Московский государственный университет имени М.В. Ломоносова

Факультет вычислительной математики и кибернетики

Кафедра вычислительных методов

Попко Федор Дмитриевич

Параллельная реализация явной разностной схемы для решения уравнения теплопроводности.

Отчет

Москва, 2024

Содержание

1 Постановка задачи

В данной работе рассматривается задача одномерной теплопроводности с источником. Уравнение теплопроводности описывается следующим образом:

$$\frac{\partial u}{\partial t} = A\frac{\partial^2 u}{\partial x^2} + f(x), \quad x \in [0, L], \quad t > 0, \tag{1}$$

где:

- $u(x, t)$ — температура в точке x в момент времени t,
- A — коэффициент теплопроводности,
- $f(x)$ — функция источника тепла.

Граничные условия:

$$u(0, t) = u(L, t) = 0, \quad t \geq 0. \tag{2}$$

Начальное условие:

$$u(x, 0) = \sin\left(\frac{\pi x}{L}\right), \quad x \in [0, L]. \tag{3}$$

Функция источника задаётся в виде:

$$f(x) = \sin\left(\frac{\pi x}{L}\right). \tag{4}$$

Основная цель эксперимента — вычислить распределение температуры $u(x, t)$ на отрезке $x \in [0, L]$ в течение времени $t \in [0, T]$, используя численный метод решения уравнения теплопроводности с разностной схемой.

2 Выбор алгоритма для параллельной реализации

Для решения одномерного уравнения теплопроводности используется явная разностная схема

2.1 Разностная схема

Для численного решения используется явная схема:

$$u_i^{n+1} = u_i^n + C\left(u_{i+1}^n - 2u_i^n + u_{i-1}^n\right) + \Delta t \cdot f(x_i), \tag{5}$$

где:

- u_i^n — значение температуры в i-й точке на временном слое n,
- $C = \frac{A\Delta t}{\Delta x^2}$ — число Куранта,
- Δx — шаг по пространству,
- Δt — шаг по времени,
- $f(x_i) = \sin\left(\frac{\pi x_i}{L}\right)$ — источник тепла.

Данная схема накладывает ограничение на величину числа Куранта $C \leq 0.5$ для обеспечения устойчивости.

2.2 Алгоритм параллельных вычислений

Для распараллеливания задачи используется распределение по пространственной координате x. Весь стержень делится на P подотрезков, где P — количество процессов. Каждый процесс отвечает за вычисление температуры в своей части стержня.

1. В начальный момент времени каждый процесс инициализирует свою часть массива u_i в соответствии с начальным условием $u(x, 0) = \sin\left(\frac{\pi x}{L}\right)$.
2. На каждом временном шаге процессы обмениваются граничными значениями температуры с соседями:
   - Левый сосед передаёт правую граничную точку соседу слева.
   - Правый сосед передаёт левую граничную точку соседу справа.
3. После обмена данными каждый процесс обновляет значения температуры в своей области, используя разностную схему.
4. Процессы периодически собирают данные на процесс с рангом 0 для записи промежуточных результатов.

2.3 Реализация

Параллельная реализация осуществляется с использованием библиотеки MPI (Message Passing Interface), позволяющей распределить вычисления между несколькими процессами. Основные функции MPI, используемые в алгоритме:

- `MPI_Init`, `MPI_Finalize` — инициализация и завершение работы с MPI.

- `MPI_Comm_rank`, `MPI_Comm_size` — определение ранга текущего процесса и общего числа процессов.
- `MPI_Isend`, `MPI_Irecv` — асинхронный обмен данными между процессами.
- `MPI_Gather` — сбор данных со всех процессов на один процесс.

3 Численные эксперименты и анализ масштабируемости

Для оценки производительности алгоритма были проведены численные эксперименты на суперкомпьютере «Ломоносов-2». В рамках экспериментов исследовалась сильная масштабируемость алгоритма, а также проводилось сравнение численного решения с аналитическим.

3.1 Параметры экспериментов

Для моделирования использовались следующие параметры:

- Длина стержня: $L = 100.0$;
- Время моделирования: $T = 10.0$;
- Коэффициент теплопроводности: $A = 0.1$;
- Общее число узлов по пространству: $N_x = 16384, 32768$;
- Количество процессов: $P = 1, 2, 4, 8, 16, 32$.

Численные эксперименты включали:

1. Изучение сильной масштабируемости: фиксировался размер задачи ($N_x = 16384, 32768$), а число процессов увеличивалось от 1 до 32.
2. Сравнение численного и аналитического решения на нескольких временных слоях.

3.2 Метрики масштабируемости

Для анализа производительности использовались следующие метрики:

- **Время выполнения**:

$$T_{\text{exec}}(P) = \text{общее время выполнения задачи для } P \text{ процессов.}$$

- **Ускорение**:

$$S(P) = \frac{T_{\text{exec}}(1)}{T_{\text{exec}}(P)},$$

где $T_{\text{exec}}(1)$ — время выполнения на 1 процессе.

- **Эффективность**:

$$E(P) = \frac{S(P)}{P} = \frac{T_{\text{exec}}(1)}{P \cdot T_{\text{exec}}(P)}.$$

3.3 Результаты сильной масштабируемости для $N_x = 16384$

На Рисунке 1 представлен график сильной масштабируемости, показывающий зависимость времени выполнения T_{exec} от числа процессов P. Ускорение и эффективность представлены на Рисунках 2 и 9 соответственно.

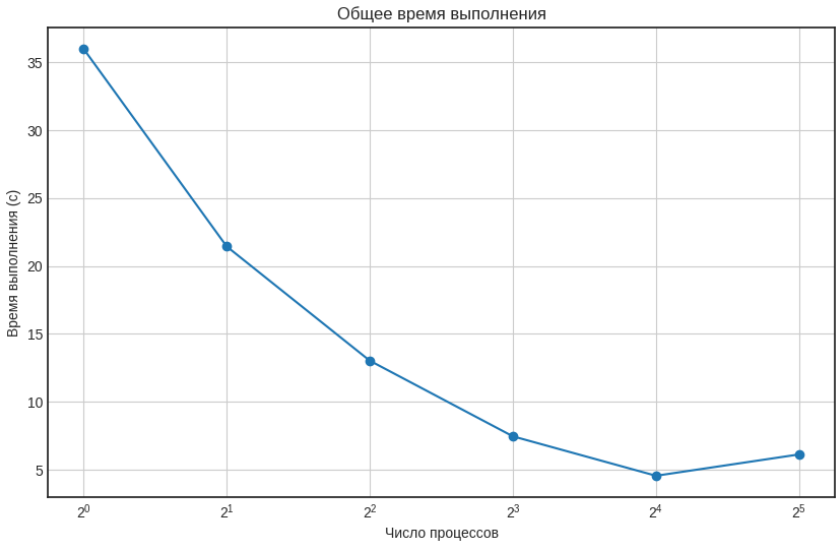

Рис. 1: График сильной масштабируемости.

На графике 1 видно, что при увеличении числа процессов P время выполнения T_{exec} уменьшается. Однако при значениях $P >= 32$ снижение времени выполнения замедляется из-за накладных расходов на обмен данными между процессами. Это отражается и на графике ускорения 2

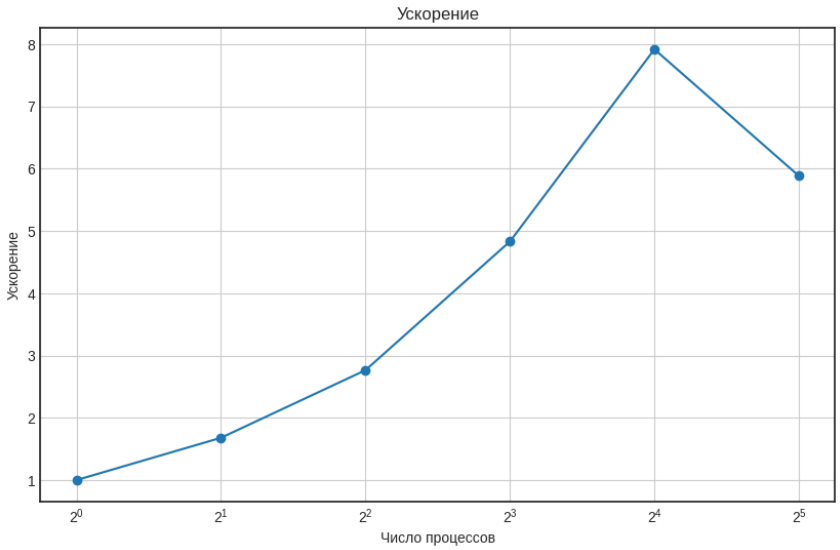

Рис. 2: График ускорения.

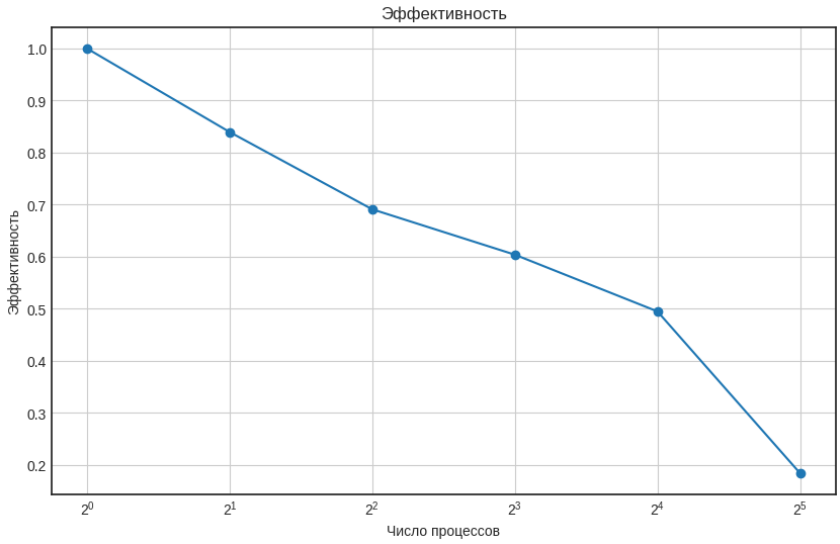

Рис. 3: График эффективности.

Эффективность $E(P)$ близка к единице при малых P, но начинает снижаться с увеличением числа процессов, в связи с увеличением времени, затрачиваемого на коммуникацию между процессами.

3.4 Результаты сильной масштабируемости для $N_x = 32768$

На Рисунке 4 представлен график сильной масштабируемости, показывающий зависимость времени выполнения T_{exec} от числа процессов P. Ускорение и эффективность представлены на Рисунках 5 и 6 соответственно.

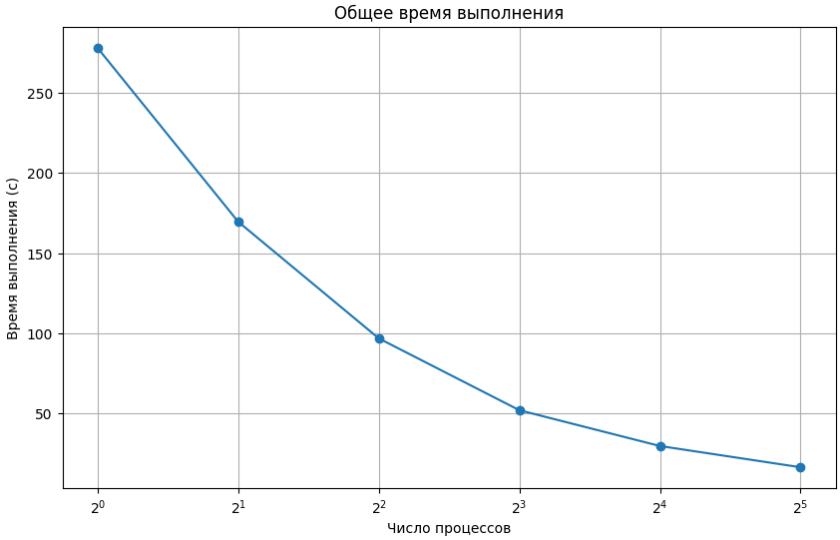

Рис. 4: График сильной масштабируемости.

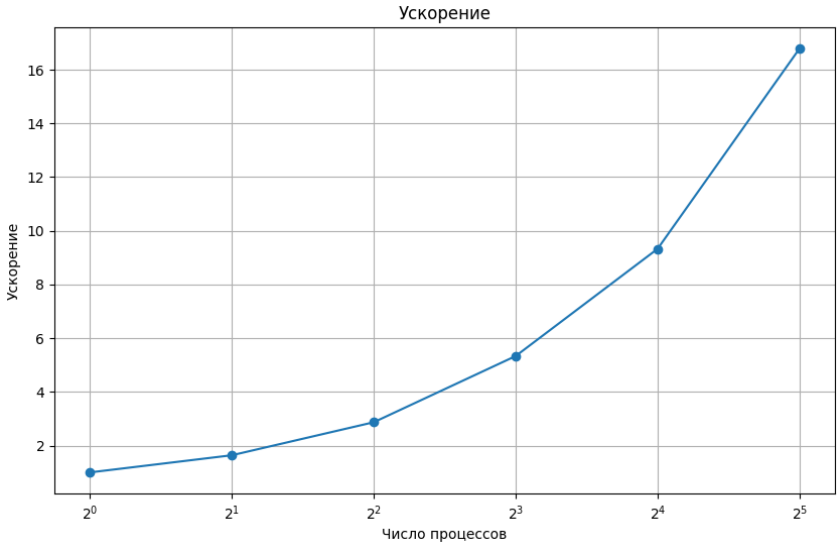

Рис. 5: График ускорения.

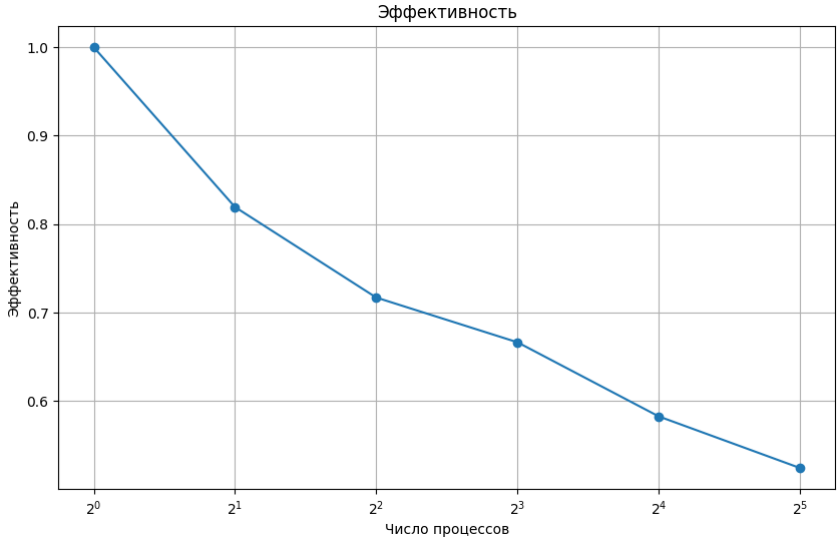

Рис. 6: График эффективности.

3.5 Сравнение численного и аналитического решения

Для проверки корректности численного алгоритма решение сравнивалось с аналитическим:

$$u(x,t) = \sin\left(\frac{\pi x}{L}\right)\left[\frac{L^2}{\pi^2 A} + \left(1 - \frac{L^2}{\pi^2 A}\right) e^{-A(\pi^2/L^2)t}\right].$$

На Рисунке 7 приведены графики численного и аналитического решений для различных временных шагов, а также погрешность численного решения.

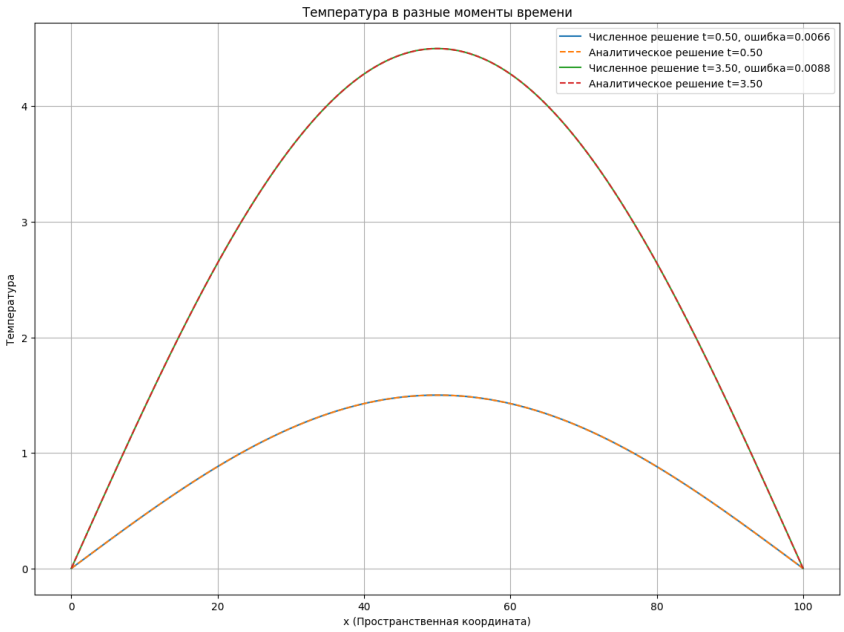

Рис. 7: Сравнение численного и аналитического решений.

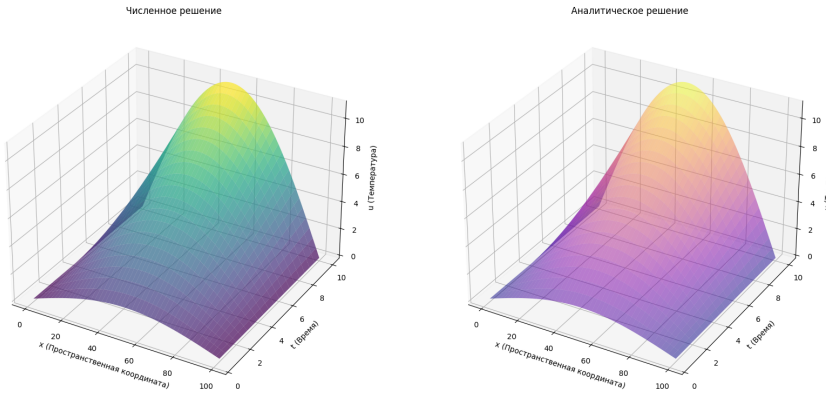

Рис. 8: Численное и аналитическое решения.

3.6 Выводы

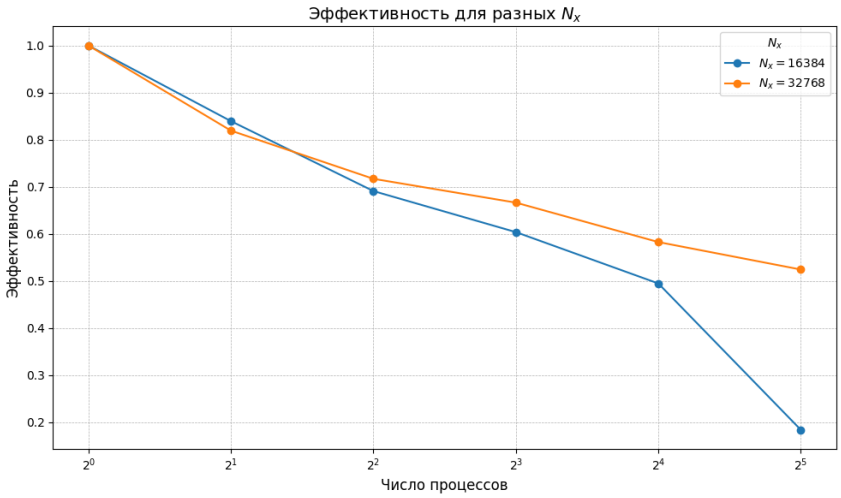

Рис. 9: График эффективности для разных разбиений по пространству

Эффективность $E(P)$ для обоих значений N_x уменьшается с ростом числа процессов P. Для задачи с $N_x = 32768$ эффективность выше, чем для $N_x = 16384$, при одинаковом числе процессов, так как при большем размере задачи вычислительная нагрузка на каждый процесс возрастает, а влияние накладных расходов на коммуникацию уменьшается.

3.7 Программно-аппаратная среда

Аппаратное обеспечение: вычисления проводились на вычислительном узле с архитектурой **x86_64**, оснащённом двумя процессорами **Intel Xeon E5-2697 v3** с тактовой частотой от 1.2 ГГц до 3.6 ГГц. Каждый процессор содержит 14 физических ядер и поддерживает до 28 потоков, в сумме даёт 56 логических процессоров.

Оперативная память составляет 62 ГБ, из которых 33 ГБ доступно для вычислений, а 16 ГБ использовались в качестве буферов.

Программное обеспечение: для компиляции использовался компилятор **GCC** версии 4.8.5 с поддержкой стандарта `C99` и опцией оптимизации второго уровня `-O2`. Для выполнения программы применялась библиотека `MPI` версии `2021.2.0` от Intel OneAPI, обеспечивающая параллельное взаимодействие между процессами. Подключённые библиотеки:

- `libm.so.6` — стандартная математическая библиотека;
- `libmpifort.so.12` — интерфейс MPI для Fortran;
- `libmpi.so.12` — основной интерфейс MPI;
- `librt.so.1` — библиотека для работы с реальным временем;
- `libpthread.so.0` — библиотека потоков POSIX;
- `libdl.so.2` — динамическая загрузка;
- `libc.so.6` — стандартная библиотека C.

Сборка программы выполнялась с использованием следующего Makefile:

```
# Compiler
CC = mpicc

# Compiler flags
CFLAGS = -std=c99 -O2

# Target executable name
TARGET = heat_equation_mpi

# Source file
SRC = heat_equation_mpi.c

# Math library
LIBS = -lm
```

```
# Default number of processes, which can be overridden from the command line
NP ?= 4

# Default rule: build the target
all: $(TARGET)

# Rule to build the executable
$(TARGET): $(SRC)
    $(CC) $(CFLAGS) -o $(TARGET) $(SRC) $(LIBS)

# Rule to run the program
run: $(TARGET)
    mpirun -np $(NP) ./$(TARGET)

# Rule to clean the build files and all files generated by the build result_*
clean:
    rm -f $(TARGET)
    rm -f *.dat
```

3.8 Ссылки на исходный код и алгоритмы

Код программы и подробное описание алгоритма доступны в следующих источниках:

- Исходный код программы: GitHub репозиторий.
- Алгоритмы и теория: Алговики.