего порцию данных в i- ом процессе.
rcount - размер порции, принимаемой в ранге адресата.
root - ранг процесса, выполняющего рассылку данных.
stype, **rtype** – типы рассылаемых и принимаемых данных.

Обобщенная передача данных от всех процессов одному процессу:

```
int MPI_Gather(void *sbuf,int scount,MPI_Datatype stype,
               void *rbuf,int rcount,MPI_Datatype rtype,
               int root, MPI_Comm comm),
```

где
sbuf, **scount**, **stype** - параметры передаваемого сообщения,
rbuf, **rcount**, **rtype** - параметры принимаемого сообщения,
root – ранг процесса, выполняющего сбор данных,
comm - коммуникатор, в рамках которого выполняется передача данных.

```
int MPI_Gatherv(void *sbuf, int scount, MPI_Datatype stype,
 void *rbuf, int *rcounts, int *displs, MPI_Datatype rtype,
 int root, MPI_Comm comm)
```

sbuf – адрес рассылаемых данных в процессе отправителе.
scount – число рассылаемых данных.
rbuf – адрес буфера в принимающем процессе.
scounts – целочисленный массив, содержащий количество элементов, принимаемых от каждого процесса (индекс равен рангу процесса, длина равна числу процессов в коммуникаторе).
displs – целочисленный массив, смещений относительно начала массива rbuf (индекс равен рангу адресата, длина равна числу процессов в коммуникаторе).
root – ранг принимающего процесса.
stype, **rtype** – типы рассылаемых и принимаемых данных.

Индивидуальные задания

1 Параллельное вычисление полинома по схеме Горнера. Число элементов массива коэффициентов полинома увеличить до значения кратного числу процессоров. Добавленные в начале массива элементы заполнить нулями.

2 Вычислить скалярное произведение двух векторов, распределенных по процессам.

3 Умножение матрицы на вектор. Разделять матрицу по процессам целыми строками. Вектор передавать целиком.

4 Умножение матрицы на вектор. Разделить каждую строку матрицы и вектор по процессам. Обработку всей матрицы поместить в цикл по строкам.

5 Определить математическое ожидание и среднее квадратическое отклонение для массива равномерно распределенных случайных чисел.

6 Для массива равномерно распределенных случайных чисел найти среднее значение и центр размаха.

7 Вычислить Y[i] = a * X[i] + Y[i] (SAXPY) для массива случайных чисел. Определить максимальное и минимальное значения полученного массива.

8 Вычислить Z[i] = a * X[i] + Y[i] (SAXPY) для массива случайных чисел. Определить максимальное и минимальное значения полученного массива.

9 Вычислить поэлементное сложение элементов двух матриц Y = X + Y.

10 Вычислить поэлементное умножение элементов двух матриц Y = X * Y.

11 Для матрицы с четным числом строк сложить элементы соседних строк (0,1; 1,2; и т.д.).

12 Подсчитать сумму положительных и отрицательных элементов массива.

13 Вычислить скалярное произведение соседних пар строк матрицы с четным числом строк (0,1; 2,3; и т.д.).

14 Вычислить среднее арифметическое пар соседних элементов (0,1; 1,2; и т.д). Обеспечить четное число элементов частей массива.

15 Вычислить среднее геометрическое абсолютных значений пар соседних элементов массива (0,1; 2,3; и т.д.). Обеспечить четное число элементов частей массива.

16 Отрицательные элементы массива возвести в квадрат, из положительных элементов извлечь квадратный корень..

17 Получить log(x) для положительных элементов массива и exp(x) для отрицательных. Результаты записать в новый массив.

18 Вычислить среднее арифметическое и среднее геометрическое элементов массива из положительных чисел.

19 Имеется два массива и число X вычислить Y[i] = A[i] *X + B[i] для всех элементов массивов A и B.

20 Переписать элементы каждой строки матрицы в обратном порядке.

21 Определить центр размаха массива, распределенного по процессам.

22 Вычислить полином методом циклической редукции.

23 Получить все частные суммы массива крупноблочным распараллеливанием массива по процессам.

24 Все положительные элементы строки матрицы умножить на первый элемент строки, а отрицательные – на последний элемент строки.

25 Реализовать алгоритм циклической редукции для крупноблочного параллелизма.

26 Найти максимальный и минимальный элементы каждой строки матрицы.

27 Найти сумму элементов каждого столбца матрицы, распределенной по процессам блоками строк.

28 Найти произведение элементов каждого столбца матрицы, распределенной по процессам блоками строк.

Пример программ. Вычисление максимального и минимального значений каждой строки матрицы

```
/* mamimatd.c  -- program for parallel calculate
 *        -- max and min for rows of matrix
 *        -- input matrix, two result vectors and auxilary arrays
 *           are declared as dynamic arrays
 * execute -- mpirun -np numprocs ./mamimatd rows cols
 * Author -- Fefelov N.P., TUSUR, kaf. ASU
 * Last modification -- 3.04.19
*/

#include <stdio.h>
#include <mpi.h>
#include <stdlib.h>
#define  dp 50.0f

     /* Function for init of matrix     */
void init_matrix(int m1, int n1, float *arr1)
    { int i, j;
     for(i=0;i<m1;i++)
     for(j=0;j<n1;j++)
         arr1[i*n1+j] = (float)rand()/RAND_MAX*dp*2.0f - dp;
    }

     /* Function for max and min the each row of matrix */
     /* Matrix  and two vectors   */
void mamirows(int m1, int n1, float *arr1,
                  float *arr2, float *arr3)
          { int i, j; float ma, mi, memb;
     for(i=0;i<m1;i++)
     {    ma = mi = arr1[i*n1+0];
          for(j=1;j<n1;j++)
          { memb = arr1[i*n1+j];
               if (memb < mi) mi = memb;
          else if (memb > ma) ma = memb;
          }
          arr2[i] = mi; arr3[i] = ma;
     }
       }

     /* Function for output matrix   */
  void prnmatr (int m1, int n1, float *arr1, char *zag)
    {int i, j;
     printf("%s\n",zag);
     for (i=0; i<m1; i++)
       {for (j=0; j < n1; j++) printf ("%8.2f",arr1[i*n1+j]);
```

```
     printf("\n");  }; printf("\n");
    }

     /* Function for output vector   */
void prnvect (int n1, float *arr1, char *zag, int rnk)
    {int i, j;
     printf("%s %d\n",zag,rnk);
     for (j=0; j < n1; j++) printf ("%8.2f",arr1[j]);
     printf("\n");
    }

int main(int argc, char **argv)
 {
 int m, n;
 int rank,size;
 int nach,count,i,scol,ost;

    /* Enter m and n from the command line  */
    m = atoi(argv[1]);   n = atoi(argv[2]);

 double time1,time2;
 float *ma;                   /* Input  matrix   */
 float *vmin, *vmax, *smin, *smax; /* Output vectors  */
    /* Auxilary arrays for processes */
 int  *adispls, *acounts, *vcounts, *vdispls;

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

 printf("Proccess %d of %d is started\n",rank,size);

    /* Parts rows for process  */
  count = m / size;  ost = m % size;

if (rank==0)   /* Process 0 - master  */
 {
     /* storage for output vectors */
    smin = (float *) calloc(m,sizeof(float));  // for sequential
    smax = (float *) calloc(m,sizeof(float));  // calculation
    vmin = (float *) calloc(m,sizeof(float));  // for parallel
    vmax = (float *) calloc(m,sizeof(float));  // calculation

     /* storage for the input matrix and its initialization*/
    ma = (float *) calloc(m*n ,sizeof(float));
     init_matrix(m,n,ma);
     prnmatr(m,n,ma,"Input matrix MA");

          /* Sequential calculate  */
     mamirows(m,n,ma,smin,smax);
     prnvect(m,smin,"Sequential minimum SMIN, rank",rank);
```

```
     prnvect(m,smax,"Sequential maximum SMAX, rank",rank);

 time1=MPI_Wtime();  //Time begining of parallel programm

/* auxilary arrays for distributing matrix rows to processes */
  adispls = (int *)malloc(size * sizeof(int));
  acounts = (int *)malloc(size * sizeof(int));
  vdispls = (int *)malloc(size * sizeof(int));
  vcounts = (int *)malloc(size * sizeof(int));

  for(i=0;i < size;i++)
    {
  scol = i < ost ? count+1 : count;
            acounts[i] = scol*n; vcounts[i] = scol;
  nach = i*scol + (i >= ost ? ost : 0);
           adispls[i] = nach*n;  vdispls[i] = nach;
    }
        /*     output of auxilary arrays for test  */
  for (i=0; i<size; i++) printf("%10d", acounts[i]); printf("
acounts\n");
  for (i=0; i<size; i++) printf("%10d", adispls[i]); printf("
adispls\n");
  for (i=0; i<size; i++) printf("%10d", vcounts[i]); printf("
vcounts\n");
  for (i=0; i<size; i++) printf("%10d", vdispls[i]); printf("
vdispls\n");

 }   /* End of work process 0  */

     /* All processes  */
    /* Rows in part of matrix ma for rank */
  scol = rank < ost ? count+1 : count;
    /* Offset (in rows) part for rank in matrix ma */
  nach = rank*scol + (rank >= ost ? ost : 0);

     float *map;         /* pointer on part matrix in rank */
     float *bmin, *bmax; /* pointers on output vectors in rank */
     map =  (float *) calloc(scol*n,sizeof(float));
     bmin = (float *) calloc(scol, sizeof(float));
     bmax = (float *) calloc(scol, sizeof(float));

    /* Distribution of the rows of the matrix to other processes
*/
    MPI_Scatterv(ma,acounts,adispls,MPI_FLOAT,map,
                 scol*n,MPI_FLOAT,0,MPI_COMM_WORLD);

     /* Calculation in the one rank */
     printf("Process %d\n", rank);
     prnmatr(scol,n,map,"Part matrix MA");
     mamirows(scol,n,map,bmin,bmax);
     prnvect(scol,bmin,"Part minimum BMIN, rank",rank);
```

```
    prnvect(scol,bmax,"Part maximum BMAX, rank",rank);

       /* Assembly of partial vectors into master-process  */
   MPI_Gatherv(bmin,scol,MPI_FLOAT,
               vmin,vcounts,vdispls,MPI_FLOAT,0,MPI_COMM_WORLD);

   MPI_Gatherv(bmax,scol,MPI_FLOAT,
               vmax,vcounts,vdispls,MPI_FLOAT,0,MPI_COMM_WORLD);

    /* Storage release of local arrays in each process */
       free(map); free(bmin); free(bmax);

 if (rank==0)  /* Only master-process  */
 {
        /* Results of the parallel calculanions */
    printf("Results of parallel.\n");
    prnvect(m,vmin,"Parallel minimum VMIN, rank",rank);
    prnvect(m,vmax,"Parallel maximum VMAX, rank",rank);

    time2=MPI_Wtime();  //Time ending programm
  printf("\ntime= %f\n",time2-time1);  // Time calculating

    /* Storage release of arrays in master-process */
   free(ma); free(smin); free(smax); free(vmin); free(vmax);
   free(adispls); free(acounts); free(vdispls); free(vcounts);
 }  /* End of work master-process */
 MPI_Finalize();
 return 0;
}
```

Результаты программы вычисления максимального и минимального значений строк матрицы

```
[alpha@asu.local@cluster GLAVMATR]$ mpirun -np 3 ./mamimatd6 10 8
Proccess 1 of 3 is started
Proccess 0 of 3 is started
Proccess 2 of 3 is started

Input matrix MA
   34.02  -10.56   28.31   29.84   41.16  -30.24  -16.48   26.82
  -22.22    5.40   -2.26   12.89  -13.52    1.34   45.22   41.62
   13.57   21.73  -35.84   10.70  -48.37  -25.71  -36.28   30.42
  -34.33   -9.91  -37.02  -39.12   49.89  -28.17    1.29   33.91
   11.26  -20.40   13.76    2.43   -0.64   47.28  -20.75   27.14
    2.67   26.99   -9.98   39.15  -21.67  -14.75   30.77   41.90
  -43.02   44.93    2.60  -41.39  -30.78   16.32   39.02  -15.11
  -43.58  -48.00   -4.23  -43.69  -26.17   47.06   40.22   35.09
  -23.33    3.98  -12.48   26.02    1.25   16.77    3.16  -46.07
   -6.24   43.18   43.08   22.10  -21.57   23.85   14.00  -14.60
```

```
Sequential minimum SMIN, rank 0
  -30.24  -22.22  -48.37  -39.12  -20.75  -21.67  -43.02  -48.00
-46.07  -21.57
Sequential maximum SMAX, rank 0
   41.16   45.22   30.42   49.89   47.28   41.90   44.93   47.06
26.02   43.18
        32        24        24  acounts
         0        32        56  adispls
         4         3         3  vcounts
         0         4         7  vdispls

Process 0
Part matrix MA
   34.02  -10.56   28.31   29.84   41.16  -30.24  -16.48   26.82
  -22.22    5.40   -2.26   12.89  -13.52    1.34   45.22   41.62
   13.57   21.73  -35.84   10.70  -48.37  -25.71  -36.28   30.42
  -34.33   -9.91  -37.02  -39.12   49.89  -28.17    1.29   33.91
Part minimum BMIN, rank 0
  -30.24  -22.22  -48.37  -39.12
Part maximum BMAX, rank 0
   41.16   45.22   30.42   49.89

Process 1
Part matrix MA
   11.26  -20.40   13.76    2.43   -0.64   47.28  -20.75   27.14
    2.67   26.99   -9.98   39.15  -21.67  -14.75   30.77   41.90
  -43.02   44.93    2.60  -41.39  -30.78   16.32   39.02  -15.11
Part minimum BMIN, rank 1
  -20.75  -21.67  -43.02
Part maximum BMAX, rank 1
   47.28   41.90   44.93

Process 2
Part matrix MA
  -43.58  -48.00   -4.23  -43.69  -26.17   47.06   40.22   35.09
  -23.33    3.98  -12.48   26.02    1.25   16.77    3.16  -46.07
   -6.24   43.18   43.08   22.10  -21.57   23.85   14.00  -14.60
Part minimum BMIN, rank 2
  -48.00  -46.07  -21.57
Part maximum BMAX, rank 2
   47.06   26.02   43.18

Results of parallel.
Parallel minimum VMIN, rank 0
  -30.24  -22.22  -48.37  -39.12  -20.75  -21.67  -43.02  -48.00
-46.07  -21.57
Parallel maximum VMAX, rank 0
   41.16   45.22   30.42   49.89   47.28   41.90   44.93   47.06
26.02   43.18

time= 0.000274
```

Программа генерации случайных чисел

```
/* Program generation of random  numbers
*  Name:         genarg
*  Arguments:    number of rand, range, startnum
*  Executions:   genarg number range startnum
*  Example:      genarg 500 100.0 127
*  Numbers:      500, range: (-100.0 - +100.0), startnum 127
*  Output:       output of range random numbers in standard output
*  Autor:        Fefelov N.P
*  Last modification:   21.06.11                */

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

int main(int argc, char *argv[])
{
 int n,i,st;
 double x,dp,rmax,max,min;
 if (argc < 4) {
  fprintf(stderr,"\nERROR!\n");
  fprintf(stderr,"Execution: genarg rand_numbers range start_num\
n");
  fprintf(stderr,"Example:   genarg 500 100.0 10\n");
  fprintf(stderr,"Numbers: 500, range: -100.0 - +100.0, start_num:
10\n");
  fflush(stderr);
  return 3;
 }

 n =  atoi(argv[1]);
 dp = atof(argv[2]);
 st=atoi(argv[3]);
 srand(st);
 for(i=0;i<n;i++)
 {
  x=(float)rand()/RAND_MAX*dp*2.0f-dp;
  printf("%20.10f",x);
 }
 return 0;
}
```

ПРИМЕЧАНИЕ: Программа выводит результаты на экран. Для получения файла последовательности случайных чисел воспользуйтесь переназначением вывода этой программы в файл.

Для генерации массива случайных чисел в программе лабораторной работы вставьте оператор ядра этой программы в свою программу.

Лабораторная работа 3. Производные типы данных в MPI

Цель работы

Освоить применение функций MPI для конструирования производных типов данных и передачи на их основе выбранных частей двумерного массива.

Лабораторное задание

1 Для индивидуального задания выбрать наиболее подходящий способ конструирования производного типа для выборки части массива.

2 Определить аргументы функции конструирования, их представление. Составить алгоритмы вычисления аргументов конструктора производного типа (векторный тип – размер блока и расстояние между блоками; для индексного типа – сконструировать массивы размеров блоков и расстояний каждого блока от начала массива).

3 Организовать в программе две матрицы и массив по размерам упакованного представления данных, для размещения в нем выбранных данных.

4 Составить программу с двумя процессами и использованием производного типа для передачи части исходного массива в другой процесс.

5 В нулевом процессе заполнить одну матрицу целыми числами, начиная с 1. Распечатать ее с указанием ранга процесса. В другом процессе обнулить эту матрицу.

6 Объявить имя производного типа данных. Записать MPI функцию конструирования производного типа, а также функции регистрации и освобождения производного типа.

7 Используя сконструированный производный тип данных, передать данные из нулевого процесса в матрицу другого процесса.

8 Еще раз послать матрицу с производным типом другому процессу. В другом процессе принять посланные данные в массив по размеру выбранных данных базового типа.

9 В другом процессе распечатать полученную матрицу и вектор выбранных элементов.

10 Организовать обратную передачу в нулевой процесс вектора выбранных элементов базового типа данных. В нулевом процессе принять эти данные в другую матрицу с использованием производного типа данных.

11 Вывести полученную другую матрицу с указанием ранга процесса.

ЗАМЕЧАНИЕ: Не забудьте, что для разных пар функций передачи следует использовать различные теги.

Содержание отчета

В отчет включить:

- цель работы;
- индивидуальное задание;
- описание применяемого способа конструирования производного типа;
- полный текст параллельной программы;
- результаты работы программы.

Методические указания

Производные типы данных используются для более компактной передачи данных. Они создаются во время выполнения программы с помощью функций-конструкторов на основе типов данных, существующих к моменту вызова такого конструктора.

Производный тип данных характеризуется последовательностью базовых типов и набором значений смещения относительно начала буфера обмена. Смещения могут быть как

положительными, так и отрицательными, не требуется их упорядоченность. Один элемент данных может встречаться в конструируемом типе многократно.

Создание любого производного типа состоит из четырех действий:

– объявление имени типа (MPI_Datatype имя_типа);
– конструирование типа;
– регистрация типа (MPI_Type_commit);
– аннулирование типа (MPI_Type_free).

Для каждого действия существует MPI функции. Описание их находятся в документации по MPI.

Существует четыре типа конструкторов данных:

– непрерывный (MPI_Type_contiguous);
– векторный (MPI_Type_vector);
– индексный (MPI_Type_indexed);
– структурный (MPI_Type_struct).

Только после регистрации производный тип данных можно использовать в функциях пересылки данных.

Для выбора прямоугольных фрагментов массива (частей строк (столбцов) с блоками одного размера) лучше использовать векторный способ конструирования. Индексный способ подойдет, когда блоки данных неодинаковые.

Для удобства обозрения полученных результатов заполняйте исходную матрицу целыми числами от 1 до M*N. Принимающую в другом процессе матрицу заполнить нулями до приема данных от передающего процесса.

Передачу данных в другой процесс сделать дважды: в получающую матрицу и массив по размеру упакованного типа. В первом случае в функции приема использовать сконструированный тип данных, во втором – базовый тип.

ПРИМЕЧАНИЯ:

1 Для более четкого обозрения размеров блоков и их смещений рекомендуется сделать рисунок входного массива в одну строку (по клеткам), отметить границы строк и выделить штриховкой выбираемые для производного типа клетки.

2 Матрицу из M строк и N столбцов лучше описать как одномерный массив из M* N элементов. Тогда легче будет определить расположение отдельных блоков производного типа данных.

Индивидуальные задания

1 Выбрать нечетные строки исходной матрицы.
2 Выбрать четные строки исходной матрицы.
3 Выбрать левую половину матрицы (по столбцам).
4 Выбрать правую половину матрицы (по столбцам).
5 Выбрать четные столбцы матрицы.
6 Выбрать нечетные столбцы матрицы.
7 Выбрать главную диагональ матрицы.
8 Выбрать побочную диагональ матрицы.
9 Выбрать левый верхний треугольник матрицы.
10 Выбрать правый верхний треугольник матрицы.
11 Выбрать левый нижний треугольник матрицы.
12 Выбрать правый нижний треугольник матрицы.
13 Выбрать левую верхнюю часть матрицы, включая строку K и столбец L.
14 Выбрать правую верхнюю часть матрицы, включая строку K и начиная со столбца L.
15 Выбрать левую нижнюю часть матрицы, начиная со строки K и по столбец L.
16 Выбрать правую нижнюю часть матрицы, начиная со строки K и столбца L.

17 Выбрать прямоугольную часть матрицы, начиная с координаты (K,L) и по координату (P,Q).

18 Составить новую матрицу из черных клеток шахматной доски M*N.

19 Создать новую трехдиагональную матрицу.

20 Включить в новую матрицу каждый третий столбец старой матрицы.

21 Включить в новую матрицу каждую третью строку старой матицы.

22 Создать новую матрицу из трех средних столбцов матрицы с нечетным числом столбцов

23 Создать новую матрицу из четырех средних столбцов матрицы с четным числом столбцов.

24 Из матрицы M*N с нечетным числом строк и столбцов сформировать новую матрицу, в которой будут заполнены средняя строка и столбец исходной матрицы (крест).

25 Из матрицы M*N сформировать новую матрицу, в которой будут выбраны только крайние строки и столбцы исходной матрицы (рамка).

26 Из квадратной матрицы M*M получить квадратную матрицу, в которой будут заполнены только главная и побочная диагонали (диагональный крест).

27 Из матрицы M*N с нечетным числом строк и столбцов получить новую матрицу с заполненными нечетными строками и столбцами (решетка).

28 В квадратной матрице с нечетным числом строк и столбцов выбрать треугольную часть матрицы с вершиной в центре матрицы и основанием в последней строке.

29 В квадратной матрице с нечетным числом строк и столбцов выбрать треугольную часть матрицы с вершиной в центре матрицы и основанием в первом столбце..

30 В квадратной матрице с нечетным числом строк и столбцов выбрать треугольную часть матрицы с вершиной в центре матрицы и основанием в первой строке.

31 В квадратной матрице с нечетным числом строк и столбцов выбрать треугольную часть матрицы с вершиной в центре матрицы и основанием в последнем столбце.

32 Из матрицы M*N с четным числом строк и столбцов выбрать из первой строки правую половину строки, из второй – левую половину и т.д.

Лабораторная работа 4. Обработка и выполнение программ в среде OpenMP

Цель работы

Освоить применение основных директив, функций и переменных окружения OpenMP на примере параллельной программы численного интегрирования.

Лабораторное задание

1 Последнюю программу лабораторной работы вычисления интеграла: integi.c или integn.c. скопировать в новый файл.

2 Проанализировать функции MPI для включении аналогичных директив и функций OpenMP в программу.

3 Убрать из программы функции MPI и сделать последовательную программу. Отметить места функций MPI для последующего включения аналогичных конструкций OpenMP.

4 Получить загрузочный модуль и выполнить последовательную программу. Проверить совпадение результатов с параллельной программой.

5 Спланировать параллельные регионы программы. Определить набор общих и локальных переменных. Назначить переменную для редукции.

6 Добавить в программу директивы parallel и for с необходимыми параметрами и типами переменных. Распределение итераций цикла не планировать. Полученные в отдельных потоках частные суммы собрать переменной редукции в общую сумму для значения интеграла.

7 Обработать и выполнить параллельную программу OpenMP с числом процессов, установленных в системе.

8 Включить в программу функции OpenMP и вывести значения переменных интерфейса:

максимально возможное число потоков - **omp_get_max_threads()**,

установка числа потоков в параллельной области - **omp_set_num_threads(n)** ,

определение числа потоков в параллельной области - **omp_get_num_threads()**,

номер каждого потока параллельного региона - **omp_get_threads_num()**,

время работы программы - **omp_get_wtime()**.

Определение числа потоков в параллельной области должно выполняться в главном потоке параллельной области (в директиве **master**).

9 Включить в директиву FOR параметр распределения итераций (schedule). Размер блока итераций (chunk) выбрать таким, чтобы в каждом потоке выполнялось несколько частей итераций цикла. Сравнить время выполнения программы в статическом (static) и динамическом (dynamic) режимах.

10 Доработать программу назначением числа нитей параллельной части программы параметром командной строки при запуске программы.

11 Выполнить программу для разного числа нитей и записи результатов в текстовые файлы. Попробуйте менять значения параметра распределения итераций в цикле.

Содержание отчета

В отчет включить:

- цель работы;
- индивидуальное задание (функции);
- полный текст параллельной программы с функциями OpenMP;

– перечень использованных в программе директив и функций OpenMP;

– результаты работы программы для различного числа потоков, способов планирования итераций цикла и точности вычислений.

Методические указания

1. Обработка и выполнение программы

В программу необходимо поместить директиву – **#include <omp.h>.**

Для компиляции программы используется команда:

gcc –fopenmp -lm –o имя_исп_прогр имя_исх_прогр

остальные необходимые ключи – как в компиляторе **gcc.**

Выполнение программы производится командой

./имя_исп_прогр [аргумент (число нитей в параллельной области) …]

2. Способы задания числа нитей в параллельной области

Число потоков для выполнения параллельных частей программы в операционной системе определяется переменной окружения OMP_NUM_THREADS. Вывести ее значение можно командой LINUX

export | grep OMP_NUM_THREADS

Для задания числа нитей в программе до образования параллельной области используется функция

omp_set_num_threads(n).

В директиве **parallel** также можно задать число нитей параметром **num_threads(n)**.

Приоритет способов задания числа нитей: директива **parallel**, функция **omp_set_num_threads**, переменная окружения **OMP_NUM_THREADS.**

Номер потока определяется функцией **tr = omp_get_thread_num()**

Лабораторная работа 5. Синхронизация потоков в OpenMP

Цель работы

Освоить методы синхронизации в параллельных программах в задаче Производитель-Потребитель и других задачах, выполняемых на множестве параллельных секций в среде OpenMP.

Лабораторное задание

1 Проанализировать индивидуальное задание и создать макет программы с необходимым числом секций в параллельной области на основе шаблона индивидуального задания.

2 Алгоритмы секций оформить функциями. Если несколько секций выполняют одинаковый алгоритм, передать в функцию номер секции. Остальные переменные, используемые в функции сделать глобальными.

3 Ввести в программу и в секции предложенные в задании средства синхронизации.

4 Ограничить выполнение программ секций числом итераций . Число итераций вводить с терминала или передать в главную программу через аргумент командной строки..

5 В программе Производитель – Потребитель (две секции в параллельной области) в каждой итерации вычислять новое значение передаваемой переменной (привязать значение к номеру итерации).

6 При необходимости включить в алгоритмы задержку на случайный интервал времени: **sleep(rand() % t + 1)**, задержка от 1 до t секунд.

7 В каждой секции выводить сведения о старте и окончании алгоритма с указанием назначения секции, номера итерации, полученного значения и другую необходимую информацию. Все данные вывести в одну строку для удобства обозрения результатов.

8 Получить два варианта программы: без синхронизации и с синхронизацией.

9 Выполнить обе программы с выводом на экран и файл результатов. Сравнить полученные данные.

Содержание отчета

В отчет поместить
– цель работы;
– индивидуальное задание;
– тексты программ;
– распечатку результатов.

Исполнение параллельных процессов с синхронизацией

1 Задача производитель-потребитель

Два параллельных процесса выполняют совместно общую работу. Первый (производитель) формирует данные и помещает их в общий буфер, другой (потребитель) берет эти данные из общего буфера и обрабатывает. Оба процесса должны работать поочередно, начинает работу производитель. Общий буфер является критическим ресурсом. Необходимо применить к нему операции взаимоисключения и синхронизации поочередного обращения к общему буферу.

2 Методы синхронизации. Задача Производитель–Потребитель

2.1 Взаимоисключение с чередованием процессов (шаблон программы)

```
var NP: 1,2;

    procedure PROC1;                    procedure PROC2;
    begin                               begin
    while (true) do                     while (true) do
    begin                               begin
     while NP=2 do;                      while NP=1 do;
     критический участок 1;              критический участок 2;
     NP:=2;                              NP:=1;
     независимая часть 1;                независимая часть 1;
    end                                 end
    end;                                end;
begin
   NP:=1;
cobegin
  PROC1;
  PROC2;
coend;
end.
```

2.2 Синхронизация двумя блокирующими переменными. Одиночный буфер

Здесь используются две общих переменных p для производителя и c для потребителя. Условие синхронизации процессов будет таким:

```
c  <=  p  <=  c+1
```

Ниже приведен шаблон программы. Производится передача элементов массива.

```
int buf, iters;
 void* sender ()                 void* reciver ()
 {                               {
   int a[iters];                   int b[iters];
   while (p < iters){              while (c < iters){
     while (p != c) ;                while (p <= c) ;
     a[p] = создать;                 b[c] = buf;
     buf = a[p];                     обработать b[c];
        p = p + 1;                   c = c + 1;
    }                               }
   return NULL;                     return NULL;
 }                               }
int main ( int argc, char *argv[]){
 p = 0; c = 0;
 cobegin   // Блок параллельного выполнения
       sender;
       receiver;
 coend;
 return 0;
}
```

2.3 Синхронизация двумя семафорами. Одиночный буфер

```
var Buf:Record;
Start,Finish:Semaphore;

procedure PRODUSER;
VAR Rec:Record;
begin
   создать запись;
      P(Finish);
Write(Rec,Buf);
      V(Start);
end;

procedure CONSUMER;
VAR Rec:Record;
begin
       P(Start);
   Read(Rec,Buf);
       V(Finish);
обработать запись;
end;

begin
Start.C:=0; Finish.C:=1;
cobegin
  Repeat PRODUSER Until FALSE;
  Repeat CONSUMER Until FALSE;
coend;
end.
```

2.4 Синхронизация двумя семафорами и мьютексом. Множественный буфер

```
VAR Buf:array [1..N] of Record;
    Full,Empty,S:Semaphore;

    procedure PRODUSER;
    VAR Rec:Record;
    ip: integer; ip=0;
    begin
       создать запись;
          P(Empty);
          P(S);
    Write(Rec,Buf[ip]);
          V(S);
    ip = (ip+1) mod n;
          V(Full);
    end;

    procedure CONSUMER;
    VAR Rec:Record;
    ic:integer; ic=0;
    begin
          P(Full);
          P(S);
    Read(Rec,Buf[ic]);
          V(S);
    ic = (ic+1) mod n;
          V(Empty);
    обработать запись;
    end;

begin
S.C:=1; Full.C:= 0; Empty.C:= N;
cobegin
  Repeat PRODUSER Until FALSE;
  Repeat CONSUMER Until FALSE;
coend;
end.
```

2.5 Несколько производителей один потребитель

```
// Производственная линия
#include <stdio.h>
#include <unistd.h>
```

```
#include <semaphore.h>
#include <pthread.h>
sem_t semA, semB, semC, semAB;
void* createA (void* argv){
    while(1){
    sleep(1);
    sem_post(&semA);
    printf("произвел A\n");
    }
}
void* createB (void* argv){
    while(1){
    sleep(2);
    sem_post(&semB);
    printf("произвел B\n");
    }
}
void* createC (void* argv){
    while(1){
    sleep(3);
    sem_post(&semC);
    printf("произвел C\n");
    }
}
void* createAB (void* argv){
    while(1){
    sem_wait(&semA);
    sem_wait(&semB);
    sem_post(&semAB);
    printf("собрал AB\n");
    }
}
void* createWidget (){
    while(1){
    sem_wait(&semAB);
    sem_wait(&semC);
    printf("собрал ABC\n");
    }
}
int main(){
    pthread_t threadA;
    pthread_t threadB;
    pthread_t threadC;
    pthread_t threadAB;

    sem_init(&semA, 0, 0);
    sem_init(&semB, 0, 0);
    sem_init(&semC, 0, 0);
    sem_init(&semAB, 0, 0);

    pthread_create( &threadA, NULL, createA, NULL);
    pthread_create( &threadB, NULL, createB, NULL);
    pthread_create( &threadC, NULL, createC, NULL);
    pthread_create( &threadAB, NULL, createAB, NULL);
    createWidget();
}
```

2.6 Решение задачи «производитель-потребитель» с использованием условных переменных (фрагмент)

```
int full;
pthread_mutex_t mx;
pthead_cond_t cond;
int data;
```

```
void *producer(void *) {
 while(1) {
  int t=produce();
  pthread_mutex_lock(&mx)
  while(full) {
   pthread_cond_wait(&cond, &mx);
  }
  data=t;
  full=1;
  pthread_mutex_unlock(&mx);
  pthread_cond_signal(&mx);
 }
 return NULL;
}
```

```
void *producer(void *) {
 while(1) {
  int t=produce();
  pthread_mutex_lock(&mx)
  while(!full) {
   pthread_cond_wait(&cond, &mx);
  }
  data=t;
  full=1;
  pthread_mutex_unlock(&mx);
  pthread_cond_signal(&mx);
 }
 return NULL;
}
```

2.7 Алгоритм Деккера

```
VAR C1,C2:Boolean;
       NP:1,2;
```

```
procedure PROC1;
begin
while (true) do
begin
C1:=TRUE;
while C2 do
  If NP=2 then
    begin
      C1:=FALSE;
      While NP=2 do;
C1:=TRUE;
    end;
критический участок 1;
NP:=2;
   C1:=FALSE;
независимая часть 1;
end
end;
```

```
procedure PROC2;
begin
while (true) do
begin
C2:=TRUE;
while C1 do
  If NP=1 then
    begin
      C2:=FALSE;
      While NP=1 do;
C2:=TRUE;
    end;
критический участок 2;
NP:=1;
   C2:=FALSE;
независимая часть 2;
end
end;
```

```
begin
   NP:=1;
   C1:=FALSE; C2:=FALSE;
cobegin
  PROC1;
  PROC2;
coend;
end.
```

2.8 Алгоритм Петерсона

```
VAR C1,C2:Boolean;
      NP:1,2;

procedure PROC1;                    procedure PROC2;
begin                               begin
while (true) do                     while (true) do
begin                               begin
C1:=TRUE;                           C2:=TRUE;
NP:=2;                              NP:=1
while (C2 and NP=2)do;              while (C1 and NP=1)do;
критический участок 1;              критический участок 2;
   C1:=FALSE;                          C2:=FALSE;
независимая часть 1;                независимая часть 2;
end                                 end
end;                                end;


begin
   C1:=FALSE; C2:=FALSE;
cobegin
  PROC1;
  PROC2;
coend;
end.
```

3 Синхронизация в задаче Читатели-Писатели

3.1 Синхронизация с приоритетом читателей

```
VAR Nrdr: integer; W,R: Semaphore;
procedure READER;                   procedure WRITER;
begin                               begin
 P(R);                               P(W);
 Nrdr:=Nrdr+1;                       Писать данные;
 If Nrdr = 1 then P(W);              V(W);
 V(R);                              End;
 Читать данные;
 P(R);
 Nrdr:=Nrdr-1;
 If Nrdr = 0 then V(W);
 V(R);
end;
Begin
 Nrdr:=0; W.C:=1; R.C:=1;
 cobegin
  Repeat READER Until FALSE;
  . . .
  Repeat READER Until FALSE;
  Repeat WRITER Until FALSE;
  . . .
```

```
  Repeat WRITER Until FALSE;
 coend;
end.
```

3.2 Синхронизация с приоритетом писателей

```
VAR Nrdr: integer; W,R,S: Semaphore;

procedure READER;
begin
 P(S);
 P(R);
 Nrdr:=Nrdr+1;
 If Nrdr = 1 then P(W);
 V(S);
 V(R);
 Читать данные;
 P(R);
 Nrdr:=Nrdr-1;
 If Nrdr = 0 then V(W);
 V(R);
end;

procedure WRITER;
begin
 P(S);
 P(W);
 Писать данные;
 V(S);
 V(W);
End;

Begin
Nrdr:=0; W.C:=1; R.C:=1; S.C:=1;
cobegin
  Repeat READER Until FALSE;
  . . .
  Repeat READER Until FALSE;
  Repeat WRITER Until FALSE;
  . . .
  Repeat WRITER Until FALSE;
coend;
end.
```

3.3 Решение задачи читателей/писателей с использованием семафоров (приоритетная запись)

```
int readcount, writecount;
semaphore x = 1, y=1, z=1,
rsem = 1, wsem = 1;
void reader() {
 while(true) {
  wait(z);
  wait(rsem);
  wait(x);
  readcount++;
  if (readcount==1)
   wait(wsem);
  signal(x);
  signal(rsem);
```

```
void writer() {
 while(true) {
  wait(y);
  writecount++;
  if (writecount==1)
   wait(rsem);
  signal(y);
  wait(wsem);
  WRITEUNIT();
```

```
  signal(z);                          signal(wsem);
  READUNIT();                         wait(y);
  wait(x);                            writecount--;
  readcount--;                        if (writecount==0)
  if (readcount==0)                    signal(rsem);
   signal(wsem);                      signal(y);
  signal (x);                        }
 }                                  }
}

void main ()
{ readcount = writecount = 0;
  parbegin(reader, writer);
}
```

3.4 Решение задачи читателей/писателей с использованием семафоров (честное распределение ресурсов)

```
#include <pthread.h>
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <semaphore.h>
#define M 4 //num of WR
#define N 3 //num of RE
unsigned int iter; //iteration
sem_t accessM, readresM, orderM; //sem.
unsigned int readers = 0; //Number of readers accessing the
resource
void *reader(void *prm)
{
 int num1=*(int*)prm;
 int i=0,r;
 for(i;i<iter;i++) {
  if (sem_wait(&orderM)<=0)
   // printf("%d Читатель %d в очереди_________Ч%d\
n",i,num1,num1);
   //Remember our order of arrival
   sem_wait(&readresM); // We will manipulate the readers counter
  if (readers == 0)    //If there are currently no readers (we
came first)...
    sem_wait(&accessM); // ...requests exclusive access to the
resource for readers
   readers++;           // Note that there is now one more reader
   sem_post(&orderM);  // Release order of arrival semaphore (we
have been served)
   sem_post(&readresM);// We are done accessing the number of
readers for now
   printf("%d Работает читатель %d_________________Ч%d\n",
i,num1,num1);// Here the reader can read the resource at will
   r=1+rand()%4;
   sleep(r);
```

```
   sem_wait(&readresM);// We will manipulate the readers counter
   readers--;          // We are leaving, there is one less reader
   if (readers == 0)   // If there are no more readers currently
reading...
    sem_post(&accessM); // ...release exclusive access to the
resource
   sem_post(&readresM); // We are done accessing the number of
readers for now
  }
}
void *writer(void *prm)
{
 int num2=*(int*)prm;
 int j=0,r;
 for(j;j<iter;j++) {
  if(sem_wait(&orderM)<=0) //printf("%d Писатель %d в
очереди__________П%d\n",j,num2,num2); // Remember our order of
arrival
   sem_wait(&accessM);     // Request exclusive access to the
resource
  sem_post(&orderM);       // Release order of arrival semaphore
(we have been served)
  printf("%d Работает писатель %d_________________П%d\n",
               j,num2,num2);    // Here the writer can modify the
resource at will
  r=1+rand()%5;
  sleep(r);
  sem_post(&accessM);     // Release exclusive access to the
resource
 }
}
void main(){
 pthread_t threadRE[N];
 pthread_t threadWR[M];
 sem_init(&accessM,0,1);
 sem_init(&readresM,0,1);
 sem_init(&orderM,0,1);
 printf("Введите количество итераций: ");
 scanf("%d",&iter);
 printf("Iter                          ОЧЕРЕДЬ/ВЫПОЛНЕНИЕ\n");
 int i;
 for(i=0;i<M;i++) {
  pthread_create(&(threadWR[i]),NULL,writer,(void*)&i);
 }
 for(i=0;i<N;i++) {
  pthread_create(&(threadRE[i]),NULL,reader,(void*)&i);
 }
 for(i=0;i<N;i++) {
  pthread_join(threadRE[i],NULL);
 }
 for(i=0;i<M;i++) {
```

```
  pthread_join(threadWR[i],NULL);
 }
 sem_destroy(&accessM);
 sem_destroy(&readresM);
 sem_destroy(&orderM);
}
```

4 Обедающие философы

```
// Семафоры доступа к вилкам
Semaphore fork[5] = { 1, 1, 1, 1, 1 };
// Поток-философ (для всех философов одинаковый)
Philosopher(i)
{
  // i – номер философа
  while (1) {
    <Размышление>
    P(fork[i]); // Доступ к левой вилке
    P(fork[(i+1)%5]); // Доступ к правой вилке
    <Питание>
    // Освобождение вилок
    V(fork[i]);
    V(fork[(i+1)%5])
 }
}
Begin cobegin
     Repeat Philosopher(1) Until FALSE;
     Repeat Philosopher(2) Until FALSE;
     Repeat Philosopher(3) Until FALSE;
     Repeat Philosopher(4) Until FALSE;
     Repeat Philosopher(5) Until FALSE;
coend; end.
```

В другом варианте можно изменить порядок взятия вилок для одного из философов (например, для четвертого), который должен брать сначала правую вилку, а только затем левую. Изучение нового варианта алгоритма синхронизации показывает, что он гарантирует отсутствие тупиков.

5 Спящий брадобрей

```
#define CHAIRS 5      /*Количество стульев для посетителей*/
typedef int semafore
semafore customers=0; /*Количество ожидающих посетителей*/
semafore barbers=0;   /*Количество брадобреев ждущих клиентов*/
semafore mutex=1;     /*Для взаимного исключения*/
int waiting=0;        /*Ожидающие (не обслуживаемые) посетители*/
void barber(){
 while (true){
  down(&customers);  /*Если посетителей нет, то ждать*/
  down(&mutex);      /*Запрос доступа к waiting*/
  waiting=waiting-1; /*Уменьшение числа ожидающих посетителей*/
  up(&barbers);      /*Один брадобрей готов к работе*/
  up(&mutex);        /*Отказ доступа к waiting*/
```

```
  cut_hair();     /*Клиента обслуживают (вне критической области)*/
 }
}
void customer(){
 down(&mutex);          /*Вход в критическую область*/
 if (waiting<CHAIRS){ /*Если есть свободные стулья, то ...*/
  waiting=waiting+1;  /*Увеличить число ожидающих клиентов*/
  up(&customers);     /*При необходимости разбудить брадобрея*/
  up(&mutex);         /*Отказ от доступа к waiting*/
  down(&barbers);     /*Если брадобрей занят, то ждать*/
  get_haircut();      /*Клиента усаживают и обслуживают*/
 }
 else{                /*Если нет свободных стульев, то ...*/
  up(&mutex);         /* придётся уйти*/
 }
}
```

Средства синхронизации потоков

В OpenMP для взаимоисключения и синхронизации потоков можно использовать те же самые средства, которые используются в операционных системах: мьютексы, семафоры, события-переменные. Для них в OpenMP организованы директивы и функции, которые облегчают создание параллельных программ: директивы atomic, critical, механизм замков (locks). Простые замки действуют как мьютексы. Прямого аналога с числовыми семафорами множественные замки не обеспечивают.

1 Семафоры в потоках

Имена функций семафоров для потоков начинаются с префикса sem_. Применяют четыре базовые функции семафоров. Как и большинство системных вызовов, все функции возвращают 0 в случае успешного выполнения.

Семафор создается функцией sem_init.

Прототип функции

```
#include <semaphore.h>
sem_t  sem;
int  sem_init  (sem_t   &sem,  int  pshared,  unsigned  int
value);
```

Эта функция инициализирует семафор, на который указывает параметр sem и присваивает ему начальное целочисленное значение value. Если pshared равно 0, семафор локален по отношению к текущему процессу (наш случай).

Следующая пара функций управляет значением семафора

Прототип функции

```
#include <semaphore.h>
int sem_wait (sem_t  &sem);
int sem_post (sem_t  &sem);
```

Обе они принимают указатель на объект-семафор, инициализированный вызовом sem_init. Это аналоги примитивов P(S) и V(S) семафоров Дейкстры.

Последняя функция семафоров sem_destroy. Она очищает семафор, когда он больше не нужен.

Прототип функции

```
#include <semaphore.h>
int sem_destroy (sem_t *sem);
```

2 Взаимоисключение с помощью мьютексов

Мьютекс – упрощенный двоичный семафор. Его удобно использовать для взаимоисключения нескольких процессов в работе в критическом участке.

Базовые функции для использования мьютексов очень похожи на функции семафоров

Прототипы функций

```
#include <pthread.h>
pthread_mutex_t  mutex
int  phtread_mutex_init  (pthread_mutex_t  *mutex,  NULL);
int  phtread_mutex_lock  (pthread_mutex_t  *mutex);
int  phtread_mutex_unlock  (pthread_mutex_t  *mutex);
int  phtread_mutex_destroy (pthread_mutex_t  *mutex);
```

3. Условная переменная

Условная переменная позволяет потокам ожидать выполнения некоторого условия (события), связанного с разделяемыми данными. Над условными переменными определены две основные операции: *информирование* о наступлении события и *ожидание* события. При выполнении операции «информирование» один из потоков, ожидающих на условной переменной, возобновляет свою работу.

Условная переменная всегда используется совместно с мьютексом. Перед выполнением операции «ожидание» поток должен заблокировать мьютекс. При выполнении операции «ожидание» указанный мьютекс автоматически разблокируется. Перед возобновлением ожидающего потока выполняется автоматическая блокировка мьютекса, позволяющая потоку войти в критическую секцию, после критической секции рекомендуется разблокировать мьютекс. При подачи сигнала другим потокам рекомендуется так же функцию «сигнализации» защитить мьютексом.

Прототипы функций для работы с условными переменными содержатся в файле pthread.h. Ниже приводятся прототипы функций вместе с пояснением их синтаксиса и выполняемых ими действий.

pthread_cond_init(pthread_cond_t* cond, const pthread_condattr_t* attr); – инициализирует условную переменную *cond* с указанными атрибутами *attr* или с атрибутами по умолчанию (при указании 0 в качестве *attr*).

pthread_cond_destroy(pthread_cond_t* cond); – уничтожает условную переменную *cond*.

pthread_cond_signal(pthread_cond_t* cond); – информирование о наступлении события потоков, ожидающих на условной переменной *cond*.

pthread_cond_wait(pthread_cond_t* cond , pthread_mutex_t* mutex); – ожидание события на условной переменной *cond*.

Функция pthread_cond_wait() используется, чтобы атомарно освободить мьютекс и заставить вызывающий поток блокироваться по переменной состояния.

Блокированный поток пробуждается с помощью вызова pthread_cond_signal(),

Проверка состояния обычно проводится в цикле while, который вызовет hread_cond_wait():

```
pthread_mutex_lock();
while(condition_is_false)
  pthread_cond_wait();
pthread_mutex_unlock();
```

Чтобы разблокировать определенный поток, используется функция pthread_cond_signal():

```
int pthread_cond_signal(pthread_cond_t *cv);
```

Она разблокирует поток, заблокированный переменной состояния cv. Функция pthread_cond_signal() возвращает 0 - после успешного завершения. Любое другое значение указывает, что произошла ошибка.

Следует всегда вызывать pthread_cond_signal() под защитой мьютекса, используемого с сигнальной переменной состояния. В ином случае переменная состояния может измениться между тестированием соответствующего состояния и блокировкой в вызове.

Пример использования условной переменной для взаимоисключения двух потоков.

```
pthread_mutex_t mutex;
pthread_cond_t cond ;
int condition = 0;
```

```
…
pthread_mutex_lock( &mutex );
while( condition == 1 )
 pthread_cond_wait(&cond,
                   &mutex );
 printf("Produced %d\n",
        ++count);
 condition = 1;
 pthread_cond_signal(&cond);
 pthread_mutex_unlock(&mutex);
…
```

```
…
pthread_mutex_lock( &mutex );
while( condition == 0 )
 pthread_cond_wait(&cond,
                   &mutex);
 printf("Consumed %d\n",
        count );
 condition = 0;
 pthread_cond_signal(&cond);
 pthread_mutex_unlock(&mutex);
```

```
void main(){
 …
 phtread_mutex_init  (&mutex,  NULL);
 …
}
```

4 Замки в OpenMP

Один из вариантов синхронизации в OpenMP реализуется через механизм замков (*locks*). В качестве замков используются общие целочисленные переменные (размер должен быть достаточным для хранения адреса).

Замок может находиться в одном из трёх состояний: неинициализированный, разблокированный или заблокированный. Разблокированный замок может быть захвачен некоторой нитью. При этом он переходит в заблокированное состояние. Нить, захватившая замок, и только она может его освободить, после чего замок возвращается в разблокированное состояние. Есть два типа замков: простые замки и множественные замки. Множественный замок может многократно захватываться одной нитью перед его освобождением, в то время как простой замок может быть захвачен только однажды. Для множественного замка вводится понятие коэффициентазахваченности *(nesting count)*. Изначально он устанавливается в ноль, при каждом следующем захватывании увеличивается на единицу, а при каждом освобождении уменьшается на единицу. Множественный замок считается разблокированным, если его коэффициент захваченности равен нулю.

Для инициализации простого или множественного замка используются соответственно функции **omp_init_lock()** и **omp_init_nest_lock()**.

Си:

```
omt_lock_t lock;
void omp_init_lock(omp_lock_t *lock);
void omp_init_nest_lock(omp_nest_lock_t *lock);
```

После выполнения функции замок переводится в разблокированное состояние. Для множественного замка коэффициент захваченности устанавливается в ноль.

Функции **omp_destroy_lock()** и **omp_destroy_nest_lock()** используются

для переведения простого или множественного замка в неинициализированное состояние.

Си:

```
void omp_destroy_lock(omp_lock_t *lock);
void omp_destroy_nest_lock(omp_nest_lock_t *lock);
```

Для захватывания замка используются функции **omp_set_lock()** и **omp_set_nest_lock()**.

Си:

```
void omp_set_lock(omp_lock_t *lock);
void omp_set_nest_lock(omp_nest_lock_t *lock);
```

Вызвавшая эту функцию нить дожидается освобождения замка, а затем захватывает его. Замок при этом переводится в заблокированное состояние. Если множественный замок уже захвачен данной нитью, то нить не блокируется, а коэффициент захваченности увеличивается на единицу.

Для освобождения замка используются функции **omp_unset_lock()** и **omp_unset_nest_lock()**.

Си:

```
void omp_unset_lock(omp_lock_t *lock);
void omp_unset_nest_lock(omp_lock_t *lock);
```

Вызов этой функции освобождает простой замок, если он был захвачен вызвавшей нитью. Для множественного замка уменьшает на единицу коэффициент захваченности. Если коэффициент станет равен нулю, замок освобождается. Если после освобождения замка есть нити, заблокированные на операции, захватывающей данный замок, замок будет сразу же захвачен одной из ожидающих нитей.

Обработка программы с использованием потоков

Для применения системных вызовов с использованием потоков в программе необходимо включить в нее заголовочный файл phtread.h и скомпоновать программу с библиотекой потоков, используя ключ –lpthread.

gcc –fopenmp –lpthread –o имя_загр_файла имя_исх_файла.c

Индивидуальные задания

1 Производитель – потребитель. Множественный буфер. Три семафора (п. 2.4).

2 Производитель – потребитель. Одиночный буфер. Два семафора (п. 2.3).

3 Читатели-писатели. Приоритет читателей. Два семафора и счетчик (п. 3.1).

4 Читатели-писатели. Приоритет писателей. Два семафора и критическая секция (п. 3.2).

5 Производитель – потребитель. Множественный буфер. Два семафора и мьютекс п.2.4).

6 Читатели-писатели. Приоритет писателей. Три семафора (п. 3.2).

7 Смоделировать алгоритм Деккера (2.7).

8 Обедающие философы. Синхронизация семафорами (п. 4).

9 Производитель – потребитель. Одиночный буфер. Замок и условная переменная (п.2.3).

10 Читатели-писатели. Приоритет читателей. Два замка и счетчик (п. 3.1).

11 Обедающие философы. Синхронизация замками (п. 4).

12 Производитель – потребитель. Одиночный буфер. Мьютекс и условная переменная.

13 Читатели-писатели. Приоритет читателей. Семафор и критическая секция (п. 3.1).

14 Реализовать программу п. 2.5. Используйте мьютексы. Каждый поток производит свои числовые значения. Задать число итераций.

15 Смоделировать алгоритм Петерсона (п. 2.6).

16 Читатели-писатели. Приоритет писателей. Два замка и критическая секция (п.3.2).

17 Производитель – потребитель. Одиночный буфер. Синхронизация номером процесса.

18 Обедающие философы. Синхронизация мьютексами (п. 4).

19 Читатели-писатели. Приоритет читателей. Замок и критическая секция (п. 3.1).

20 Производитель – потребитель. Одиночный буфер. Два замка (2.3).

21 Реализовать программу п. 3.3. Использовать семафоры.

22 Читатели-писатели. Приоритет читателей. Множественный замок для читателей.

23 Производитель – потребитель. Множественный буфер. Два семафора и замок (п. 2.4).

24 Производитель – потребитель. Одиночный буфер. Два мьютекса (п. 2.3).

25 Читатели-писатели. Приоритет писателей. Три замка (п. 3.2).

26 Обедающие философы. Синхронизация мьютексами (п. 4).

27 Производитель – потребитель. Одиночный буфер. Две условных переменных.

28 Читатели-писатели. Приоритет читателей. Мьютекс и критическая секция (п. 3.1).

29 Читатели-писатели. Программа п. 3.4.

30 Производитель – потребитель. Множественный буфер (п.2.4). Реализовать синхронизацию производителя и потребителя моделированием числового семафора замком и счетчиком.