МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение

высшего образования   
**«Национальный исследовательский   
Нижегородский государственный университет им. Н.И. Лобачевского»**

**(ННГУ)**

**Институт информационных технологий, математики и механики**

**Кафедра математического обеспечения и суперкомпьютерных технологий**

Направление подготовки: «Прикладная математика и информатика»

Профиль подготовки: «Вычислительная математика и суперкомпьютерные технологии»

Отчет по лабораторной работе №3

**Выполнил:** студент группы 381903-3м

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Панов А.А.

Подпись

**Проверил:**

к.ф.-м. н., доц., доцент каф. МОСТ

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Баркалов К.А.

Подпись

Нижний Новгород  
2020 Содержание

[Введение 3](#_Toc42017364)

[1. Постановка задачи 4](#_Toc42017365)

[2. Метод частичной дискретизации 5](#_Toc42017366)

[3. Метод Рунге-Кутта 4 порядка 6](#_Toc42017367)

[4. Параллельная реализация 7](#_Toc42017368)

[5. Вычислительные эксперименты 9](#_Toc42017369)

[6. Заключение 12](#_Toc42017370)

[Список литературы 13](#_Toc42017371)

# 

# Введение

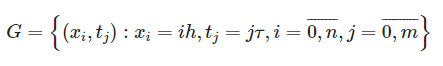
Многие физические задачи описываются дифференциальными уравнениями в частных производных. Очень часто аналитически решить такие уравнения невозможно. Такие уравнения обычно решаются численно. В данной лабораторной на примере решения динамической одномерной задачи теплопроводности (задача о «стержне») будут рассмотрены параллельные реализации некоторых методов.

# Постановка задачи

С помощью метода частичной дискретизации, используя метод Рунге-Кутта 4-ого порядка решить динамическую одномерную задачу теплопроводности вида:

Со следующими граничными/начальными условиями:

Требуется найти численное решение на равномерной сетке:



,

Программа на языке C++ должна реализовывать функцию со следующим заголовком:

class heat\_task {  
public:  
 double T; // момент времени, в который необходимо аппроксимировать

u(x, t)  
 double L; // длина стержня  
 int n; // размер сетки по x  
 int m; // размер сетки по t  
 double initial\_condition(double x); // функция, задающая начальное

условие  
 double left\_condition (double t); // функция, задающая граничное

условие при x = 0  
 double right\_condition (double t); // функция, задающая граничное

условие при x = L  
 double f(double x, double t); // функция, задающая внешнее

воздействие  
};  
void heat\_equation\_runge\_kutta(heat\_task task, double \* v);

Должны быть выполнены условия на порядок сходимости:

Размерность сетки nm ≤ 226.

# Метод частичной дискретизации

Рассмотрим простейшую версию уравнения теплопроводности:

Аппроксимируем вторую производную разностным оператором второго порядка:

Выполним замену:, после этого в матричном виде можно записать:

*.* Далее к этому выражение нужно применить метод Рунге-Кутта.

# Метод Рунге-Кутта 4 порядка

Cпомощью данного метода выражение вида можно вычислить итерационно:

В данном случае, так как A матрица, то и коэффициенты будут векторными.

Лабораторной поставлена немного другая задача (с ненулевыми граничными условиями, и с внешней функцией воздействия f), но полученные формулы легко дополняются, в частности функция будет иметь вид:

## Сходимость

Чтобы данный метод сходился необходимо чтобы шаг по времени был меньше квадрата шага по x, т. е. . Порядок сходимости данного метода оценивается .

# Параллельная реализация

Реализация кода по формулам выполняется несложно (хотя нужно соблюдать аккуратность с коэффициентами).

Компоненты каждого векторного коэффициента можно вычислять параллельно. Также компоненты можно вычислять параллельно. Единственная проблема, что внутри таких параллельных циклов довольно мало вычислений. В таком случае доля накладных расходов связанных с синхронизацией после каждого параллельного цикла, довольна большая, что плохо сказывается на масштабируемости. Также «вредит» большое количество операций чтения/записей относительно количества вычислительных операций.

Общий вычислительный цикл выглядит так:

for (int tick = 0; tick < m; tick++)

{

curTime = dt \* tick;

v[0] = task.left\_condition(curTime);

v[n] = task.right\_condition(curTime);

calc\_k1(task, v, k1, curTime);

calc\_k2(task, v, k1, k2, curTime + dt/2, tmp.data());

calc\_k3(task, v, k2, k3, curTime + dt/2, tmp.data());

calc\_k4(task, v, k3, k4, curTime + dt, tmp.data());

calc(task, v, k1, k2, k3, k4, curTime);

}

Пример вычисления коэффициента k2 (вычисления других k аналогично):

void calc\_k2(heat\_task &task, const double \*v, const double \*k1, double \*k2, double t,

double \*tmp)

{

const int n = task.n;

const double h = task.L / n;

const double coeff = 1 / (h\*h);

const double dt = task.T/(task.m \* 2.0);

tmp[0] = task.left\_condition(t);

tmp[n] = task.right\_condition(t);

#pragma omp parallel for

for (int i = 1; i < n; i++)

tmp[i] = v[i] + dt \* k1[i];

#pragma omp parallel for

for (int i = 1; i < n; i++)

k2[i] = coeff \* (tmp[i + 1] - 2 \* tmp[i] + tmp[i - 1]) + task.f(h\*i, t);

}

## Проверка корректности

Для проверки корректности была составлена задача с заранее известным аналитическим ответом. Это делалась следующим образом:

Была выбрана функция

Из условия нашей задачи:

Отсюда:

Таким образом была получена задача с известным аналитическим решением:

Для проверки корректности сравнивался ответ полученный алгоритмом, с аналитическим ответом. Использовались следующие параметры:

T = 0.1, n = 50000, m = 50000, ,

Норма вида бесконечность от разности численного и аналитического решения не превышала . При этом, при увеличении h в 10 раз «ошибка» увеличилась приблизительно в 100 раз до , что соотносится с порядком сходимости метода .

# Вычислительные эксперименты

Решалась задача с длиной стрежня L=100, временем T = 0.1, n = 50000, m = 50000 (шаг по времени должен быть ограничен шагом по пространству), , . Функция f и начальные/граничные условия использовались такие же, как в задаче с проверкой корректности.

Запуск производился на восьми ядерном процессоре Intel core i7 9700K, 16 gb ОЗУ. Использовался Intel® C++ Compiler 19.1 for Windows\* с оптимизацией O2.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Время, сек. | 1 core | 2 core | 4 core | 6 core | 8 core |
|  | 11.00 | 5.60 | 2.85 | 1.95 | 1.57 |

Таблица 1. Время решения задачи.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Масштабируемость | 1 core | 2 core | 4 core | 6 core | 8 core |
|  | 1.00 | 0.98 | 0.96 | 0.94 | 0.88 |

Таблица 2. Масштабируемость алгоритма.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Ускорение | 1 core | 2 core | 4 core | 6 core | 8 core |
|  | 1.00 | 1.96 | 3.86 | 5.64 | 7.01 |

Таблица 3. Ускорение алгоритма.

# Заключение

Численный эксперимент показал, что можно получить довольно неплохую масштабируемость (88% на 8 ядрах) при правильно подобранных n и m. Важно, чтобы n было довольно большим, чтобы время синхронизации занимало малую долю времени от общего времени вычислений. Еще одна проблема заключается в том, что параллелится цикл, сложность которого линейно зависит от количества данных. Таким образом увеличение размера данных, увеличивает число чтений/записей пропорционально числу вычислений. Таким образом, производительность упирается в скорость «памяти», а не процессора. Можно заметить, что чем сложнее функция f, тем больше будет вычислений, тем лучшей масштабируемости можно будет добиться. В целом, распараллеливание методов частичной дискретизации и Рунге-Кутта может принести значительное ускорение (в 7 раз на 8 ядрах).

# Список литературы

1. Баркалов К.А. Параллельные численные методы.
2. Самарский А. А. Введение в численные методы. – Лань, 2009.