Лабораторная работа № 5

Решение практических задач с применением технологии MPI.

Цель

Получить практический навык использования технологии MPI при решении прикладных задач.

Примеры программ

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

using namespace std;

int c_size, c_rank;

double RunMatrixTest(unsigned mSize)
{
srand(clock());
// Выделяем память под матрицу, которую будем умножать на саму себя
double *matr= new double[mSize*mSize];
// Выделяем память под матрицу-результат только в том случае, если
// находимся в головном процессе
double *result= (!c_rank)? new double[mSize*mSize]: NULL; // Выделяем
память под одну строку матрицы результата только если находимся в
"рабочем" процессе
double *row= (c_rank)? new double[mSize]: NULL;
// Если находимся в головном процессе, инициализируем исходную матрицу
if(!c_rank)
  for(unsigned i=mSize*mSize; i--; matr[i]= rand()/(double)RAND_MAX);
// Делимся матрицей со всем процессами. Если находимся в головном
// процессе, происходит рассылка, иначе - прием даннных
MPI_Bcast(matr, mSize*mSize, MPI_DOUBLE, 0, MPI_COMM_WORLD);
double tm = MPI_Wtime();
// Вычисляем произведение матрицы на саму себя
// Строки результирующей матрицы делятся между процессами: 0ю строку
```

```
// считает первый,
// 1ю - второй, 2ю- третий, 3ю - снова первый и т.д.
// Матрицу обрабатываем блоками по c size строк
for(unsigned iBlockRow=0; iBlockRow<mSize; iBlockRow+=c_size)
  {
  // Номер строки в матрице, которую должен обработать текущий
  // процесс для текущего блока
  unsigned iRow= iBlockRow+c_rank;
  if(iRow<mSize)
    {
   // Если находимся в головном процессе, то строка приемник - это часть
итоговой матрицы
   // Иначе, у нас под строку уже выделена память
   if(!c_rank)
     row= result+mSize*iRow; // Считаем результирующую строку
   for(unsigned iColumn=0; iColumn<mSize; iColumn++)
     {
     row[iColumn]= 0;
     for(unsigned k=0; k<mSize; k++)
       row[iColumn]+= matr[iRow*mSize+k]*matr[k*mSize+iColumn];
     }
   MPI_Status status;
   if(c_rank)
    // Если находимся в "рабочем" процессе, отсылаем результат головному
     MPI_Send(row, mSize, MPI_DOUBLE, 0, 0, MPI_COMM_WORLD);
   else
    // Если находимся в головном процессе, собираем результаты расчетов
"рабочих" процессов
     for(int iProc=1; (iProc < c_size) && (iBlockRow+iProc)<mSize; iProc++)
       MPI_Recv(result+(iBlockRow+iProc)*mSize, mSize, MPI_DOUBLE, iProc, 0,
MPI_COMM_WORLD, &status);
   }
   // Ждем, пока все процессы дойдут до этой строчки
  MPI_Barrier(MPI_COMM_WORLD);
  }
tm= MPI_Wtime()-tm;
if(!c_rank)
  {
   //Если находимся в головном процессе, у нас имеется полностью готовая
матрица
  delete[]matr;
  delete[]result;
  }
if(c_rank)
  delete[] row;
```

```
return tm;
}

#define _SIZE 10
int main(int argc, char *argv[])
{
// Инициализируем подсистему MPI
MPI_Init(&argc, &argv);
// Получаем номер процесса и размер коммуникатора
MPI_Comm_size(MPI_COMM_WORLD, &c_size);
MPI_Comm_rank(MPI_COMM_WORLD, &c_rank);
try {
// Запускаем процесс перемножения матриц
  double tm= RunMatrixTest(_SIZE);
  if(!c_rank)
    // Если находимся в головной процессе, выводим результат
    printf("Multiplying matrixes (%dx%d): %lfsec", _SIZE, _SIZE, tm);
}
catch(...) {

}
// Освобождаем подсистему MPI
  MPI_Finalize();

return 0;
}
      #undef SIZE
```

Задание к выполнению лабораторной работы

Разработать распределенную программу (с использованием технологии MPI), решающую задачу из приведенного списка практических задач. Номер решаемой задачи выбрать соответственно своему варианту.

$$\int_a^b f(x)dx \approx \sum_{i=1}^{n} \omega_i f(x_i),$$

где n — число точек, в которых вычисляется значение подынтегральной функции. Точки называются узлами метода, числа ω_i — весами узлов.

Одним из методов численного интегрирования является метод Симпсона, в котором подынтегральная функция на отрезке интегрирования заменяется параболой. Обычно в качестве узлов метода используют концы отрезка и его среднюю точку. В этом случае формула имеет очень простой вид:

$$\int_a^b f(x)dx \approx \frac{b-a}{6}\left(f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)\right).$$

Если разбить интервал интегрирования на 2N равных частей, то получим:

$$\int_a^b f(x)dx \approx \frac{b-a}{6N}\left(f_0 + 4\left(f_1 + f_3 + \ldots + f_{2N-1}\right) + 2\left(f_2 + f_4 + \ldots + f_{2N-2}\right) + f_{2N}\right)$$

Задача №1

Численно найти значение определенного интеграла данной функции с точностью до 10 знаков после запятой.

Теоретические сведения

Пусть необходимо найти значение интеграла $I = \int_a^b f(x)dx$.

Основная идея большинства методов численного интегрирования состоит в замене подынтегральной функции на более простую, интеграл от которой легко вычисляется аналитически. При этом для оценки значения интеграла получаются формулы вида:

$$\int_a^b f(x)dx \approx \sum_{i=1}^{n} \omega_i f(x_i),$$

где n — число точек, в которых вычисляется значение подынтегральной функции. Точки называются узлами метода, числа ω_i — весами узлов.

Одним из методов численного интегрирования является метод Симпсона, в котором подынтегральная функция на отрезке интегрирования заменяется параболой. Обычно в качестве узлов метода используют концы отрезка и его среднюю точку. В этом случае формула имеет очень простой вид:

$$\int_a^b f(x)dx \approx \frac{b-a}{6}\left(f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)\right).$$

Если разбить интервал интегрирования на 2N равных частей, то получим:

$$\int_a^b f(x)dx \approx \frac{b-a}{6N}\left(f_0 + 4\left(f_1 + f_3 + \ldots + f_{2N-1}\right) + 2\left(f_2 + f_4 + \ldots + f_{2N-2}\right) + f_{2N}\right)$$

где $f_i = f\left(a + \frac{(b-a)i}{2N}\right)$ — значение функции в i–ой точке.

Приближение функции одним полиномом на всем отрезке интегрирования, как правило, приводит к большой ошибке в оценке значения интеграла. Для уменьшения погрешности отрезок интегрирования разбивают на части и применяют численный метод для оценки интеграла на каждом из них.

Величину погрешности приближения можно оценить, сравнив приближенные значения интеграла, полученные при разбиении отрезка интегрирования на различное число частей (как правило, шаг разбиения отрезков изменяется при этом вдвое).

Задача №2

Решить задачу Коши для системы обыкновенных дифференциальных уравнений первого порядка методом Рунге–Кутта.

Теоретические сведения

Пусть дана задача Коши для системы n дифференциальных уравнений первого порядка:

$$\begin{cases} y_1' = F_1(x, y_1, \ldots, y_n) \\ \ldots \\ y_n' = F_n(x, y_1, \ldots, y_n) \\ y_1(x^{(0)}) = y_1^{(0)} \\ \ldots \\ y_n(x^{(0)}) = y_n^{(0)} \end{cases},$$

где x — независимая переменная, $y_i' = F_i(x, y_1, \ldots, y_n)$ — производная i–ой неизвестной функции, $x^{(0)}$ — значение независимой переменной в начальный момент времени, $y_i(x^{(0)}) = y_i^{(0)}$ — значение i–ой неизвестной функции в начальный момент времени.

Зададимся некоторым ненулевым шагом интегрирования h. Тогда приближенное значение i–ой неизвестной функции в точке $x^{(k+1)} = x^{(k)} + h$ можно вычислить по формуле:

$$y_i(x^{(k-1)}) = y_i^{(k-1)} \approx y_i^{(k)} + \frac{h}{6}(k_1^{(i)} + 2k_2^{(i)} + 2k_3^{(i)} + k_4^{(i)}).$$

где

$$k_1^{(i)} = F_i(x^{(k)}, y_1^{(k)}, \ldots, y_n^{(k)})$$

$$k_2^{(i)} = F_i(x^{(k)} + h/2, y_1^{(k)} + k_1^{(1)}/2, \ldots, y_n^{(k)} + k_1^{(n)}/2)$$

$$k_3^{(i)} = F_i(x^{(k)} + h/2, y_1^{(k)} + k_2^{(i)}/2, \ldots, y_n^{(k)} + k_2^{(n)}/2)$$

$$k_4^{(i)} = F_i(x^{(k)} + h, y_1^{(k)} + k_3^{(i)}, \ldots, y_n^{(k)} + k_3^{(n)})$$

Задача №3

Выполнить локальную фильтрацию растрового изображения.

Теоретические сведения

Для представления графической информации на двумерной плоскости монитора применяется растровая графика. Растровая графика оперирует с произвольными изображениями в виде растров. Растр[10] — это описание изображения на плоскости путем разбиения (дискретизации) его на одинаковые элементы по регулярной сетке и присвоением каждому элементу цветовой информации.

Наиболее распространенная цветовая модель — RGB[11]. В этой модели каждый пиксель представлен тремя числами — тремя цветовыми составляющими (красной, зеленой и голубой).

Локальная фильтрация — это метод обработки растровых изображений, позволяющий улучшить его качества: уменьшить искажения, увеличить резкость, выделить контуры и т.д. Суть метода заключается в перемещении по изображению локального окна (матрицы). Окно проходит по изображению так, что каждый пиксель

---

[10] от англ. Raster

[11] От англ. red, green, blue — красный, зеленый, голубой

один раз бывает его центром. На окне определена весовая функция $H(p,q)$, где p,q — координаты центрального пикселя окна. Эта весовая функция определяет новый цвет пикселя, который есть центр окна. Эффект фильтрации зависит только от функции H. Размер окна обычно бывает 3x3, 5x5 или 7x7 и т.д.

Например, окно для сглаживания изображения, имеет следующий вид:

$$A[i,j]=\frac{1}{8}\begin{vmatrix}0 & 1 & 0\\ 1 & 4 & 1\\ 0 & 1 & 0\end{vmatrix}$$

$D=\frac{1}{8}$ — коэффициент

$C'[x,y]=D\sum_{i,j}\left(A[i,j]*C[i,j]\right)$ — правило получения нового значения цвета пикселя

$C_{n\ m}$ — матричное представление растра.

Задача №4.

Найти произведение N матриц согласованных размеров.

Теоретические сведения

Пусть даны две матрицы $A_{n\times m}$ и $B_{m\times k}$:

$$A_{n\times m}=\begin{pmatrix}a_{11} & a_{12} & \dots & a_{1m}\\ a_{21} & a_{22} & \dots & a_{2m}\\ & & \dots & \\ a_{n1} & a_{n2} & \dots & a_{nm}\end{pmatrix},\ A_{m\times k}=\begin{pmatrix}a_{11} & a_{12} & \dots & a_{1k}\\ a_{21} & a_{22} & \dots & a_{2k}\\ & & \dots & \\ a_{m1} & a_{m2} & \dots & a_{mk}\end{pmatrix}.$$

Произведение матриц $A_{n\times m}$ и $B_{m\times k}$ называется такая матрица $C_{n\times k} = A_{n\times m} \times B_{m\times k}$, каждый элемент которой равен: $c_{ij} = \sum_{k\ 1}^{m} a_{ik} b_{kj}$.

Операция перемножения матриц определена только для матриц согласованного размера (число столбцов левой матрицы должно совпадать с числом строк в правой) и не обладает свойством коммутативности: $A_{n\ m} \times B_{m\times k} \neq B_{m\ k} \times A_{n\times m}$.

Задача №5

Решить систему линейных алгебраических уравнений методом Гаусса–Жордана.

Теоретические сведения

Пусть дана система из n линейных алгебраических уравнений от n неизвестных:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n = b_2 \\ \ldots \\ a_{n1}x_1 + a_{n2}x_2 + \ldots + a_{nn}x_n = b_n \end{cases}.$$

Матрицу коэффициентов данной системы можно записать в виде:

$$\left(\begin{array}{cccc|c} a_{11} & a_{12} & \ldots & a_{1n} & b_1 \\ a_{21} & a_{22} & \ldots & a_{2n} & b_2 \\ & \ldots & & & \ldots \\ a_{n1} & a_{n2} & \ldots & a_{nn} & b_n \end{array}\right),$$

где последний столбец – свободные коэффициенты.

Суть метода заключается в приведении матрицы коэффициентов с учетом свободных членов сначала к верхнему треугольному, а затем к единичному виду:

$$\left(\begin{array}{cccc|c} a_{11} & a_{12} & \dots & a_{1n} & b_1 \\ a_{21} & a_{22} & \dots & a_{2n} & b_2 \\ & \dots & & & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} & b_n \end{array}\right) \rightarrow \left(\begin{array}{cccc|c} 1 & a_{12}' & \dots & a_{1n}' & b_1' \\ 0 & 1 & \dots & a_{2n}' & b_2' \\ & & \dots & & \dots \\ 0 & 0 & \dots & 1 & b_n' \end{array}\right) \rightarrow \left(\begin{array}{cccc|c} 1 & 0 & \dots & 0 & b_1'' \\ 0 & 1 & \dots & 0 & b_2'' \\ & & \dots & & \dots \\ 0 & 0 & \dots & 1 & b_n'' \end{array}\right).$$

В последнем столбце после всех преобразований будут искомые неизвестные.

Алгоритм метода:

1. Выбирается первая колонка слева, в которой есть хоть одно отличное от нуля значение.
2. Если самое верхнее число в этой колонке есть нуль, то меняется вся первая строка матрицы с другой строкой матрицы, где в этой колонке нет нуля.
3. Все элементы первой строки делятся на верхний элемент выбранной колонки.
4. Из оставшихся строк вычитается первая строка, умноженная на первый элемент соответствующей строки, с целью получить первым элементом каждой строки (кроме первой) нуль.
5. Далее проводим такую же процедуру с матрицей, получающейся из исходной матрицы после вычёркивания первой строки и первого столбца.
6. После повторения этой процедуры (n-1) раз получаем верхнюю треугольную матрицу
7. Вычитаем из предпоследней строки последнюю строку, умноженную на соответствующий коэффициент, с тем, чтобы в предпоследней строке осталась только 1 на главной диагонали.
8. Повторяем предыдущий шаг для последующих строк. В итоге получаем единичную матрицу и решение на месте свободного вектора (с ним необходимо проводить все те же преобразования).
9. Чтобы получить обратную матрицу, нужно применить все операции в том же порядке к единичной матрице.

Задача №6

Найти матрицу, обратную данной квадратной, с помощью союзной матрицы.

Теоретические сведения

Пусть дана квадратная матрица:

$$A_{n\times n}=\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ & & \dots & \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix}.$$

Обратной называется такая матрица $A_{n\times n}{}^{-1}$. для которой выполнено условие: $A_{n\times n}\times A_{n\times n}{}^{-1}=E$, где E — единичная матрица.

Для поиска обратной матрицы можно воспользоваться равенством:

$$A_{n\times n}{}^{-1}=\frac{1}{\det A}\begin{pmatrix} A_{11} & A_{12} & \dots & A_{1n} \\ A_{21} & A_{22} & \dots & A_{2n} \\ & & \dots & \\ A_{n1} & A_{n2} & \dots & A_{nn} \end{pmatrix},$$

где $\det A$ — определитель матрицы A, $A_{ij}=(-1)^{i-j}M_j^i$ — алгебраическое дополнение элемента a_{ij} матрицы A, M_j^i — минор (определитель матрицы, получающийся из A вычеркиванием i–й строки и j–го столбца).

Задача №7

Реализовать алгоритм блочного шифрования в режиме электронной кодовой книги (простой замены).

Теоретические сведения

Алгоритм блочного шифрование перерабатывает открытый текст в шифротекст блоками фиксированного размера. Размер блока зависит от самого алгоритма или его настроек.

Открытый текст некоторой длинны с помощью блочного алгоритма шифрования можно переработать в шифротекст различными способами (режимами). Самым простым из них (и самым

небезопасным) является режим электронной кодовой книги. При работе в этом режиме, каждый блок шифротекста получается путем шифрования соответствующего блока открытого текста по данному ключу.

Задача №8

Найти матрицу, обратную данной квадратной методом Гаусса.

Пусть дана квадратная матрица:

$$A_{n\times n}=\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ & & \dots & \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix}.$$

Обратной называется такая матрица $A_{n\times n}^{-1}$, для которой выполнено условие: $A_{n\times n}\times A_{n\times n}^{-1}=E$, где E — единичная матрица.

Возьмем две матрицы: саму матрицу $A_{n\times n}$ и единичную $E_{n\times n}$. Приведём матрицу $A_{n\times n}$ путем элементарных преобразований к единичной матрице методом Гаусса (см. задачу №5). После применения каждой операции к первой матрице применим ту же операцию ко второй матрице. Когда приведение первой матрицы к единичному виду будет завершено, вторая матрица окажется равной искомой обратной матрице $A_{n\times n}^{-1}$.

Задача №9

Найти кратчайшие пути между всеми вершинами графа с помощью алгоритма Флойда.

Алгоритм Флойда–Уоршелла (рус.) — http://ru.wikipedia.org/wiki/Алгоритм_Варшалла

Задача №10

Найти все возможные размещения N ферзей на шахматной доске размеров NxN.

Теоретические сведения

Рассмотрим шахматную доску, которая имеет размеры не 8x8, а NxN, где N>0. Как известно, шахматный ферзь атакует все клетки и фигуры на одной с ним вертикали, горизонтали и диагонали. Любое расположение нескольких ферзей на шахматной доске будем называть их размещением. Размещение называется допустимым, если ферзи не атакуют друг друга. Размещение N ферзей на шахматной доске NxN называется полным. Допустимые полные размещения существуют не при любом значении N. Например, при N=2 или 3 их нет. При N=4 их лишь 2, причем они зеркально отражают друг друга.