Московский физико-технический институт

Проект по вычислительной математике

Уравнение теплопроводности

Студент: Пучков Кирилл
Группа: 774

Москва
2020

Формулировка задачи

Уравнение теплопроводности — дифференциальное уравнение в частных производных второго порядка, которое описывает распределение температуры в заданной области пространства и ее изменение во времени.

В пространстве с произвольной системой координат $\mathbf{r} = (r_1, \ldots, r_n)$ уравнение теплопроводности имеет вид

$$\frac{\partial u}{\partial t} - a^2 \Delta u = f(\mathbf{r}, t),$$

где a — положительная константа (число a^2 является коэффициентом температуропроводности), $\Delta = \nabla^2$ — оператор Лапласа и $f(\mathbf{r}, t)$ — функция тепловых источников. Искомая функция $u = u(\mathbf{r}, t)$ задает температуру в точке с координатами $\mathbf{r}$ в момент времени t.

Рассмотрим задачу Коши для однородного уравнения теплопроводности:

$$\begin{gathered} \frac{\partial u}{\partial t} - a^2 \Delta u = 0, \quad x \in \mathbb{R}^n,\ t > 0, \\ u(x,\ 0) = \varphi(x), \quad x \in \mathbb{R}^n, \end{gathered}$$

где $\varphi(x)$ — начальная функция, непрерывная и ограниченная на всём пространстве, и искомая функция $u = u(x, t)$ является непрерывной и ограниченной при $t \geqslant 0$ и всех значениях аргумента x.

Также договоримся, что будем рассматривать случай одной пространственной переменной x (задача о нагревании или охлаждении стержня). Тогда уравнение теплопроводности принимает вид:

$$u_t - a^2 u_{xx} = f(x, t)$$

Переформулируем условие:

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2}, x \in [0, 1], u(x, 0) = \varphi(x), u(0, t) = u_1, u(1, t) = u_r$$

Разностная аппроксимация уравнения теплопроводности

Введем в области $\overline{D} = \{0 \leq x \leq l, 0 \leq t \leq T\}$ равномерную сетку с шагом h по координате и шагом τ по времени:

$$x_i = i \cdot h, i = 0, 1, \ldots, N, h \cdot N = l;$$
$$t_j = j \cdot \tau, j = 0, 1, \ldots, M, \tau \cdot M = T.$$

Уравнение теплопроводности содержит как производные по пространственной переменной x, так и по времени t, поэтому для построения его разностной аппроксимации придется использовать узлы сетки, соответствующие различным j. Все узлы сетки, отвечающие фиксированному j, называют j-м временным слоем. Свойства разностных схем зависят от того, на каком слое j по времени аппроксимируется выражение $\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2}$.

Явная схема

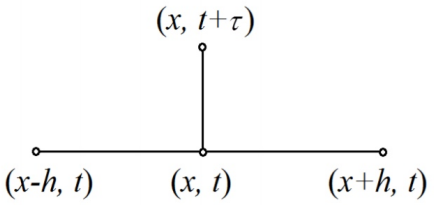


Рис. 1: Шаблон явной схемы для уравнения теплопроводности

Соответствующий разностный оператор $L^{(0)}_{h\tau}u$ имеет вид:

$$L^{(0)}_{h\tau}u = \frac{u(x, t+\tau) - u(x,t)}{\tau} - \frac{u(x+h,t) - 2 \cdot u(x,t) + u(x-h,t)}{h^2}$$

Далее для краткости будем использовать следующие стандартные обозначения:

$$u = u(x,t); \hat{u} = u(x, t+\tau).$$

Тогда:

$$u_t = \frac{\hat{u} - u}{\tau}, L^{(0)}_{h\tau}u = u_t - u_{\overline{x}x}$$

Найдем погрешность аппроксимации разностным оператором $L^{(0)}_{h\tau}u$ исходного дифференциального оператора L в точке (x,t). В случае достаточно гладкой функции $u(x,t)$ при достаточно малых шагах h и τ имеем:

$$u_t = \frac{u(x, t+\tau) - u(x,t)}{\tau} = \frac{\partial u(x,t)}{\partial t} + O(\tau),$$

$$u_{\overline{x}x} = \frac{\partial^2 u}{\partial x^2} + O(h^2).$$

Следовательно, разностный оператор $L^{(0)}_{h\tau} u$ аппроксимирует дифференциальный оператор L с погрешностью $O(\tau + h^2)$ в точке (x, t). Тогда разностное уравнение $L^{(0)}_{h\tau} u = 0$ будет аппроксимировать исходное дифференциальное уравнение теплопроводности с первым порядком погрешности по τ и вторым по h.

Реализация

Значение сеточной функции на верхнем временном слое $j+1$ рассчитывается по ее значениям на нижнем слое p. Для удобства положим $\varphi(x) = 0$.

$$u_m^{n+1} = u_m^n + \frac{\tau}{h^2} \cdot (u_{m1}^n 2 \cdot u_m^n + u_{m+1}^n).$$

Тогда последовательный алгоритм выглядит следующим образом:

for $n = 0...N$ **do**
  for $m = 0...M$ **do**
    $u_m^{n+1} = u_m^n + \frac{\tau}{h^2}(u_{m-1}^n - 2u_m^n + u_{m+1}^n)$
  end for
  $u^n = u^{n+1}$
end for
save u^n

Рис. 2:

Message Passing Interface (MPI)

Постановка задачи

Решить одномерное уравнение теплопроводности при заданных граничных условиях и начальном распределении температуры $f(x)$.

Входные параметры:

1. Момент времени $Time$, в который требуется узнать распределение температуры.

2. Число разбиения координаты M.

Результат работы программы — распределение температуры в стержне.
Требования:

1. Распараллелить программу, используя среду MPI (обязательно использовать вызовы MPISend/MPIRecv

2. Построить графики ускорения и эффективности для числа процессов от 1 до 28

Решение

Температура покоординатно хранится в двух массивах, содержащих её распределение в момент n и $n+1$ соответсвенно.

```
// Массивы температуры для момента времени n и n + 1 соотв
етственно
   double *u0 = (double*) malloc(sizeof(double) * M);
   double *u1 = (double*) malloc(sizeof(double) * M);
```

Рис. 3: Массивы температур

Для распараллеливания на процессы данные декомпозируются.

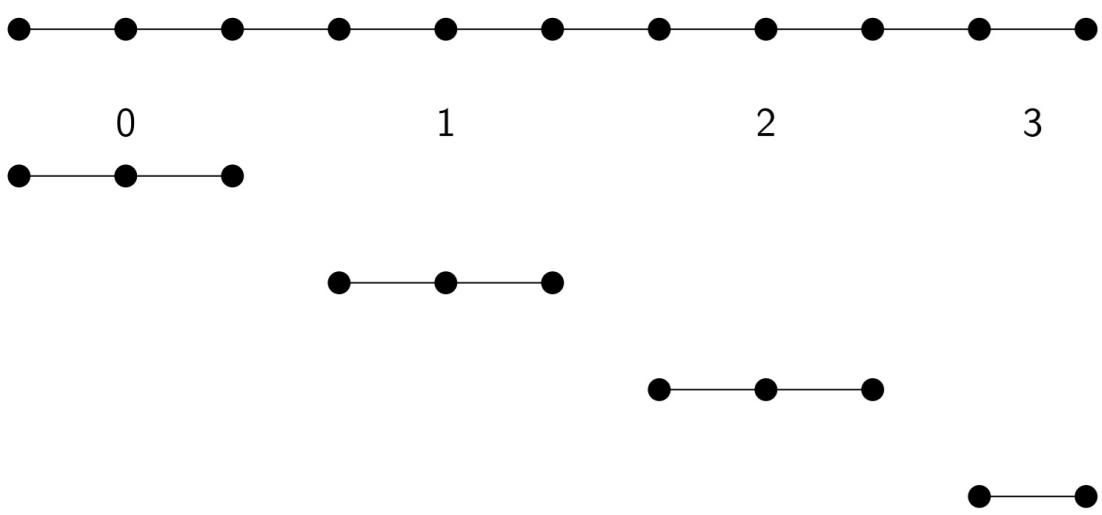


Рис. 4: Декомпозиция

Число разбиений по координате M делится на требуемое количество процессов $size$ равномерно, далее каждый процесс вычиляет свои заны ответственности (индексы подсчета). i-ый процесс производит вычисления между i-ым левым индексом и $i+1$-ым (не включая). Так как для вычисления значения m-ого элемента требуются значения $m, m-1, m+1$ элементов, то во время подсчета краевых значений происходит обмен краевыми узлами. Я реализовал "честное"разделения данных, не отда-

вая предпочтение последнему процессу, а передавая первым процессам по одному узлу.

```
// Массив индексов передаваемых точек
    size_t *left_index = (size_t*) malloc(sizeof(size_t)
 * size + 1);
    left_index[0] = 1;

// Чтобы избежать костылей при передаче массивов 0-ому п
роцессу,
    // определяю правый конец последнего массива
    left_index[size] = M - 1;

// Итеративно определяю левые концы отрезков, передаваем
ые каждому процессу

// Правый конец i-го процесса = левому концу (i + 1)-го
    for(int i = 1; i < size; i++) {
        left_index[i] = left_index[i - 1] + (M / size) +
 ((i - 1) < ((M % size) - 2));
    }
```

Рис. 5: Вычисление зон ответственности

Обмен краевым узлами (Рис. 7) реализован за $O(size)$: i-ый процесс, кроме нулевого, отсылает свой неизмененный крайний левый элемент $i-1$ процессу и принимает измененный элемент для дальнейшего пересчета; i-ый процесс, кроме последнего, отсылает свой неизмененный крайний правый элемент $i+1$ процессу и принимает измененный элемент для дальнейшего пересчета. (Рис. 6)

Далее проводим вычисления $n+1$ временного слоя и обмен данными (Рис. 8).

После крайней итерации требуется обновление значений элементов всего массива, так как каждый процесс содержит только свои обновленные данные. Следовательно, требуется выбрать единый процесс, например нулевой, которому остальные процессы пришлют обновленные данные. (Рис. 9)

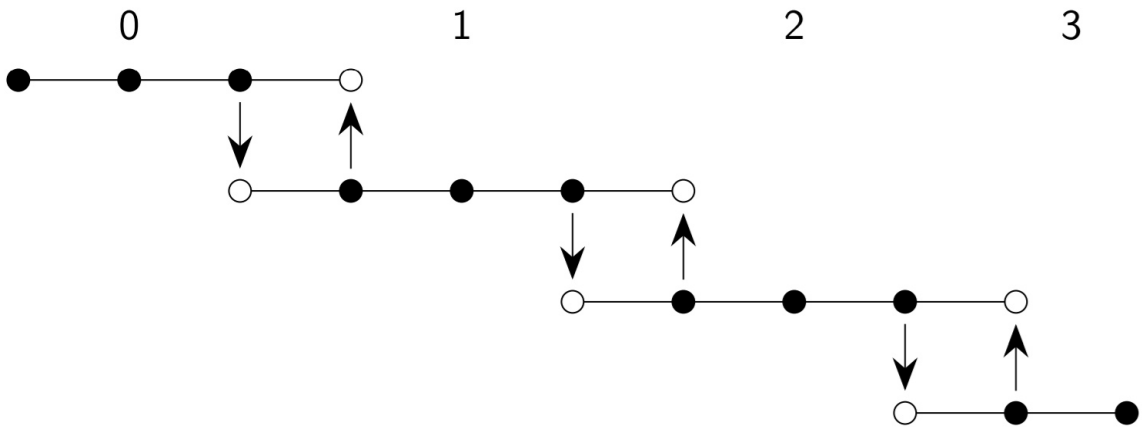


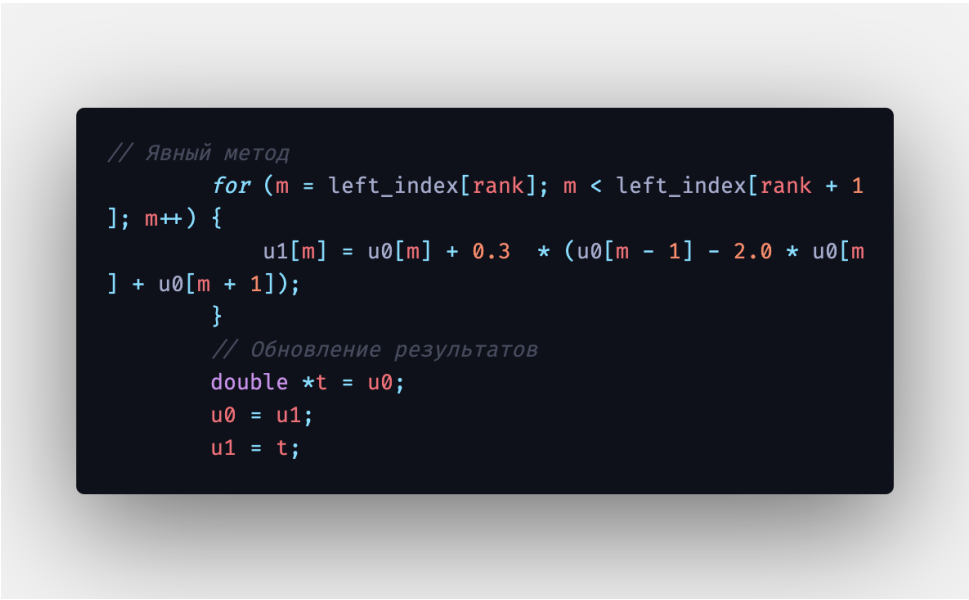
Рис. 6: Обмен краевыми узлами

```
// Обмен краевыми узлами
        if(rank ≠ 0) {
            MPI_Send(u0 + left_index[rank], 1,
MPI_DOUBLE, rank - 1, 0,
            MPI_COMM_WORLD);
            MPI_Recv(u0 + left_index[rank] - 1, 1,
MPI_DOUBLE, rank - 1, 0,
            MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        }
        if(rank ≠ size - 1) {
            MPI_Send(u0 + left_index[rank + 1] - 1, 1,
MPI_DOUBLE, rank + 1, 0,
            MPI_COMM_WORLD);
            MPI_Recv(u0 + left_index[rank + 1], 1,
MPI_DOUBLE, rank + 1, 0,
            MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        }
```

Рис. 7: Обмен краевыми узлами

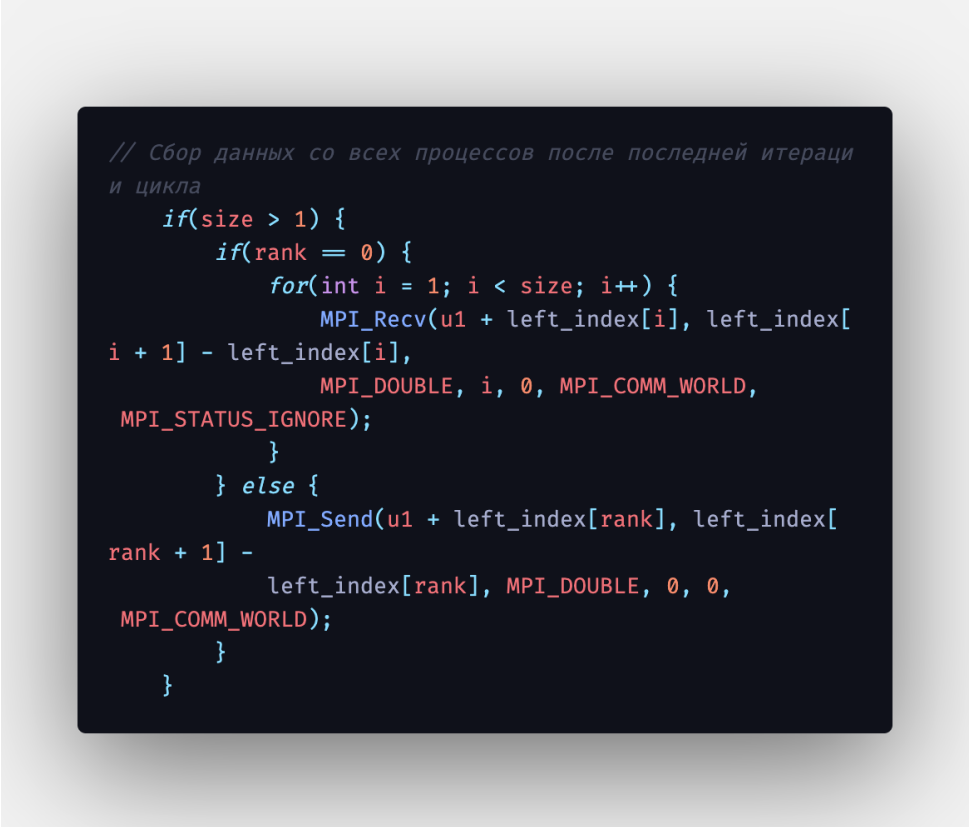
Результат

Построим распределение температуры от координаты в разные моменты времени. По теории распределение температуры при $time \to \infty$ стремит-



```
// Явный метод
        for (m = left_index[rank]; m < left_index[rank + 1
]; m++) {
            u1[m] = u0[m] + 0.3  * (u0[m - 1] - 2.0 * u0[m
] + u0[m + 1]);
        }
        // Обновление результатов
        double *t = u0;
        u0 = u1;
        u1 = t;
```

Рис. 8: Вычисление следующего слоя и обмен данными



```
// Сбор данных со всех процессов после последней итераци
и цикла
    if(size > 1) {
        if(rank == 0) {
            for(int i = 1; i < size; i++) {
                MPI_Recv(u1 + left_index[i], left_index[
i + 1] - left_index[i],
                MPI_DOUBLE, i, 0, MPI_COMM_WORLD,
 MPI_STATUS_IGNORE);
            }
        } else {
            MPI_Send(u1 + left_index[rank], left_index[
rank + 1] -
            left_index[rank], MPI_DOUBLE, 0, 0,
 MPI_COMM_WORLD);
        }
    }
```

Рис. 9: Сбор данных нулевым процессов

ся к линейному. Как видно на графике №10, наши результаты совпадают с теоретическими.

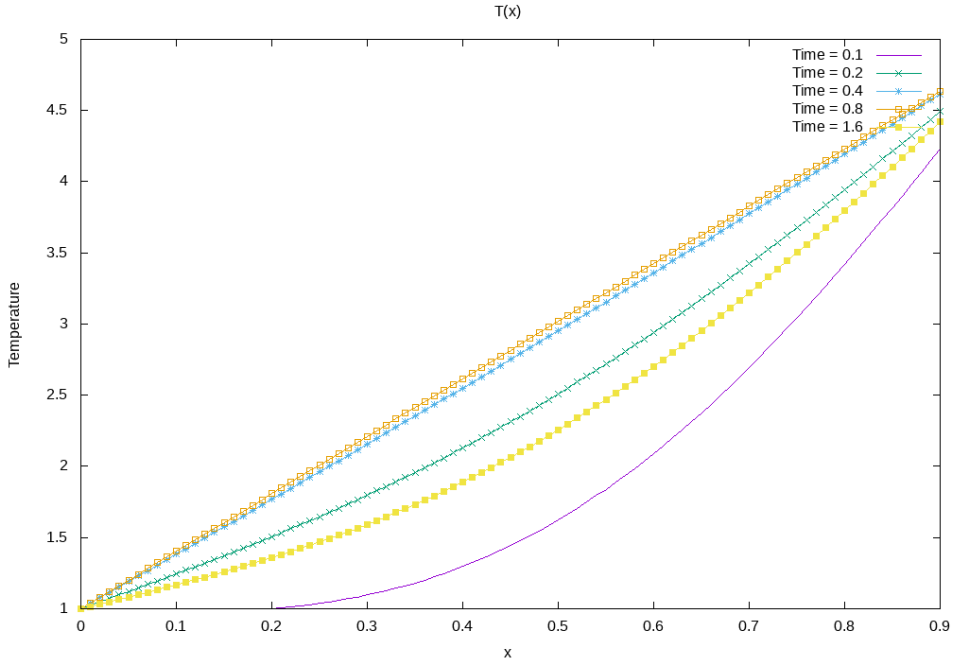


Рис. 10: Распределение температуры в зависимости от времени

Ускорение

Также было проведено исследование ускорение и эффективности алгоритма количестве процессов от 1 до 28.

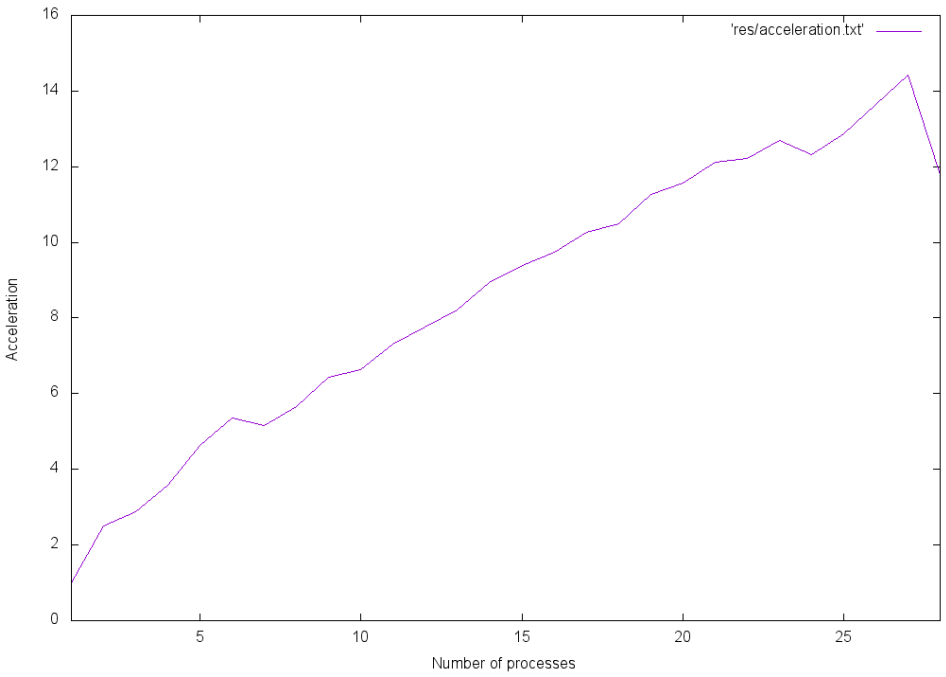


Рис. 11: Ускорение параллельного алгоритма

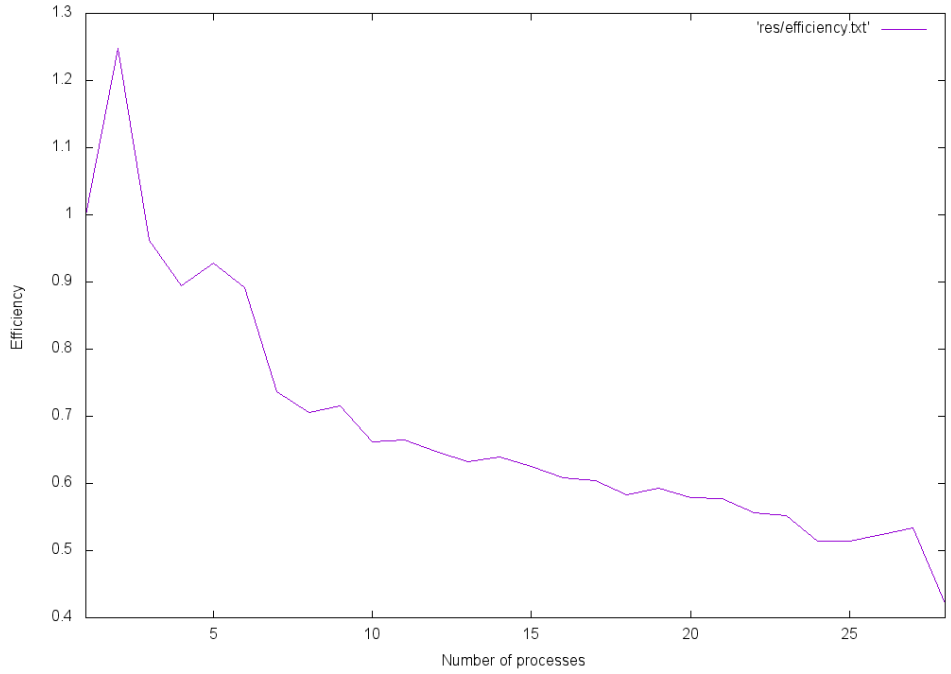


Рис. 12: Эффективность параллельного алгоритма

Open Multi-Processing (OMP)

Постановка задачи

Решить одномерное уравнение теплопроводности при заданных граничных условиях и начальном распределении температуры $f(x)$ используя механизмы замков *lock* на границах зон ответственности.

Входные параметры:

1. Момент времени $Time$, в который требуется узнать распределение температуры.

2. Число разбиения координаты M.

Результат работы программы — распределение температуры в стержне. Провести тестирование ускорения на числе потоков от 1 до 4.

Решение

Алгоритм решения задачи аналогичен предыдущему, за исключением использования механизма *locks*. Теперь взаимодействие на границах зон ответственности осуществляется с помощью замков, которых создано на паре на каждый процесс. (Рис. 13)

Работая с граничными узлами, мы блокируем один из замков соседа, считываем его неизмененное значение и потом проводим защищеное вычисление бокового узла, заблокировав свой замок. (Рис. 14)

```
// Создание замка
    omp_lock_t* lock = (omp_lock_t*)
malloc(sizeof(omp_lock_t) * 2 * size);
    //Инициализация замка
    for (size_t i = 0; i < 2 * size; ++i)
{
        omp_init_lock(&lock[i]);
    }
```

Рис. 13: Массив *lock*

```
if ((m == left_index[id]) && (id != 0)) {
                    // Запоминаем боковой узел
                    omp_set_lock(&lock[id - 1 + size]);
                    double left = u0[left_index[id] - 1];
                    omp_unset_lock(&lock[id - 1 + size]);

                    // Проводим защищенно вычисления
                    omp_set_lock(&lock[id]);
                    u1[m] = u0[m] + 0.3 * (left - 2.0 * u0[m] + u0[m + 1]);
                    omp_unset_lock(&lock[id]);
                }

                if ((m == left_index[id + 1] - 1) && (id != size - 1)) {
                    // Запоминаем боковой узел
                    omp_set_lock(&lock[id + 1]);
                    double right = u0[left_index[id + 1]];
                    omp_unset_lock(&lock[id + 1]);

                    // Проводим защищенно вычисления
                    omp_set_lock(&lock[id + size]);
                    u1[m] = u0[m] + 0.3 * (u0[m - 1] - 2.0 * u0[m] + right);
                    omp_unset_lock(&lock[id + size]);
                }

                u1[m] = u0[m] + 0.3 * (u0[m - 1] - 2.0 * u0[m] + u0[m + 1]);
```

Рис. 14: Защищенное вычисление

Чтобы значения обновлялись одновременно и никакой процесс не закончил цикл раньше, мы используем механизм эпох и атомарную секцию. (Рис. 15)

```
/ Атомарно инкрементируем, показывая, что процесс закончил работу
            #pragma omp atomic
            epoc++;

            #pragma omp single
            {
                // Не обновляем результат, пока не проработали все процессы
                while (epoc < size) {
                    __asm volatile ("pause" ::: "memory");
                }

                // Обновление результатов
                double *t = u0;
                u0 = u1;
                u1 = t;
            }
```

Рис. 15: Синхронизация на границах зон ответственности

Результат

Построим распределение температуры от координаты в разные моменты времени. По теории распределение температуры при $time \to \infty$ стремится к линейному. Как видно на графике №16, наши результаты совпадают с теоретическими.

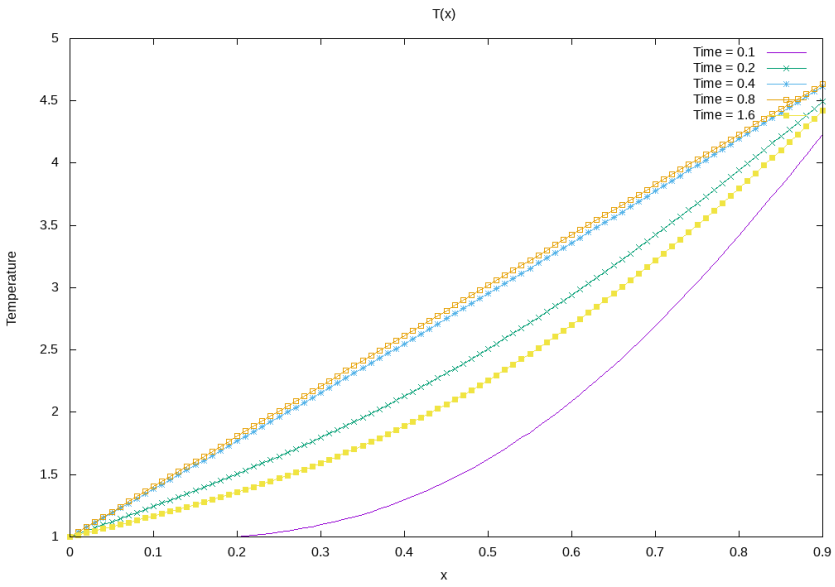


Рис. 16: Распределение температуры в зависимости от времени

Ускорение

Также было проведено исследование ускорение и эффективности алгоритма количестве процессов от 1 до 4.

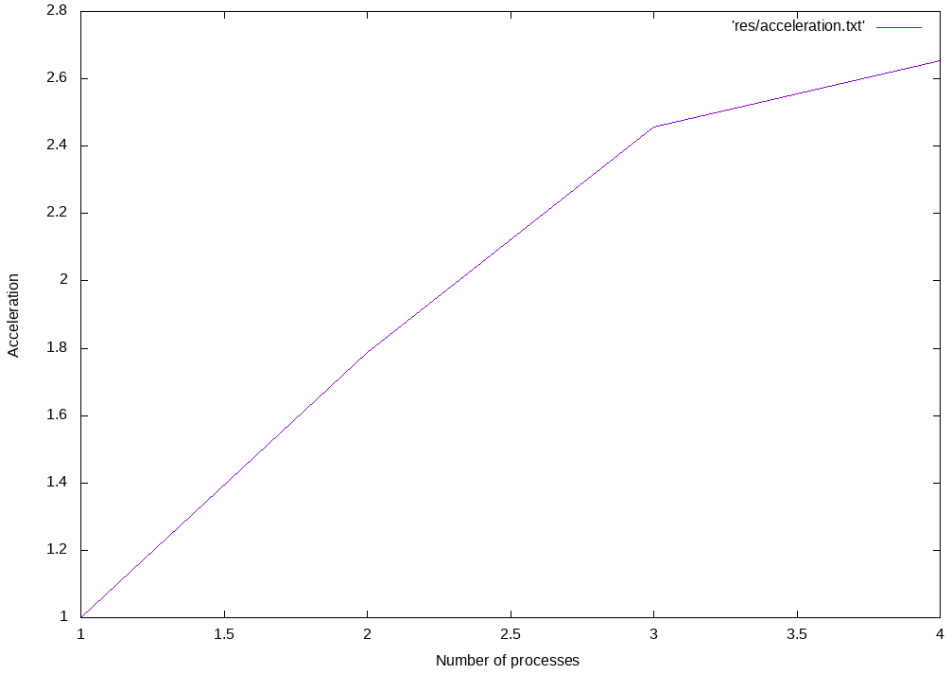


Рис. 17: Ускорение параллельного алгоритма

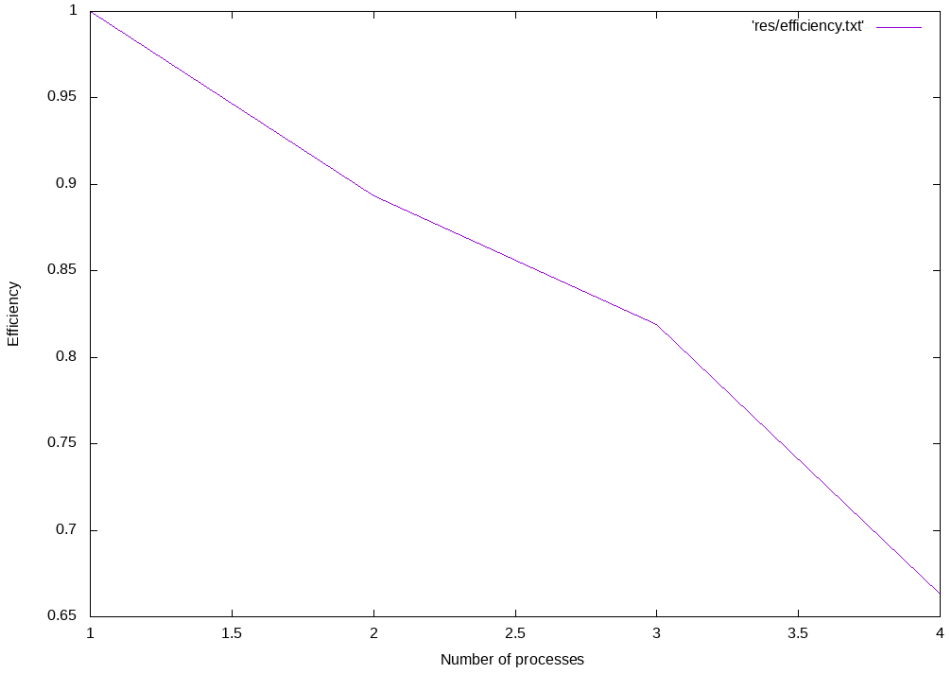


Рис. 18: Эффективность параллельного алгоритма

Выводы

В данной работе мы продемонстрировали алгоритм рассчета уравнения теплопроводности на распределенных системах двумя способами: используя MPI и OMP. Обе алгоритма показали результат, совпадающий с теоретическим, а также продемонстрировали качественное улучшения времени работы на распределенных системах.

Список использованной литературы

1. Петровский И. Г. Лекции об уравнениях с частными производными. — гл. IV, § 40. — Любое издание.

2. Тихонов А. Н., Самарский А. А. Уравнения математической физики. — гл. III. — Любое издание.

3. Разностная аппроксимация начально-краевой задачи для уравнения теплопроводности. Понятие явной и неявной схемы.

4. Уравнение теплопроводности на распределенных системах

5. Уравнения математической физики. Задачи и решения

6. РАЗНОСТНЫЕ МЕТОДЫ РЕШЕНИЯ ЗАДАЧ ТЕПЛОПРОВОДНОСТИ

7. ЧИСЛЕННЫЕ МЕТОДЫ РЕШЕНИЯ ЗАДАЧ ТЕПЛО- И МАССОПЕРЕНОСА

8. Уравнение теплопроводности

9. Краевая задача