# Задание

Написать программу, на языке С++ в среде VisualStudio “**. Модель, реализующая матрицу Якоби с граничными условиями (состояние пространства за пределом матрицы принимается равным 0)”:**

а) начальное воздействие прикладывается к выбранной точке (координаты точки и величина воздействия являются входными данными) и удерживаются в течение всего времени моделирования.

б) начальное воздействие прикладывается к выбранной точке (координаты точки и величина воздействия являются входными данными) в первом такте моделирования и затем снимается.

Однородные вычислительные структуры или среды (ОВС), как правило, относятся к классу ОКМД (согласно классификации Флинна) и представляют собой регулярную структуру из однотипных процессорных элементов (ПЭ).

Каждый ПЭ, в зависимости от типа ОВС, может как обладать алгоритмически полным набором операций, так и реализовывать один вид операций, жестко заданный в структуре микросхемы на этапе проектирования, а также выполнять операции обмена или взаимодействия с другими ПЭ.

# OpenMP

Среди специалистов, занимающихся параллельными вычислениями, популярна шутка «Параллельные вычисления — технология будущего... и так будет всегда». Эта шутка не теряет актуальность уже несколько десятилетий. Аналогичные настроения были распространены в сообществе разработчиков архитектур компьютеров, обеспокоенном тем, что скоро будет достигнут предел тактовой частоты процессоров, однако частоты процессоров продолжают повышаться, хотя гораздо медленнее, чем раньше. Сплав оптимизма специалистов по параллельным вычислениям и пессимизма архитекторов систем способствовал появлению революционных многоядерных процессоров.

Главные производители процессоров сместили акцент с повышения тактовых частот на реализацию параллелизма в самих процессорах за счет использования многоядерной архитектуры. Идея проста: интегрировать в один процессор более одного ядра. Система, включающая процессор с двумя ядрами, по сути, не отличается от двухпроцессорного компьютера, а система с четырехядерным процессором — от четырехпроцессорного. Этот подход позволяет избежать многих технологических проблем, связанных с повышением тактовых частот, и создавать при этом более производительные процессоры.

Все это прекрасно, но если ваше приложение не будет использовать несколько ядер, его быстродействие никак не изменится. Именно здесь и вступает в игру технология OpenMP, которая помогает программистам на C++ быстрее создавать многопоточные приложения.

Стандарт OpenMP был разработан в 1997 г. как API, ориентированный на написание портируемых многопоточных приложений. Сначала он был основан на языке Fortran, но позднее включил в себя и C/C++. Последняя версия OpenMP — 2.0;  ее полностью поддерживает Visual C++ 2005. Стандарт OpenMP поддерживается и платформой Xbox 360.

в Visual C++ 2005 параметр компилятора /openmp. (Вы можете активизировать директивы OpenMP на страницах свойств проекта, выбрав Configuration Properties, C/C++, Language и изменив значение свойства OpenMP Support.) Встретив параметр /openmp, компилятор определяет символ \_OPENMP, с помощью которого можно выяснить, включены ли средства OpenMP. Для этого достаточно написать #ifndef \_OPENMP.

OpenMP связывается с приложениями через библиотеку импорта vcomp.lib. Соответствующая библиотека периода выполнения называется vcomp.dll. Отладочные версии библиотек импорта и периода выполнения (vcompd.lib и vcompd.dll соответственно) поддерживают дополнительные сообщения об ошибках, генерируемых при некоторых недопустимых операциях. Имейте в виду, что Visual C++ не поддерживает статическое связывание с библиотекой OpenMP периода выполнения, хотя в версии для Xbox 360 это поддерживается.

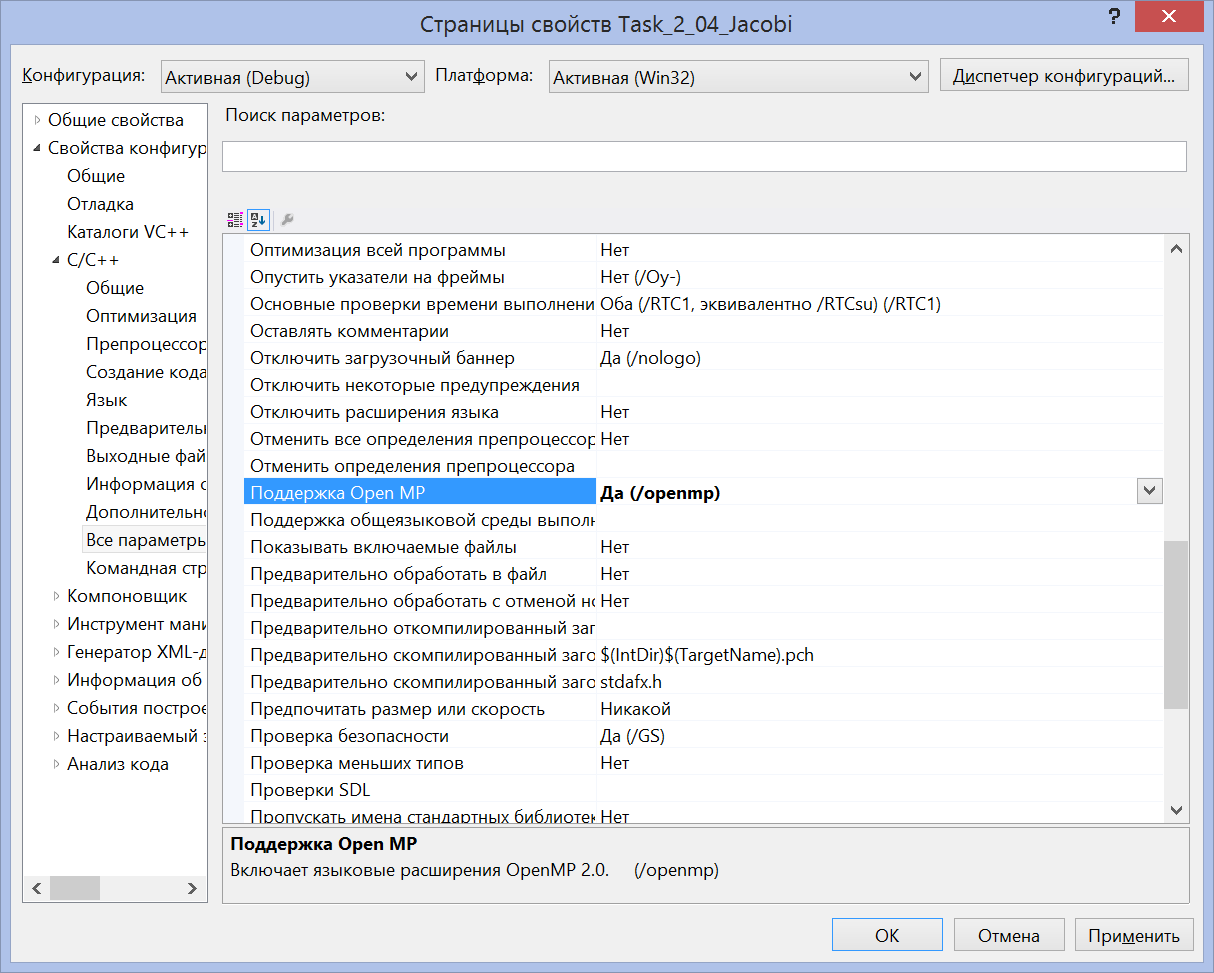
Работа OpenMP-приложения начинается с единственного потока — основного. В приложении могут содержаться параллельные регионы, входя в которые, основной поток создает группы потоков (включающие основной поток). В конце параллельного региона группы потоков останавливаются, а выполнение основного потока продолжается. В параллельный регион могут быть вложены другие параллельные регионы, в которых каждый поток первоначального региона становится основным для своей группы потоков. Вложенные регионы могут в свою очередь включать регионы более глубокого уровня вложенности.

OpenMP прост в использовании и включает лишь два базовых типа конструкций: директивы pragma и функции исполняющей среды OpenMP. Директивы pragma, как правило, указывают компилятору реализовать параллельное выполнение блоков кода. Все эти директивы начинаются с #pragma omp. Как и любые другие директивы pragma, они игнорируются компилятором, не поддерживающим конкретную технологию — в данном случае OpenMP.

Функции OpenMP служат в основном для изменения и получения параметров среды. Кроме того, OpenMP включает API-функции для поддержки некоторых типов синхронизации. Чтобы задействовать эти функции библиотеки OpenMP периода выполнения (исполняющей среды), в программу нужно включить заголовочный файл omp.h. Если вы используете в приложении только OpenMP-директивы pragma, включать этот файл не требуется.

Для реализации параллельного выполнения блоков приложения нужно просто добавить в код директивы pragma и, если нужно, воспользоваться функциями библиотеки OpenMP периода выполнения.

OpenMP поддерживает директивы parallel, for, parallel for, section, sections, single, master, critical, flush, ordered и atomic, которые определяют или механизмы разделения работы или конструкции синхронизации.



# Модель, реализующая матрицу Якоби с граничными условиями

Итерационные вычисления Якоби предназначены для нахождения решения дифференциальных уравнений Лапласа и представляют собой последовательность итераций вычисления значений ячеек новой матрицы равной усреднённому значения соседних ячеек предшествующей матрицы.

Итерационный процесс повторяется пока отличие новой матрицы от предшествующей станет незначительным.

В программе реализована работа с многомерными матрицами, хранящаяся в памяти в виде одномерной последовательности чисел.

Количество пространственных измерений и размерность массивов по каждому из измерений запрашивается при запуске программы.

Программа работает в 2-х режимах:

* с выводом результатов работы на экран (для демонстрации работоспособности);
* без вывода результатов работы на экран, но с определением времени, затрачиваемого на вычисления;

Листинг кода

#include "Header.hpp"

using namespace std;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Модель, реализующая матрицу Якоби с граничными условиями

n - количество измерений

\*m - количество точек по каждому из измерений

\*x - вектор координат точки воздействия

value - величина воздействия

mode - режим (0 - непрерывное воздействе, 1 - однократное воздействие)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int jacobi\_field(int n,int \*m,int \*x,double value,int mode, int demo){

double epsilon=1e-7; // минимальное изменение для продолжения итераций

int nmax=1000; // максимальное число итераций

int total = 1; for(int i = 0; i < n; i++) total\*=m[i];

double \*prevMatrix = new double[total]; // Многомерные массивы хранятся в виде одномерной последовательности

double \*nextMatrix = new double[total]; // Многомерные массивы хранятся в виде одномерной последовательности

if(prevMatrix==NULL) return -1; // Ошибка аллокирования памяти

if(nextMatrix==NULL) return -1; // Ошибка аллокирования памяти

for(int i = 0; i < total; i++) prevMatrix[i]=0;

int index = indexOf(x,m,n); // индекс точки воздействия

prevMatrix[index] = value;

for(int t=0; t<nmax; t++) {

if(demo!=0) cout << "\tШаг итерации:" << t << endl;

if(demo!=0) {

cout << "\tИсходное состояние матрицы:" << endl;

for(int i = 0; i < total; i++) {

cout << prevMatrix[i] << '\t';

if(i%m[0]==m[0]-1) cout << endl;

}

}

// Шаг итераций

// используется параллельная обработка элементов массива

// Однородные вычислительные структуры или среды (ОВС), как правило,

// относятся к классу ОКМД (согласно классификации Флинна)

// и представляют собой регулярную структуру из однотипных процессорных элементов (ПЭ).

// Каждый ПЭ, в зависимости от типа ОВС, может как обладать алгоритмически полным набором

// операций, так и реализовывать один вид операций, жестко заданный в структуре микросхемы

// на этапе проектирования, а также выполнять операции обмена или взаимодействия с другими ПЭ.

// OpenMP поддерживает директивы parallel, for, parallel for, section, sections, single, master, critical, flush, ordered и atomic,

// которые определяют или механизмы разделения работы или конструкции синхронизации.

// Общие переменные доступны всем потокам из группы, поэтому изменения таких переменных в одном потоке видимы другим потокам

// в параллельном регионе. Что касается частных переменных, то каждый поток из группы располагает их отдельными экземплярами,

// поэтому изменения таких переменных в одном потоке никак не сказываются на их экземплярах,

// принадлежащих другим потокам.

// частными являются индексы параллельных циклов for.

#pragma omp parallel for

for(int i = 0; i < total; i++) {

int \*x0 = new int[n];

int \*x1 = new int[n];

nextMatrix[i]=0;

vectorOf(i, x0, m, n);

// Вычисляем среднее соседних точек к точке x0 по каждому из измерений

for(int j=0; j<n; j++) {

if(x0[j]+1<m[j]) {

for(int j1=0; j1<n; j1++) x1[j1]=(j1!=j)?x0[j1]:(x0[j]+1);

nextMatrix[i] += prevMatrix[indexOf(x1, m, n)];

}

else

nextMatrix[i] += 0; // состояние пространства за пределом матрицы принимается равным 0

if(x0[j]-1>=0) {

for(int j1=0; j1<n; j1++) x1[j1]=(j1!=j)?x0[j1]:(x0[j]-1);

nextMatrix[i] += prevMatrix[indexOf(x1, m, n)];

}

else

nextMatrix[i] += 0; // состояние пространства за пределом матрицы принимается равным 0

}

nextMatrix[i] /= 2\*n;

delete x0;

delete x1;

}

if(demo!=0) {

cout << "\tНовое состояние матрицы:" << endl;

for(int i = 0; i < total; i++) {

cout << nextMatrix[i] << '\t';

if(i%m[0]==m[0]-1) cout << endl;

}

}

// Расчёт разницы (кроме точки воздействия при непрерывном воздействии)

// Нахождение максимума требует дополнительных обработок для параллельных вычислений

// Поэтому используется непараллельная версия

double delta = 0;

for(int i = 0; i < total; i++)

if(i==index&&mode==0) continue;

else if(delta<abs(nextMatrix[i]-prevMatrix[i]))

delta=abs(nextMatrix[i]-prevMatrix[i]);

if(demo!=0) cout << "\tМаксимальное изменение матрицы:" << delta << endl;

if(demo!=0) {

cout << "\tНажмите ВВОД для продолжения" << endl;

getchar();

getchar();

}

// Проверка условия прекращения итераций

if(delta<epsilon) break;

double \*temp = prevMatrix; prevMatrix = nextMatrix; nextMatrix = temp;

if(mode==0) prevMatrix[index] = value;

}

delete prevMatrix;

delete nextMatrix;

return 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Преобразование вектора координат в индекс одномерного массива

int \*x; // координаты точки

int \*m; // количество точек по каждому из измерений

int n; // количество измерений

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int indexOf(int \*x, int \*m, int n) {

int index = 0;

for(int i=n;--i>0;) {

index += x[i];

index \*= m[i-1];

}

index += x[0];

return index;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Преобразование индекса в одномерном массиве в вектор координат

int index - индекс в одномерном массиве

int \*x; // координаты точки

int \*m; // количество точек по каждому из измерений

int n; // количество измерений

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int \*vectorOf(int index, int \*x, int \*m, int n) {

for(int i=n;--i>0;) {

x[i] = index/m[i-1];

index -= x[i]\*m[i-1];

}

x[0] = index;

return x;

# }

#include "Header.hpp"

using namespace std;

// 4. Модель, реализующая матрицу Якоби с граничными условиями

// (состояние пространства за пределом матрицы принимается равным 0):

// а) начальное воздействие прикладывается к выбранной точке (координаты точки и величина воздействия являются входными данными) и удерживаются в течение всего времени моделирования.

// б) начальное воздействие прикладывается к выбранной точке (координаты точки и величина воздействия являются входными данными) в первом такте моделирования и затем снимается.

int main(){

int n; // количество измерений

int \*m; // количество точек по каждому из измерений

int \*x; // вектор координат точки воздействия

double value; // величина воздействия

int mode; // режим

int demo; // режим программы

setlocale(LC\_ALL, "Russian");

cout << "\tМодель, реализующая матрицу Якоби с граничными условиями" << endl;

cout << "\tВведите количество измерений:" << endl; cin >> n;

m = new int[n];

x = new int[n];

cout << "\tВведите количество точек по каждому из измерений m["<<n<<"]:" << endl;

cout << "\tВНИМАНИЕ!!! количество точек == размер сетки + 1" << endl;

for(int i = 0; i < n; i++)

cin >> m[i];

cout << "\tВведите координаты точки воздействия x["<<n<<"]:" << endl;

for(int i = 0; i < n; i++)

cin >> x[i];

cout << "\tВведите величину воздействия:" << endl; cin >> value;

cout << "\tУкажите режим (0 - непрерывное воздействе, 1 - однократное воздействие):" << endl; cin >> mode;

cout << "\tДемонстрационный режим програмы (0 - замер времени, 1 - демонстрация работоспособности):" << endl; cin >> demo;

clock\_t t = clock();

if(jacobi\_field(n,m,x,value,mode, demo)!=0) cout << "\tОшибка аллокирования памяти" << endl;

t = clock()-t;

double seconds = ((double)t)/CLOCKS\_PER\_SEC;

cout << "\tВремя выполнения: " << seconds << "sec" << endl;

delete m;

delete x;

getchar();

getchar();

return 0;

}

# Контрольные примеры работы программы

Пример 1.

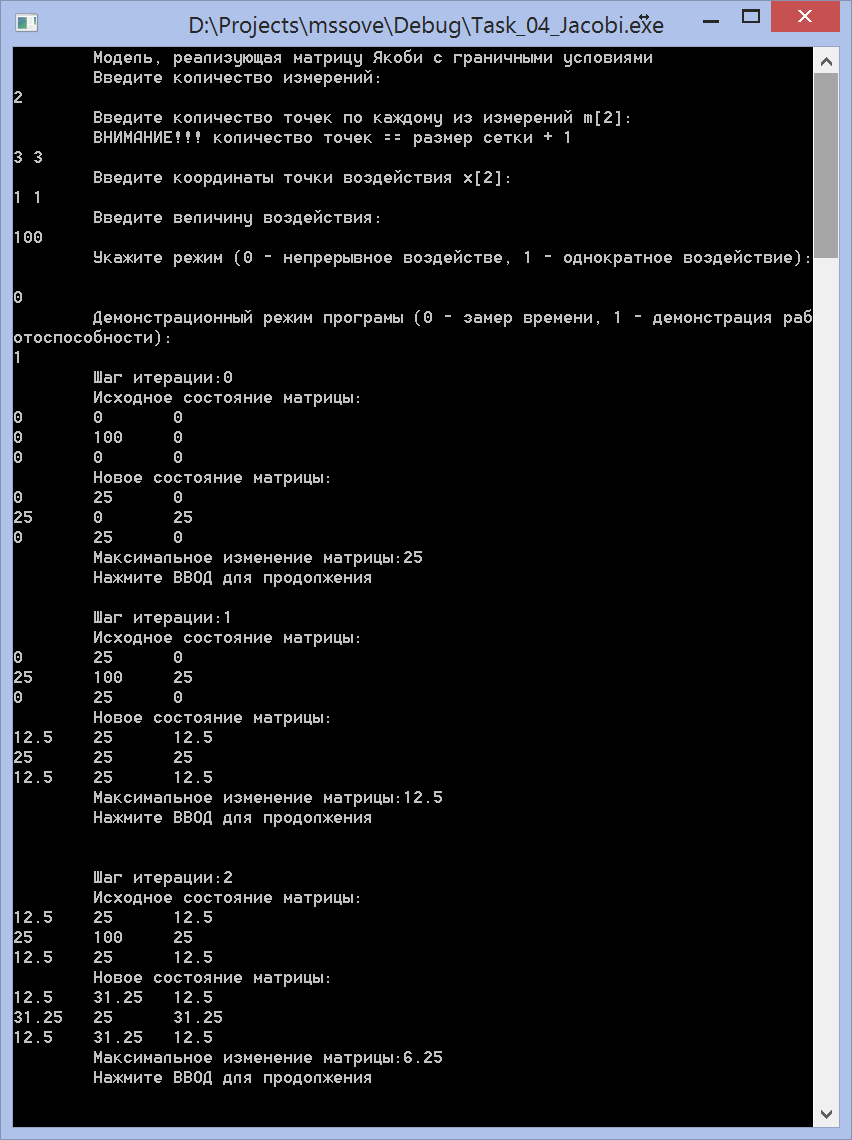


Рис.1..

# Заключение

В ходе данной лабораторной был изучены алгоритм итерационных вычислений Якоби для нахождения решения дифференциальных уравнений Лапласа. Была сделана программная реализация данного алгоритма. Для данной программной реализации были проведены ряд тестов, показывающие правильность работы алгоритма.