



Московский государственный университет имени М.В. Ломоносова

Факультет вычислительной математики и кибернетики

Кафедра интеллектуальных информационных технологий

**Отчет о выполнении задания №2 по курсу
«Суперкомпьютерное моделирование и
технологии»**

Выполнил:

Вешкин Артемий Сергеевич

студент 622 группы

вариант №5

Москва, 2023

Оглавление

| | | |
|----------|--|----------|
| 1 | Описание задания | 3 |
| 1.1 | Описание численной схемы | 3 |
| 1.2 | Оптимизация работы алгоритма с помощью директив OpenMP | 4 |
| 2 | Программная реализация | 5 |
| 3 | Визуализация расчетов | 6 |
| 4 | Анализ параллельных свойств программы | 7 |

1. Описание задания

В задании требуется реализовать решение трехмерного гиперболического уравнения в прямоугольной области:

$$\frac{\partial^2 u}{\partial t^2} = a^2 \Delta u$$

$$u|_{t=0} = \varphi(x, y, z)$$

$$\left. \frac{\partial u}{\partial t} \right|_{t=0} = 0$$

$$u(0, y, z, t) = u(L_{x'}, y, z, t) \quad u_x(0, y, z, t) = u_x(L_{x'}, y, z, t)$$

$$u(x, 0, z, t) = 0 \quad u(x, L_y, z, t) = 0$$

$$u(x, y, 0, t) = 0 \quad u(x, y, L_z, t) = 0$$

1.1. Описание численной схемы

Для численного решения данного уравнение предлагается использовать двухшаговую явную разностную схему. В ней значения на $n + 1$ шаге по времени вычисляются с использованием значений на шагах $n - 1$ и n . Схема выглядит следующим образом:

$$\frac{u_{ijk}^{n+1} - 2u_{ijk}^n + u_{ijk}^{n-1}}{\tau^2} = a^2 \Delta_h u^n \Leftrightarrow u_{ijk}^{n+1} = \tau^2 a^2 \Delta_h u^n + 2u_{ijk}^n - u_{ijk}^{n-1}$$

где $\Delta_h u^n$ - семиточечный оператор Лапласа:

$$\Delta_h u^n = \frac{u_{i-1,j,k}^n - 2u_{i,j,k}^n + u_{i+1,j,k}^n}{h^2} + \frac{u_{i,j-1,k}^n - 2u_{i,j,k}^n + u_{i,j+1,k}^n}{h^2} + \frac{u_{i,j,k-1}^n - 2u_{i,j,k}^n + u_{i,j,k+1}^n}{h^2}$$

Для начала вычислений требуется знать $u_{i,j,k}^0$ и $u_{i,j,k}^1$. $u_{i,j,k}^0$ задается из начального условия, $u_{i,j,k}^1$ вычисляется следующим образом:

$$u_{ijk}^1 = u_{ijk}^0 + a^2 \frac{\tau^2}{2} \Delta_h \varphi(x_i, y_j, z_k)$$

Для оценки погрешности схемы и инициализации первого временного слоя в задании дано аналитическое решение:

$$u_{analytical} = \sin\left(\frac{2\pi}{L_x}x\right) \cdot \sin\left(\frac{\pi}{L_y}y\right) \cdot \sin\left(\frac{\pi}{L_z}z\right) \cdot \cos(a_t \cdot t + 2\pi),$$

$$a_t = \frac{1}{2} \sqrt{\frac{4}{L_x^2} + \frac{1}{L_y^2} + \frac{1}{L_z^2}}, a^2 = \frac{1}{4\pi^2}$$

1.2. Оптимизация работы алгоритма с помощью директив OpenMP

Явная схема вычисления следующего шага по времени позволяет эффективно распараллеливать вычисления. В данном задании предлагается для оптимизации вычислений использовать директивы OpenMP. Требуется исследовать эффекты от ускорения программы для разных размеров задач и для разного числа нитей. Для вычислений и замеров используется суперкомпьютер IBM Polus.

2. Программная реализация

Программа, реализующая описанную выше разностную схему реализована на языке C++ и выложена на GitHub (https://github.com/ArtemVeshkin/MSU_OpenMP_Task). Для компиляции и запуска использовались подобные команды:

```
g++ -o main.out main.cpp -std=c++11 -fopenmp  
mpisubmit.pl -t 8 main.out --stdout std.out --stderr std.err -- 128 20 1. 1. 1. 0.01
```

Верхнеуровнево код выглядит следующим образом:

1. Обрабатываются аргументы командной строки
2. Создаются сетки u^0 и u^1
3. С помощью аналитического решения инициализируется u^0 и границы u^1
4. Вычисляются значения в узлах u^1 с помощью значений в узлах u^0
5. Далее вычисляются значения в узлах u^{n+1} с использованием значений в u^n и u^{n-1} начиная с $n = 2$
6. Выводится время работы программы и значения ошибок

Также в процессе работы программы имеется возможность залогировать в текстовом виде значения ошибок на всех шагах и содержимое узлов сеток. Для этого используются аргументы командной строки **log_grids** и **log_errors**.

В некоторых местах программы используется директива **#pragma omp parallel for**. Она позволяет выполнять цикл for параллельно с использованием заданного при старте программы числа нитей. С помощью этой директивы ускорены следующие фрагменты программы:

- Очистка узлов пространственной сетки
- Перенос значений узлов с одной сетки на другую
- Инициализация сеток u^0 и u^1
- Вычисление значений в узлах u^1
- Вычисление значений в узлах u^{n+1}

3. Визуализация расчетов

Для визуализации работы программы был произведен расчет на сетке $L_x = 1, L_y = 1, L_z = 1, T = 1$, с 128 пространственными и 256 временными шагами. Сетка на каждом временном шаге сохранена и визуализированна при помощи языка python и его пакета matplotlib.

Для визуализации в трехмерном пространстве были отрисованы значения узлов сетки а также степень их отклонения от эталонных для каждого момента времени. GIF-анимации визуализаций находятся в GitHub репозитории (https://github.com/ArtemVeshkin/MSU_OpenMP_Task).

Ниже прикладываю поведение значений ошибок для описанной выше сетки.

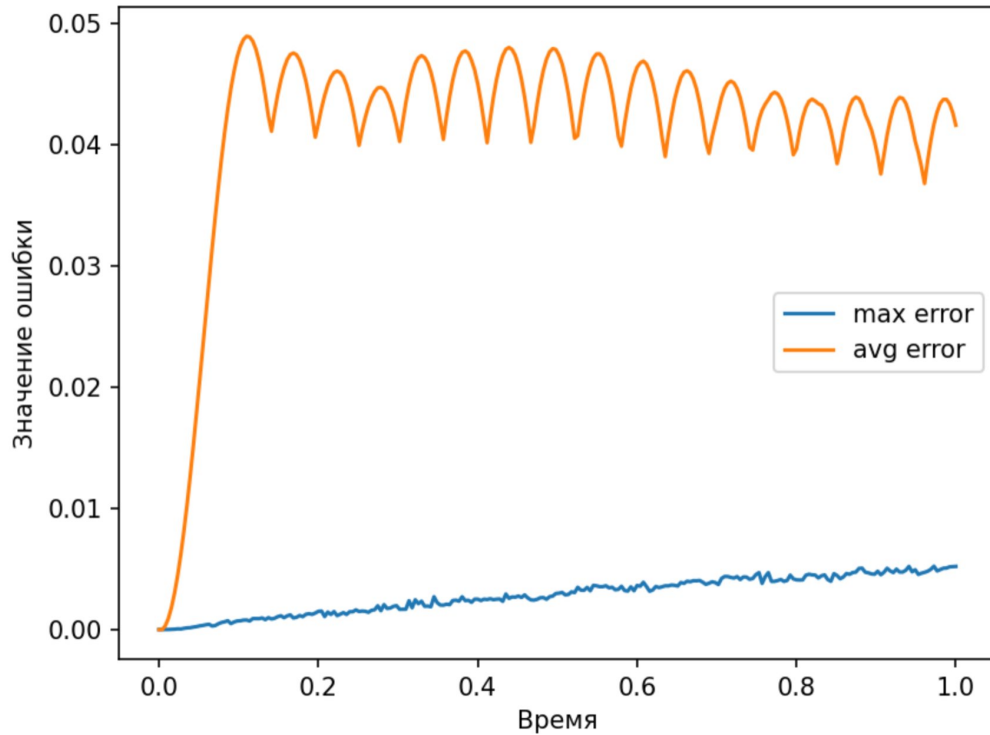


Рис. 1. Значения средней и максимальной ошибок для сетки $L_x = 1, L_y = 1, L_z = 1, T = 1$, с 128 пространственными и 256 временными шагами

4. Анализ параллельных свойств программы

Для разных пространственных сеток и разного числа OpenMP нитей были проведены замеры эффективности работы программы. Результаты приведены в таблице ниже:

| <i>Число OpenMP нитей</i> | <i>Число точек сетки N^3</i> | <i>Время решения</i> | <i>Ускорение</i> | <i>Средняя ошибка</i> | <i>Максимальная ошибка</i> |
|-----------------------------------|---|----------------------|------------------|---------------------------|--------------------------------|
| 1 | 128^3 | $2894ms$ | 1 | 0.000016 | 0.00086 |
| 2 | 128^3 | $1896ms$ | 1.52 | 0.000016 | 0.00086 |
| 4 | 128^3 | $1492ms$ | 1.94 | 0.000016 | 0.00086 |
| 8 | 128^3 | $1443ms$ | 2.01 | 0.000016 | 0.00086 |
| 16 | 128^3 | $1559ms$ | 1.86 | 0.000016 | 0.00086 |
| 32 | 128^3 | $1813ms$ | 1.59 | 0.000016 | 0.00086 |
| 1 | 256^3 | $22184ms$ | 1 | 0.000008 | 0.00017 |
| 2 | 256^3 | $14017ms$ | 1.58 | 0.000008 | 0.00017 |
| 4 | 256^3 | $11952ms$ | 1.86 | 0.000008 | 0.00017 |
| 8 | 256^3 | $9177ms$ | 2.42 | 0.000008 | 0.00017 |
| 16 | 256^3 | $11231ms$ | 1.98 | 0.000008 | 0.00017 |
| 32 | 256^3 | $13559ms$ | 1.64 | 0.000008 | 0.00017 |

Из проведенных экспериментов видно, что после определенного числа нитей программа перестает ускоряться и даже начинает работать немного медленнее чем при меньшем числе нитей. Скорее всего это связано с тем, что накладные расходы начинают доминировать над выигрышем от параллелизма.