Санкт-Петербургский политехнический университет Петра Великого
Институт прикладной математики и механики
Высшая школа прикладной математики и вычислительной физики

Параллельные вычислени

Лабораторная работа №1
«Решение задачи Коши для системы линейных неоднородных ОДУ методом Пикара»

Работу выполнила:
студентка II курса
магистратуры
Добрецова Е.В.
Группа:
5040102/40101
Преподаватель:
Козлов К.Н.

Санкт-Петербург
2026

Содержание

1. Постановка задачи и её формализация

Дана система линейных неоднородных обыкновенных дифференциальных уравнений с постоянными коэффициентами:

$$\frac{d\boldsymbol{y}}{dt} = A\boldsymbol{y} + \boldsymbol{f}(t), \tag{1}$$

и следующими начальными условиями:

$$y_i = 0, \quad i = \overline{1, n}. \tag{2}$$

Здесь $\boldsymbol{y}(t) = \begin{pmatrix} y_1(t) \\ y_2(t) \\ \dots \\ y_n(t) \end{pmatrix}$, $\boldsymbol{f}(t) = \begin{pmatrix} f_1(t) \\ f_2(t) \\ \dots \\ f_n(t) \end{pmatrix}$, матрица A имеет размерность $n \times n$ и является нижнетреугольной ($a_{ij} = 0$ при $i > j$) с единичными ненулевыми элементами.

Необходимо решить задачу методом последовательных приближений Пикара с применением циклической схемы распределения уравнений по процессам для обеспечения равномерной загрузки процессоров, осуществляя программирование на языке C с применением технологии OpenMP. Также требуется провести анализ метода путём следующих исследований:

- зависимость времени выполнения от размера системы;
- зависимость времени выполнения от числа параллельных процессов/потоков;
- зависимость ускорения от числа параллельных процессов/потоков;
- зависимость эффективности параллелизации от числа параллельных процессов/-потоков.

2. Алгоритм метода Пикара и условия его применимости

2.1. Описание метода

Метод Пикара для задачи (1) с начальными условиями (2) определяет последовательность приближений $\boldsymbol{y}^{(k)}(t)$ следующим образом:

$$\boldsymbol{y}^{(k+1)}(t) = \boldsymbol{y}_0 + \int_{t_0}^{t} \left(A\boldsymbol{y}^{(k)}(\tau) + \boldsymbol{f}(\tau)\right) d\tau, \quad k = 0, 1, 2, \dots$$

Процесс продолжается до выполнения критерия сходимости:

$$\left\|\boldsymbol{y}^{(k+1)} - \boldsymbol{y}^{(k)}\right\| < \varepsilon$$

Интеграл вычисляется численно по квадратурной формуле трапеций на каждом временном интервале $[t_m, t_{m+1}]$:

$$\boldsymbol{y}_{m+1}^{(k+1)} = \boldsymbol{y}_m^{(k+1)} + \frac{\Delta t}{2}\left[A\boldsymbol{y}_m^{(k)} + \boldsymbol{f}(t_m) + A\boldsymbol{y}_{m+1}^{(k)} + \boldsymbol{f}(t_{m+1})\right], \tag{3}$$

где $\Delta t = t_{m+1} - t_m$, $\boldsymbol{y}_m^{(k)}$ - приближение в узле t_m на k-й итерации.

2.2. Параллелизация вычислений

На каждой итерации вычисления правых частей для разных компонент системы независимы, так как они используют значения предыдущего приближения $\boldsymbol{y}^{(k)}$, которые на этом этапе полностью известны. Поэтому в данной задаче применяется декомпозиция по данным (parallelism across space). При таком способе работы каждая компонентная функция $y_i(t)$ считается независимо, уравнения равномерно распределяются между потоками OpenMP. Используется циклическая схема распределения.
Каждый поток независимо выполняет:

1. вычисление своих компонент
$$A\boldsymbol{y}^{(k)} + \boldsymbol{f}(t),$$
2. применение квадратурной формулы трапеций для этих компонент.

Синхронизация необходима только между итерациями k и $k+1$, когда все потоки должны завершить обновление своих компонент.

2.3. Условия применимости метода

Итерационный процесс Пикара сходится при выполнении следующих условий:

1. липшицевость правой части по переменной y;
2. ограниченность интервала интегрирования;
3. корректность задания начальных условий.

2.4. Стандарт OpenMP

OpenMP (от англ. Open Multi-Processing) - открытый стандарт для распараллеливания программ на языках C, C++ и Fortran, дающий описание совокупности директив компилятора, библиотечных процедур и переменных окружения, предназначенных для программирования многопоточных приложений на многопроцессорных системах с общей памятью.

3. Предварительный анализ задачи (проверка условий применимости метода)

Проверим условие Липшица по y для правой части в рассматриваемой области.

$F(t, y) = Ay + f(t)$ линейна по y, поэтому удовлетворяет усл. Липшица в любой ограниченной области:

$$\|F(t, y_1) - F(t, y_2)\| = \|A(y_1 - y_2)\| \leq L \|y_1 - y_2\|,$$

$L = \|A\|$

Для обеспечения сходимости метода Пикара отрезок выбирается из условия сжимаемости оператора. Поскольку правая часть удовлетворяет условию Липшица с константой $L = \|A\|$, а левая граница принята за 0, правая граница должна быть выбрана с учётом ограничения на длину отрезка:

$$t \leq \frac{1}{\|A\|}.$$

Поэтому будем изучать метод Пикара на отрезке $[0, \frac{1}{n+1}]$, где n – размерность системы.

4. Тестовый пример с детальными расчётами для задачи малой размерности

Рассмотрим первые две итерации метода Пикара для задачи размерности 3.

Дана система из трёх линейных неодн. ОДУ:

$$\frac{dy}{dt} = Ay + f(t),$$

$$A = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{pmatrix}, \quad f(t) = \begin{pmatrix} t \\ t^2 \\ 3\sin t \end{pmatrix}, \quad 0 \le t \le 0{,}3, \quad \Delta t = 0{,}1$$

Сетка: $t_m = \frac{m}{10}$, $m = \overline{0,3}$

Начальные условия: $y_m^{(0)} = 0$, $m = \overline{0,3}$

Дальнейшие значения $y_{m+1}^{(k+1)}$ находятся по формуле (3).

Пусть доступно $p = 3$ потока OpenMP. Тогда поток с номером $m-1$, $m = \overline{1,3}$ будет считать y_m. Все потоки используют значения $y^{(k)}$, полученные на предыдущей итерации, поэтому вычисления полностью независимы.

Поток 0: вычисляет $\frac{dy_1}{dt} = y_1 + t$

Поток 1: вычисляет $\frac{dy_2}{dt} = y_1 + y_2 + t^2$

Поток 2: вычисляет $\frac{dy_3}{dt} = y_1 + y_2 + y_3 + 3\sin t$

После данных вычислений компоненты собираются в общий вектор, затем переходим к следующей итерации.

Итерация 1

$y^{(0)} \equiv 0 \Rightarrow A y_m^{(0)} + f(t_m) = f(t_m)$

$f(t_0) = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$, $f(t_1) = \begin{pmatrix} 0,1 \\ 0,01 \\ 0,099 \end{pmatrix}$, $f(t_2) = \begin{pmatrix} 0,2 \\ 0,04 \\ 0,199 \end{pmatrix}$,

$f(t_3) = \begin{pmatrix} 0,3 \\ 0,09 \\ 0,296 \end{pmatrix}$

Узел 0:

$y_0^{(1)} = 0$

Узел 1:

$y_1^{(1)} = 0 + 0,05 \cdot (f(t_1) + f(t_0)) = 0,05 \cdot f(0,1) = \begin{pmatrix} 0,005 \\ 0,0005 \\ 0,00499 \end{pmatrix}$

Узел 2:

$y_2^{(1)} = y_1^{(1)} + 0,05 (f(t_2) + f(t_1)) = \begin{pmatrix} 0,005 \\ 0,0005 \\ 0,00499 \end{pmatrix} +$

$+ 0,05 \begin{pmatrix} 0,3 \\ 0,05 \\ 0,2985 \end{pmatrix} = \begin{pmatrix} 0,02 \\ 0,003 \\ 0,0199 \end{pmatrix}$

Узел 3:

$y_3^{(1)} = y_2^{(1)} + 0,05 (f(t_3) + f(t_2)) =$

$= \begin{pmatrix} 0,02 \\ 0,003 \\ 0,0199 \end{pmatrix} + 0,05 \begin{pmatrix} 0,5 \\ 0,13 \\ 0,495 \end{pmatrix} = \begin{pmatrix} 0,045 \\ 0,0095 \\ 0,0447 \end{pmatrix}$

$y_0^{(1)} = (0,0,0)^T$

$y_1^{(1)} = (0,005; 0,0005; 0,00499)^T$

$y_2^{(1)} = (0,02; 0,003; 0,0199)^T$

$y_3^{(1)} = (0,045; 0,0095; 0,0447)^T$

Итерация 2

Узел 0:

$y_0^{(2)} = 0$

Узел 1:

$$A y_1^{(1)} = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} 0{,}005 \\ 0{,}0005 \\ 0{,}00499 \end{pmatrix} = \begin{pmatrix} 0{,}005 \\ 0{,}0055 \\ 0{,}0105 \end{pmatrix}$$

$$y_1^{(2)} = y_0^{(2)} + 0{,}05\left(A y_0^{(1)} + f(t_0) + A y_1^{(1)} + f(t_1)\right) =$$

($A y_0^{(1)} \to 0$, $f(t_0) \to 0$)

$$= 0{,}05\left(\begin{pmatrix} 0{,}005 \\ 0{,}0055 \\ 0{,}0105 \end{pmatrix} + \begin{pmatrix} 0{,}1 \\ 0{,}01 \\ 0{,}099 \end{pmatrix}\right) = \begin{pmatrix} 0{,}00525 \\ 0{,}000775 \\ 0{,}00548 \end{pmatrix}$$

Узел 2:

$$A y_2^{(1)} = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} 0{,}02 \\ 0{,}003 \\ 0{,}0199 \end{pmatrix} = \begin{pmatrix} 0{,}02 \\ 0{,}023 \\ 0{,}0429 \end{pmatrix}$$

$$y_2^{(2)} = y_1^{(2)} + 0{,}05\left(A y_1^{(1)} + f(t_1) + A y_2^{(1)} + f(t_2)\right) =$$

$$= \begin{pmatrix} 0{,}00525 \\ 0{,}000775 \\ 0{,}00548 \end{pmatrix} + 0{,}05\left(\left(\begin{pmatrix} 0{,}005 \\ 0{,}0055 \\ 0{,}0105 \end{pmatrix} + \begin{pmatrix} 0{,}1 \\ 0{,}01 \\ 0{,}099 \end{pmatrix}\right) + \right.$$

$$\left. + \begin{pmatrix} 0{,}02 \\ 0{,}023 \\ 0{,}0429 \end{pmatrix} + \begin{pmatrix} 0{,}2 \\ 0{,}04 \\ 0{,}199 \end{pmatrix}\right) = \begin{pmatrix} 0{,}0215 \\ 0{,}0047 \\ 0{,}0231 \end{pmatrix}$$

Узел 3:

$$A y_3^{(1)} = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} 0{,}045 \\ 0{,}0095 \\ 0{,}0447 \end{pmatrix} = \begin{pmatrix} 0{,}045 \\ 0{,}0545 \\ 0{,}0992 \end{pmatrix}$$

$$y_3^{(2)} = y_2^{(2)} + 0{,}05\left(A y_2^{(1)} + f(t_2) + A y_3^{(1)} + f(t_3)\right) =$$

$$= \begin{pmatrix} 0{,}0215 \\ 0{,}0047 \\ 0{,}0231 \end{pmatrix} + 0{,}05\left(\left(\begin{pmatrix} 0{,}02 \\ 0{,}023 \\ 0{,}0429 \end{pmatrix} + \begin{pmatrix} 0{,}2 \\ 0{,}04 \\ 0{,}199 \end{pmatrix} + \right.\right.$$

$$\left.\left. + \begin{pmatrix} 0{,}045 \\ 0{,}0545 \\ 0{,}0992 \end{pmatrix} + \begin{pmatrix} 0{,}3 \\ 0{,}09 \\ 0{,}296 \end{pmatrix}\right) = \begin{pmatrix} 0{,}04975 \\ 0{,}015075 \\ 0{,}05496 \end{pmatrix}$$

$y_0^{(2)} = (0, 0, 0)^T$

$y_1^{(2)} = (0{,}00525;\ 0{,}000775;\ 0{,}00548)^T$

$y_2^{(2)} = (0{,}0215;\ 0{,}0047;\ 0{,}0231)^T$

$y_3^{(2)} = (0{,}04975;\ 0{,}015075;\ 0{,}05496)^T$

5. Подготовка контрольных тестов для иллюстрации метода

Во всех исследованиях правая часть СЛАУ имеет вид $\boldsymbol{f}(t) = sin(t + i)$, $i = \overline{(0, n-1)}$.

Для исследования зависимости времени выполнения программы от размера системы зафиксируем количество потоков $p = 4$, а также количество узлов $m = 1000$. Будем поочерёдно запускать метод на системах размерности от 1 до 30.

В исследованиях времени выполнения, ускорения и эффективности параллелизации от количества параллельных процессов будем брать поочерёдно значения количества потоков p от 1 до 12, количество потоков ограничивается количеством ядер процессора, равным 6. Зафиксируем размерность системы $n = 1000$ и количество узлов $m = 3500$, данные значения обеспечивают достаточную вычислительную нагрузку для проявления эффекта параллелизации.

Ускорение находится по формуле $a = \frac{T_1}{T_p}$, где T_1 - время выполнения программы без параллелизации, T_p - время выполнения при числе потоков p. Эффективность параллелизации вычисляется по формуле $E = \frac{T_1}{p \cdot T_p}$.

6. Модульная структура программы

```
picard├──
 analysis│
   └── plots.ipynb├──
 results└──
 src├──
     CMakeLists.txt├──
     generate.c├──
     generate.h├──
     main.c├──
     matrix_tools.c├──
     matrix_tools.h├──
     picard.c├──
     picard.h├──
     study.c└──
     study.h
```

Приведём на листинге дерево проекта.
В файле `analysis/plots.ipynb` содержится код чтения данных из `.csv` файлов и построение графиков средствами `matplotlib.pyplot`.
Опишем файлы из директории `src`, где содержится `C`-код метода.

- `matrix_tools.c`
  - `double **alloc_matrix(int n, int m)` - выделение памяти для матрицы размера $n \times m$;
  - `void free_matrix(double **y, int n)` - освобождение памяти, занятой массивом из n массивов;
  - `double compute_diff_norm(int n, int m, double **y1, double **y2)` - вычисление строковой максимум-нормы разности матриц;
  - `void matrix_mul_vector(int n, double** A, double* y_m, double* res)` - умножение матрицы на вектор.
- `picard.c`
  - `double **picard_method(int n, double **f, double**A, double eps, double t_0, double t, int m)` - функция метода Пикара для задачи размерности `n` с `m` узлами, матрицей `A` и вектором правой части `f` на отрезке $[t_0, t]$ с точностью `eps`.
- `generate.c`
  - `double **generate_test_rhs(int n, int m, double t_0, double t)` - генерация правой части для тестового примера;
  - `double **generate_rhs(int n, int m, double t_0, double t)` - генерация правой части для контрольного примера;
  - `int generate_threads(int *threads, int max_threads)` - генерация последовательности чисел от 1 до `max_threads` и её сохранение;
  - `double **generate_matrix(int n)` - генерация матрицы для контрольного примера согласно постановке задачи.
- `study.c`
  - `void study_time_vs_size(void)` - изучение зависимости времени выполнения программы от разных значений размерности системы. Функция перебирает значения размерности от 1 до 30 при фиксированном количестве потоков и записывает время выполнения программы для каждого значения размерности в `.csv`-файл для построения графика;
  - `void study_dependencies_on_threads(void)` - изучение зависимости времени выполнения программы, ускорения и эффективности параллелизации от числа параллельных процессов. Функция фиксирует размерность системы и число узлов и решает задачу методом Пикара для каждого значения числа параллельных процессов, вычисляя все перечисленные выше величины и записывая их в `.csv`-файл.

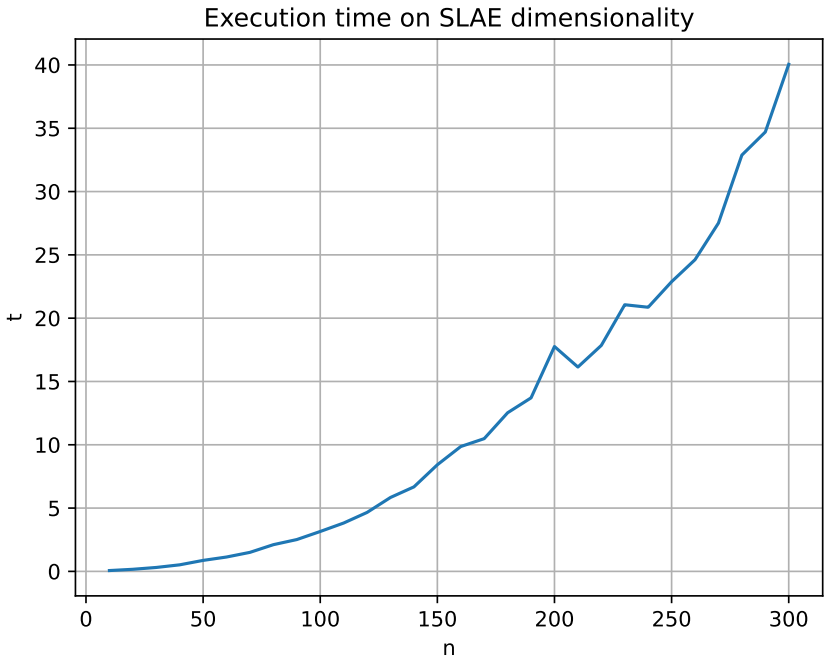

Рис. 7.1. Зависимость времени исполнения программы от размерности системы при 4 потоках

- `main.c`
  - `void run_test_case(void)` - функция исполнения тестового примера для проверки корректности реализации метода;
  - `int main(void)` - запуск функций из `study.c`.

7. Численный анализ решения задачи

Значения фиксированных переменных и диапазоны меняющихся для описанных ниже исследований приведены выше.

На Рис. 7.1 показан график зависимости времени выполнения программы от размерности системы. Из графика видно, что время выполнения программы с ростом размерности системы возрастает квадратично, что обусловлено квадратичной вычислительной сложностью операций умножения матрицы на вектор на каждом шаге метода Пикара. Параллельная реализация с использованием четырёх потоков позволяет сократить абсолютное время выполнения программы, но не меняет характер зависимости. Небольшие неровности на графике связаны с особенностями работы с памятью и накладными расходами параллельных вычислений.

График зависимости времени выполнения программы от числа параллельных процессов показан на Рис. 7.2.

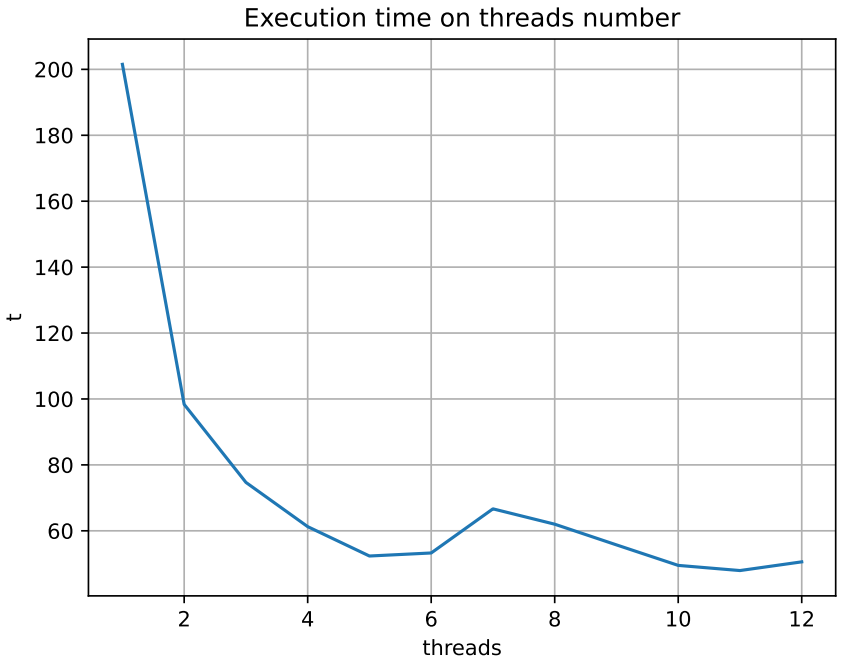

Рис. 7.2. Зависимость времени исполнения программы от числа параллельных процессов

Согласно графику, при увеличении числа потоков от 1 до 4-5 наблюдается существенное сокращение времени исполнения программы. При дальнейшем увеличении числа потоков снижение времени замедляется и, более того, возможно нго увеличение. Это связано с ростом накладных расходов на синхронизацию потоков и работу с памятью.

На Рис. 7.3 представлен график ускорения в зависимости от числа параллельных процессов/потоков. Ускорение растёт практически линейно при числе потоков до 4-5, максимальное ускорение составляет около 4.1-4.2 и достигается при 11 потоках. Нелинейный характер роста ускорения и наличие локальных провалов обусловлены накладными расходами OpenMP и конкуренцией потоков за доступ к памяти.

График эффективности параллелизации от числа потоков приведён на Рис. 7.4. Можно видеть, что при увеличении числа потоков накладные расходы и ограничения архитектуры начинают доминировать, в связи с этим жффективность параллелизации с ростом числа потоков снижается. Однако при малом числе потоков наблюдается небольшой рост эффективности.

8. Выводы

В ходе выполнения данной работы была реализована параллельная версия метода Пикара с применением циклической схемы распределения уравнений по процессам с использованием технологии OpenMP, а также проведён ряд вычислительных экспериментов.

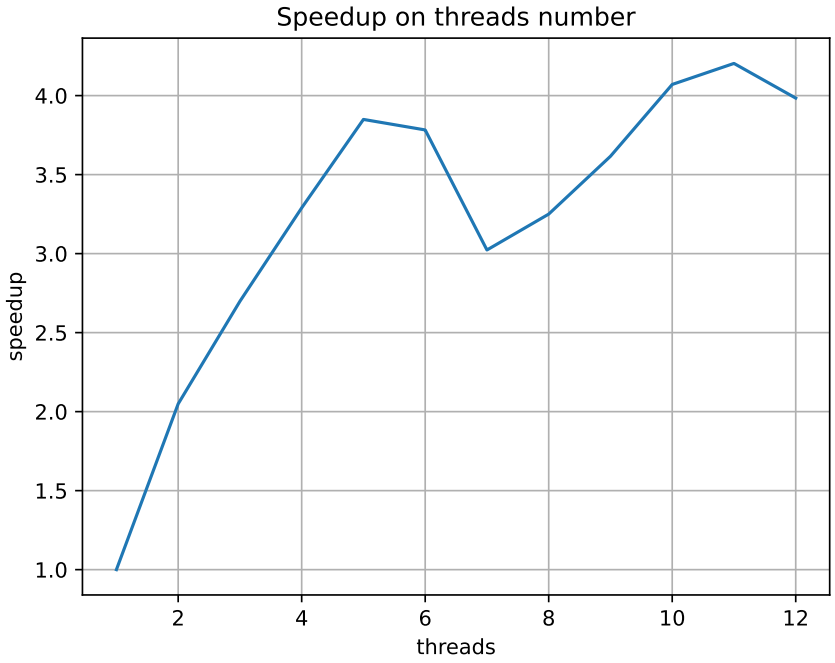

Рис. 7.3. Зависимость ускорения от числа параллельных процессов

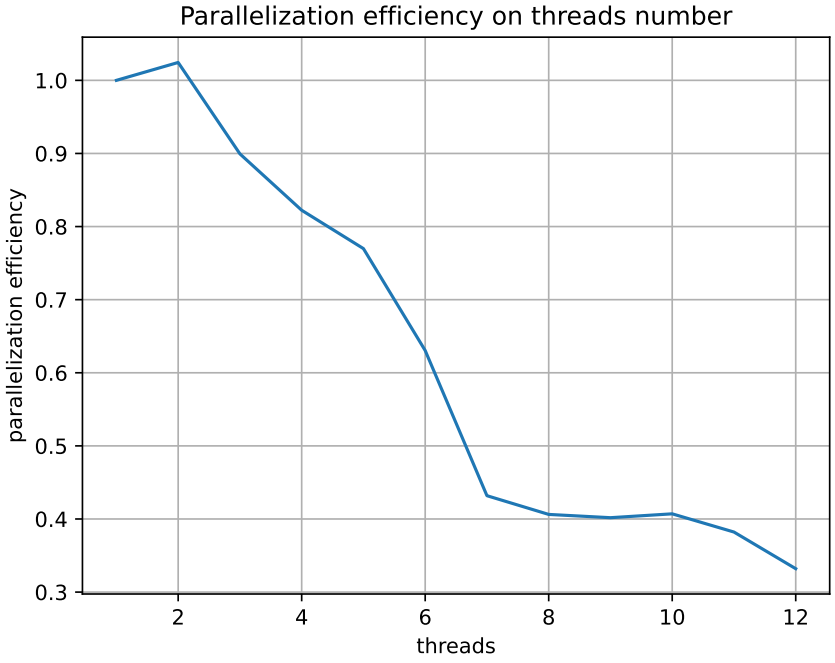

Рис. 7.4. Зависимость эффективности параллелизации от числа параллельных процессов

Исследования показали, что с ростом размерности системы время выполнения программы возрастает квадратично. Такой характер роста обусловлен квадратичной вычислительной сложностью операции умножения матрицы на вектор.

При увеличении числа потоков наблюдается существенное сокращение времени выполнения программы по сравнению с её последовательным исполнением. Наибоольший выигрыш достигается при увеличении числа потоков до значения, близкого к числу вычислительных ядер компьютера. Дальнейшее увеличение числа потоков не приводит к заметному сокращению исполняемого времени вследствие роста накладных расходов и ограничений архитектуры памяти.

Анализ ускорения показал, что при малом числе потоков наблюдается близкий к линейному рост ускорения, однако при числе потоков, большем 5, ускорение практически останавливается.

Эффективность параллелизации незначительно растёт при малом числе потоков, но при его дальнейшем увеличении эффективность снижается вследствие роста накладных расходов на синхронизацию и конкуренции за ресурсы.

Таким образом, параллельная реализация метода Пикара позволяет существенно сократить время исполнения для задач большой размерности.

9. Примечания

Код реализации метода доступен здесь.