МИНЕСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИИ БЕЛАРУСЬ
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ
Кафедра вычислительной математики

Стефанович Константин Андреевич

Построение параллельного алгоритма численного решения двумерного уравнения Пуассона на основе красно-черного упорядочивания

Курсовая работа

Научный руководитель:

доктор физ.-мат. наук,

профессор Н.А. Лиходед

Допущен к защите

«___» _____ 2018 г

Зав. кафедрой вычислительной математики, кандидат физико-математических наук, доцент В.И. Репников

Минск, 2018

РЕФЕРАТ

Работа содержит: 1 таблиц, 10 рисунков, 24 страниц, 9 сайтов, 9 формул.

Ключевые слова: ИТЕРАЦИОННЫЕ ПАРАЛЛЕЛЬНЫЕ АЛГОРИТМЫ, КРАСНО-ЧЕРНОЕ УПОРЯДОЧЕНИЕ, OPENMP.

Объект исследования – параллельный алгоритм численного решения двумерного уравнения Пуассона на основе красно-чёрного упорядочения.

Цель работы – исследование параллельного алгоритма на основе красно-чёрного упорядочения, сравнение его с параллельными алгоритмами Якоби и Зейделя.

В результате – разработан и программно реализован параллельный алгоритм на основе красно-чёрного упорядочения.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ 5

ГЛАВА 1 МАТЕМАТИЧЕСКАЯ ПОСТАНОВКА ЗАДАЧИ **6**
1.1 Аппроксимация 6
1.1.1 Итерационный процесс Якоби 7
1.1.2 Итерационный процесс Зейделя 8
1.1.3 Итерационный процесс с красно-черным упорядочением 8
1.2 Распараллеливание и Open MP 9
1.2.1 Функции исполняющей среды 11
1.2.2 Функции синхронизации/блокировки 12
1.2.3 Функции работы с таймерами 13

ГЛАВА 2 ПРАКТИЧЕСКАЯ РЕАЛИЗАЦИЯ ЗАДАЧИ **14**
2.1 Точечный параллельный алгоритм, выполняющий заданное число итераций. 14
2.1.1 Итерационный процесс Якоби 15
2.1.2 Итерационный процесс с упорядочением 15
2.2 Итерационный процесс с оценкой погрешности 19
ЗАКЛЮЧЕНИЕ 23
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 24

АННОТАЦИЯ

на курсовую работу «Построение параллельного алгоритма численного решения двумерного уравнения Пуассона на основе красно-черного упорядочивания»

В данном уравнении исследуется параллельный алгоритм численного решения двумерного уравнения Пуассона на основе красно-чёрного упорядочения. Также будет разработан и программно реализован параллельный алгоритм на основе красно-чёрного упорядочения.

ВВЕДЕНИЕ

В математике уравнение Пуассона – это эллиптическое дифференциальное уравнение в частных производных, которое широко применяется на практике (инженерное дело) и теории (теоретическая физика). В частности, оно описывает потенциальное поле, вызванное заданным распределением плотности заряда или массы. Также в физике часто используется уравнение Лапласа, которое является частным случаем уравнения Пуассона. Уравнение названо в честь знаменитого французского математика, геометра и физика Симеона Пуассона(Подробнее смотрите в 2)

Уравнение Пуассона можно представить в виде $\Delta u = f$, где Δ – оператор Лапласа, а f и φ – вещественные или комплексные функции на некотором многообразии. Обычно функция f нам дана, а функцию φ нужно найти.

В двумерных декартовых координатах уравнение принимает вид (1):

$$\frac{\partial^2 u}{\partial x_1^2} + \frac{\partial^2 u}{\partial x_2^2} = f(x_1, x_2) \tag{1}$$

Если $f = 0$, то уравнение Пуассона превращается в уравнение Лапласа: $\Delta\varphi = 0$.

Уравнение Пуассона можно решить явно с помощью функции Грина, существует также много способов численного решения, например, релаксационный метод.

ГЛАВА 1 МАТЕМАТИЧЕСКАЯ ПОСТАНОВКА ЗАДАЧИ

1.1 Аппроксимация

Рассмотрим двумерную задачу Дирихле для уравнения Пуассона (1.1):

$$\begin{cases} \frac{\partial^2 u}{\partial x_1^2} + \frac{\partial^2 u}{\partial x_2^2} = f(x_1, x_2); (x_1, x_2) \in G \\ u(x_1, x_2)|_\Gamma = \mu(x_1, x_2), \end{cases} \tag{1.1}$$

где G - прямоугольная область $[0 < x_1 < l_1] \times [0 < x_2 < l_2]$ с границей Γ.

Пусть нами задана равномерная прямоугольная сетка (1.2):

$$\{ih_1, i = 0, 1, \ldots, N_{x_1}, N_{x_1}h_1 = l_1, jh_2, j = 0, 1, \ldots, N_{x_2}, N_{x_2}h_2 = l_2\} \tag{1.2}$$

Запишем теперь разностную задачу Дирихле на данной сетке, используя пятиточечный шаблон "крест"(Формула (1.3), рисунок 1.1):

$$\begin{cases} \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{h_1^2} + \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{h_2^2} = f_{ij}, \\ u|_{\gamma_h} = \mu(x_1, x_2)|_{\gamma_h} \\ i = 0, 1, \ldots, N_{x_1}; j = 0, 1, \ldots, N_{x_2} \end{cases} \tag{1.3}$$

где γ_h - граничные узлы.

Записанную систему линейных алгебраических уравнений относительно u_{ij} обычно решают итерационными методами. Перед началом вычислений задают начальное (обычно нулевое) приближение u_{ij}^0 во внутренних точках сетки, а значения u_{ij}^0 в граничных узлах заполняют точными значениями μ_{ij}.

Рассмотрим несколько итерационных методов.

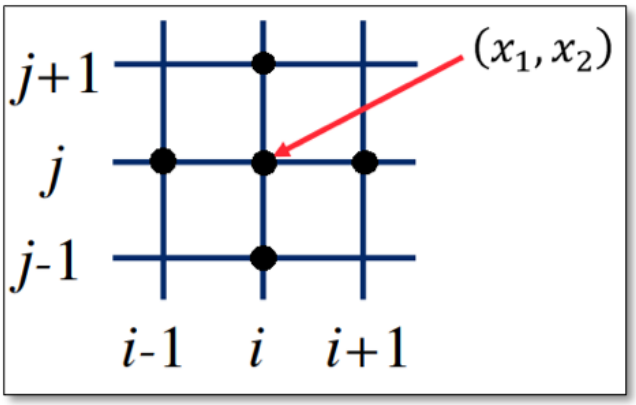

Рисунок 1.1 - Пятиточечный шаблон "крест"

1.1.1 Итерационный процесс Якоби

Итерационный процесс метода Якоби имеет следующий вид (1.4):

$$u_{i,j}^{k+1} = \left(\frac{2}{h_1^2} + \frac{2}{h_2^2}\right)^{-1} \left(\frac{u_{i+1,j}^k + u_{i-1,j}^k}{h_1^2} + \frac{u_{i,j+1}^k + u_{i,j-1}^k}{h_2^2} - f_{ij}\right) \tag{1.4}$$
$$i = 0, 1, \ldots, N_{x_1}; j = 0, 1, \ldots, N_{x_2}; k = 0, 1, \ldots$$

Если $h = h_1 = h_2$, тогда получаем (1.5):

$$u_{i,j}^{k+1} = \frac{1}{4}\left(u_{i+1,j}^k + u_{i-1,j}^k + u_{i,j+1}^k + u_{i,j-1}^k - h^2 f_{ij}\right) \tag{1.5}$$
$$i = 0, 1, \ldots, N_{x_1}; j = 0, 1, \ldots, N_{x_2}; k = 0, 1, \ldots$$

Данный метод не очень часто используется на практике, так как требует дополнительной матрицы, что увеличивает память и время работы программы, так как часть этого времени используется на копирование элементов. Подробнее смотрите в 1, 4.

1.1.2 Итерационный процесс Зейделя

Итерационный процесс метода Зейделя имеет следующий вид (1.6):

$$u_{i,j}^{k+1} = \left(\frac{2}{h_1^2} + \frac{2}{h_2^2}\right)^{-1} \left(\frac{u_{i+1,j}^{k} + u_{i-1,j}^{k+1}}{h_1^2} + \frac{u_{i,j+1}^{k} + u_{i,j-1}^{k+1}}{h_2^2} - f_{ij}\right) \quad (1.6)$$
$$i = 0, 1, \ldots, N_{x_1}; j = 0, 1, \ldots, N_{x_2}; k = 0, 1, \ldots$$

Если $h = h_1 = h_2$, тогда получаем (1.7):

$$u_{i,j}^{k+1} = \frac{1}{4}\left(u_{i+1,j}^{k} + u_{i-1,j}^{k+1} + u_{i,j+1}^{k} + u_{i,j-1}^{k+1} - h^2 f_{ij}\right) \quad (1.7)$$
$$i = 0, 1, \ldots, N_{x_1}; j = 0, 1, \ldots, N_{x_2}; k = 0, 1, \ldots$$

Алгоритм Зейделя имеет то преимущество, что в компьютерной реализации нам больше не нужно выделять два массива. Вместо этого мы можем сделать только один массив и выполнять все обновления значений только в нём. Ещё метод Зейделя дает более быстрое уменьшение ошибки, чем метод Якоби. Подробнее смотрите в 3.

1.1.3 Итерационный процесс с красно-черным упорядочением

Рассмотрим способ обхода узлов сетки, который основывается на их «красно-черном» упорядочивании (Рисунок 1.2). В соответствии с этим приемом все узлы сетки делятся на два подмножества Ω_1 и Ω_2 (сеточные узлы «раскрашиваются» в черные и красные цвета в шахматном порядке), а поскольку при получении разностных схем использовался шаблон пятиточечный «крест», для расчета значения сеточной функции в узле черного цвета нужны значения в узлах красного цвета и наоборот. В этом методе узлы в анализируемом сетке делятся на 2 блока в шахматном порядке. Во время обхода алгоритма все красные точки обновляются до черных точек. Для двумерной задачи Пуассона мы видим, что для обновления красной точки сетки требуется только информация из черных точек сетки и наоборот. Следовательно, порядок, в котором обновляются точки в каждом наборе, не имеет значения. Поскольку

блоки разного цвета являются независимыми друг от друга, итерации можно распараллелить для каждого цвета. Аналитическое исследование показывает, что скорость сходимости улучшается за счет увеличения числа узлов одного блока и легко устанавливается оптимальный размер блока для получения наилучшей скорости сходимости. Численные тесты показывают, что метод достигает большого ускорения из-за быстрой конвергенции, небольших затрат синхронизации и эффективного использования кэша данных на скалярном параллельном компьютере.

Подробнее смотрите в 5, 6.

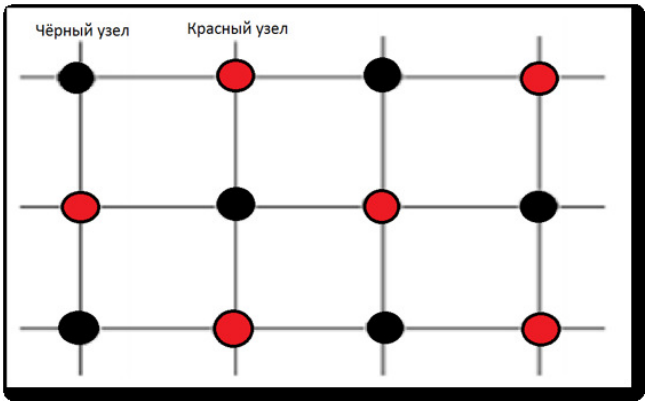

Рисунок 1.2 - ″Красно-черное″ упорядочивание узлов сетки(векторное изображение)

1.2 Распараллеливание и Open MP

Существует множество различных технологий параллельного программирования. Причем эти технологии различаются не столько языками программирования, сколько архитектурными подходами построения параллельных систем.

Определенные технологии, например, предполагают построение параллельных алгоритмов на одной машине с несколькими процессорными ядрами, другие же – нескольких компьютеров (одного или нескольких типов).

Системы параллельного программирования для работы на одной машине, начали развиваться не так давно. Не стоит думать, что это принципиально новые идеи, но именно с приходом многоядерных систем на рабочий стол, программистам стоит обратить свое внимание на такие технологии как Open MP, Intel Thread Building Blocks, Microsoft Parallel Extensions и ряд других.

Очень важно, чтобы технология параллельного программирования поддерживала возможность делать введение параллельности в программу постепенно. Идеальную параллельную программу стоит с самого начала писать параллельной, предпочтительнее, используя какой-либо функциональный язык, где не стоит вопрос распараллеливания как таковой. Однако в реальном мире, где вместо функционального *F#* есть 10 Мбайт кода на C++, и такой код надо постепенно распараллеливать. В таком случае технология Open MP (например) является довольно удачным решением, т.к. она позволяет, выделив в приложении приоритетно нуждающиеся в параллелизации места, начать распараллеливание именно их. Как это выглядит на практике – с помощью какого-то инструмента профилирования программист находит в программе «узкие места», которые замедляют работу программы, затем эти программист распараллеливает их с помощью Open MP, после чего, можно искать следующие узкие места и так, до тех пор, пока не будет получена желаемая производительность. Именно такой подход сделал технологию Open MP довольно популярной.

Open MP (Open Multi-Processing) – это набор директив компилятора, переменных окружения и библиотечных процедур, предназначенных для программирования многопоточных программ на многопроцессорных системах с общей памятью (SMP-системах). Подробнее смотрите в 8, 7

Первый стандарт Open MP был разработан в 1997 г. как ориентированный на написание легко переносимых многопоточных приложений API. Сначала он был основан на Fortran, но позднее включил в себя C и C++.

Интерфейс Open MP стал крайне популярной технологией параллельного программирования и успешно используется как для программирования суперкомпьютерных систем с множеством процессоров, так и в настольных пользовательских системах.

Над спецификацией Open MP ведет работу множество крупных производителей вычислительной техники и ПО, чья работа регулируется некоммерческой организацией «Open MP Architecture Review Board» (ARB).

В Open MP используется модель параллельного выполнения «ветвление-слияние» (Рисунок 1.3) – программа начинается как единый поток выполнения, называемый начальным. При встрече потоком параллельной конструкции, происходит создание новой группы потоков, состоящей из начального и некоторого числа дополнительных потоков, где начальный поток остается главным. Все члены новообразованной группы (включая главный) выполняют код внутри параллельной области. В конце

параллельной области есть неявный барьер, после которого выполнение пользовательского кода продолжает только главный поток. В параллельную конструкцию могут быть вложены и другие параллельные конструкции, в которых каждый поток изначального региона становится основным для своей новой группы потоков. Вложенные регионы могут в свою очередь включать регионы еще более глубокого уровня вложенности. Число

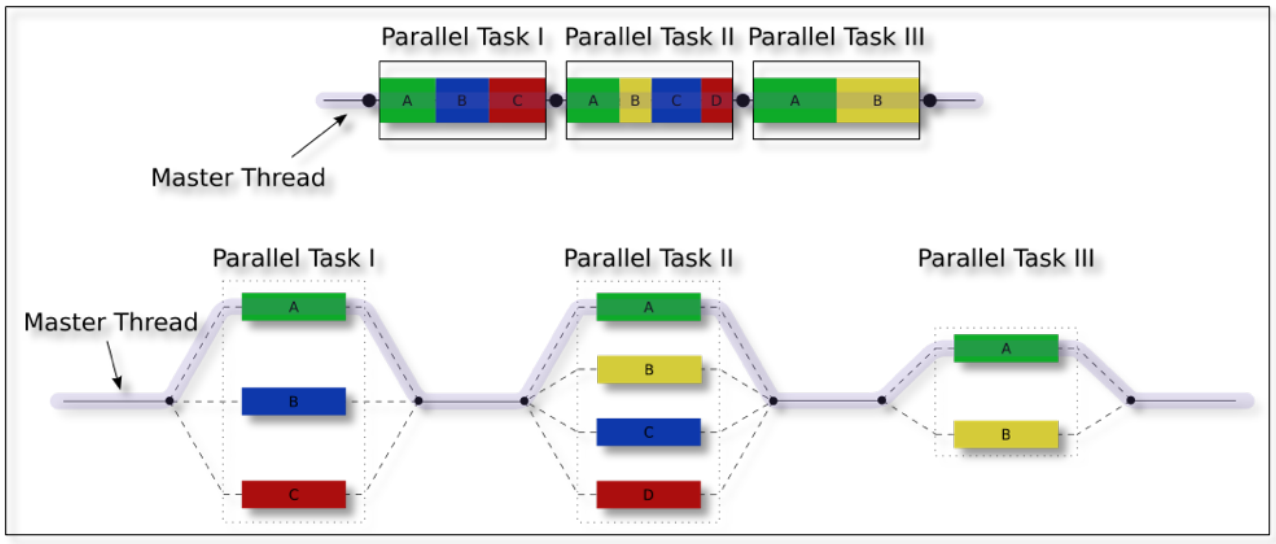

Рисунок 1.3 - Схематическое изображение модели параллельного исполнения Open MP

потоков в группе, выполняющихся параллельно, можно контролировать несколькими способами. Один из них – использование переменной окружения OMP_NUM_THREADS. Другой способ – вызов процедуры omp_set_num_threads(). Еще один способ – использование выражения num_threads в сочетании с директивой parallel.

В Open MP существует ряд вспомогательных функций. Для их использования не забудьте подключить заголовочный файл <omp.h>.

1.2.1 Функции исполняющей среды

Эти функции позволяют запрашивать и задавать различные параметры среды Open MP:

- omp_get_num_procs – возвращает число вычислительных узлов (процессоров/ядер) в компьютере;
- omp_in_parallel – позволяет потоку узнать, занимается ли он в данный момент выполнением параллельного региона;

- omp_get_num_threads – возвращает число потоков, входящих в текущую группу потоков;
- omp_set_num_thread – задает число потоков для выполнения следующего параллельного региона, который встретится текущему выполняемому потоку. Функция может помочь распределить ресурсы. Например, если мы одновременно обрабатываем звук и видео на процессоре с четырьмя ядрами, то можно создать для звука один поток, а для обработки видео –три;
- omp_get_max_threads – возвращает максимально допустимое число нитей для использования в следующей параллельной области;
- omp_set_nested – разрешает или запрещает вложенный параллелизм. Если вложенный параллелизм разрешён, то каждая нить, в которой встретится описание параллельной области, породит для её выполнения новую группу нитей и станет в ней главной;
- omp_get_nested – возвращает, разрешен ли вложенный параллелизм или нет.

Если имя функции начинается с omp_set_, то ее можно вызывать только вне параллельных регионов. Все остальные функции можно использовать как внутри параллельных регионов, так и вне таковых.

1.2.2 Функции синхронизации/блокировки

Open MP позволяет строить параллельный код без использования этих функций, так как имеются директивы, позволяющие осуществлять определенные виды синхронизации. Однако в ряде случаев эти функции удобны и даже необходимы.

В Open MP два типа блокировок: простые и вложенные. Вложенные имеют суффикс «nest». Блокировки могут находиться в одном из трех состояний – неинициализированном, заблокированном и разблокированном.

- omp_init_lock/omp_init_nest_lock – инициализация переменной типа omp_lock_t/omp_nest_lock_t. Аналог InitializeCriticalSection.
- omp_destroy_lock/omp_destroy_nest_lock – освобождение переменной типа omp_lock_t/omp_nest_lock_t. Аналог DeleteCriticalSection.

- omp_set_lock/omp_set_nest_lock – один поток выставляет блокировку, а остальные потоки ждут, пока поток, вызвавшая эту функцию, не снимет блокировку с помощью функции omp_unset_lock(). Аналог EnterCriticalSection.

- omp_unset_lock/omp_unset_nest_lock – снятие блокировки. Аналог LeaveCriticalSection.

- omp_test_lock/omp_test_nest_lock – неблокирующая попытка захвата замка. Данная функция пробует захватить указанный замок. Если это удалось, то для простого замка функция возвращает 1. Если замок захватить не удалось, то возвращается 0. Аналог TryEnterCriticalSection.

Простые блокировки (omp_lock_t) не могут быть установлены более одного раза, даже тем же потоком. Вкладываемые блокировки (omp_nest_lock_t) идентичны простым с тем исключением, что когда поток пытается установить уже принадлежащую ему вкладываемую блокировку, он не блокируется.

1.2.3 Функции работы с таймерами

- omp_get_wtime – возвращает в вызвавшем потоке астрономическое время в секундах (вещественное число двойной точности - double), прошедшее с некоторого момента в прошлом. Если некоторый участок программы окружить вызовами данной функции, то разность возвращаемых значений покажет время работы данного участка.

- omp_get_wtick () – возвращает в вызвавшем потоке разрешающую способность таймера в секундах, то есть точность таймера

- if (условие) – выполнение параллельной области по условию. Создание нескольких потоков осуществляется только при выполнении некоторого условия. Если условие не выполнено, то код выполняется в последовательном режиме.

- num_threads – явное задание количества потоков, которые будут выполнять параллельную область. По умолчанию выбирается последнее значение, установленное с помощью функции omp_set_num_threads().

ГЛАВА 2 ПРАКТИЧЕСКАЯ РЕАЛИЗАЦИЯ ЗАДАЧИ

2.1 Точечный параллельный алгоритм, выполняющий заданное число итераций.

Будем находить решение следующей задачи (2.1):

$$\begin{cases} \frac{\partial^2 u}{\partial x_1^2} + \frac{\partial^2 u}{\partial x_2^2} = 2e^{x_1+x_2}; (x_1, x_2) \in [0 < x_1 < 1] \times [0 < x_2 < 1] \\ u(0, x_2) = e^{x_2}; u(1, x_2) = e^{x_2+1}; u(x_1, 0) = e^{x_1}; u(x_1, 1) = e^{x_1+1} \end{cases} \tag{2.1}$$

Заметим, что точным решением данной задачи будет являться функция $u(x_1, x_2) = e^{x_1+x_2}$, график функции изображен на (Рисунок 2.1)

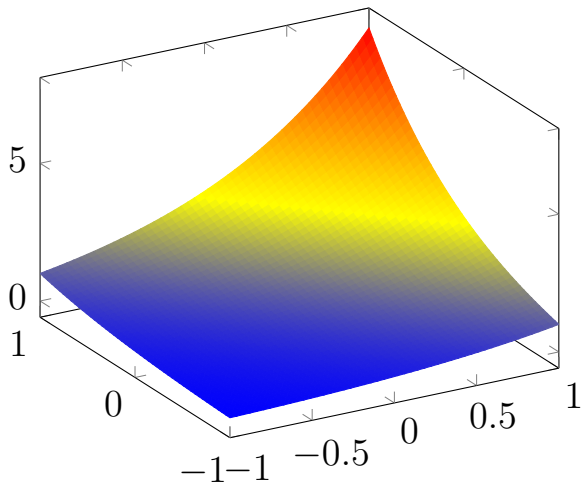

Рисунок 2.1 - График точного решения

Вычислительные эксперименты проводились на компьютере с 4-мя ядрами.

Для всех вычислений в этом разделе возьмём шаг $h = 0.025$ и три разных значения количества итераций - 10 и 100,1000.

Для удобства на каждом графике изобразим как точное, так и приближенное решение.

Точность решения на определённой итерации можно оценивать разными способами, мы будем использовать следующую формулу: $\delta = max|u_{i,j}^{n+1} - u_{i,j}^{n}|$

2.1.1 Итерационный процесс Якоби

Сначала посмотрим на поведение приближенного решения при 100 итерациях:

Погрешность, подсчитанная по приведённой выше формуле, составляет 0.0195057.

Теперь проведём 1000 итераций (Рисунок 2.2):

На этот раз видим, что приближенное решение почти совпало с точным, а, следовательно, может использоваться на практике. Погрешность составляет уже около 0.00065 $6.5 * 10^{(-4)}$). Опытным путем также можно показать, что при увеличении количества итераций до 5000 погрешность уменьшается до $2.8 * 10^{(-9)}$

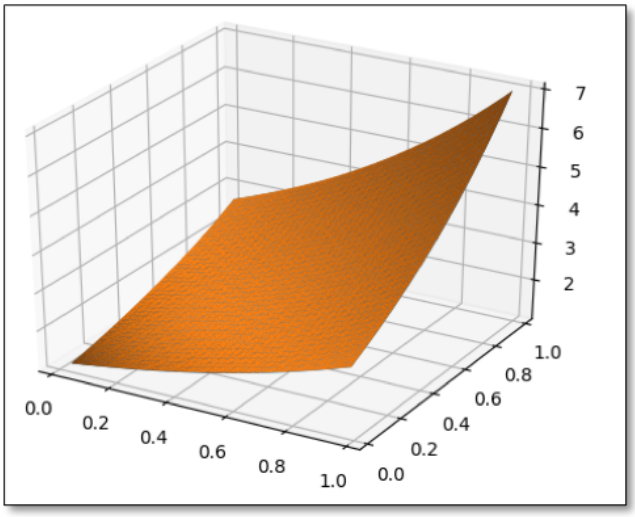

Рисунок 2.2 - График для метода Якоби, тысяча итераций

2.1.2 Итерационный процесс с упорядочением

В данном разделе мы рассмотрим алгоритм красно-черного упорядочения на 1, 2, 3, 4 и 8 потоках, также будем менять количество итераций и шаг сетки. Попробуем найти оптимальное количество потоков для решения этой задачи. Количество итераций алгоритма выбиралось так, чтобы погрешность примерно равнялась нулю. Погрешность также будет приведена в результатах работы алгоритма (Таблица 2.1, Рисунок 2.3, Рисунок 2.4).

Формула для подсчёта погрешности: $\delta = max|u_{i,j}^{n+1} - u_{i,j}^{n}|$

Каждый вычислительный эксперимент был проведён в количестве 5 раз, в графиках, диаграммах и таблицах приведены средние значения.

Таблица 2.1 - $h = 0.005$, количество итераций $= 10.000$

Номер эксперимента	Затраченное время (в секундах)	Вычислительная погрешность
1	29.190	0.000001
2	28.520	0.000001
3	28.420	0.000001
4	28.430	0.000001
5	28.470	0.000001
Итог	28.606	0.000001
1	14.700	0.000001
2	14.350	0.000001
3	14.480	0.000001
4	14.470	0.000001
5	14.500	0.000001
Итог	14.500	0.000001
1	10.170	0.000001
2	10.130	0.000001
3	10.180	0.000001
4	10.240	0.000001
5	10.270	0.000001

Красная ячейка в таблице обозначает худшее время или погрешность, зелёная – лучшее.

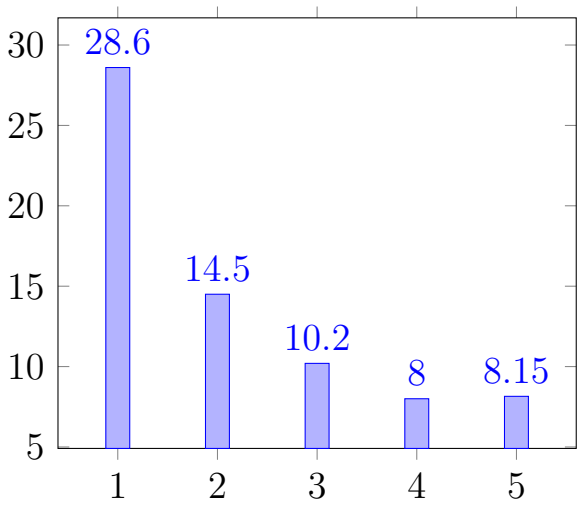

Рисунок 2.3 - Диаграмма при h = 0.005, количество итераций = 10.000

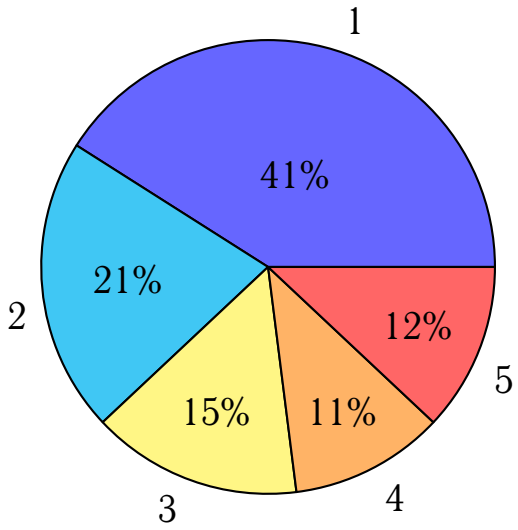

Рисунок 2.4 - Те же самые данные, но в виде "pie"диаграммы

Приведём сравнение времени выполнения последовательных и параллельных алгоритмов для алгоритмов Зейделя и красно-черного упорядочения.

Отсюда видно, что при большом количестве итераций метод Зейделя становится все менее эффективным.

Из проведенных экспериментов можно сделать следующие выводы:

- Как показали все эксперименты, кроме первого, наилучшее время всегда показывали 4 потока. Это связано с тем, сто программа тестировалась на компьютере с 4-мя ядрами. Это видно, когда мы смотрим на результаты работы программы с 8 потоками – она работает даже чуть больше, чем программа с четырьмя. Это связано с тем, что при большем количестве потоков увеличивается время переключения

их между собой. Логические ядра, судя по всему, на время работы программы не влияют.

- Для первого эксперимента никакой существенной корреляции между количеством потоков и временем работы программы выявлено не было. Это означает, что при достаточно малом шаге количество потоков перестает влиять на время работы программы.

- Наиболее эффективное количество потоков для работы на большой сетке равно количеству ядер используемого компьютера.

- Среднее ускорение при увеличении количества потоков в 2 раза составило примерно 1.8 – 1.9

- При большой размерности сетки и большом количестве итераций на практике выгоднее использовать метод Красно-черного упорядочения, а не метод Зейделя. В приведенных последовательных алгоритмах при вычислении нового значения y(i, j) используются или значения, полученные на предыдущей итерации (алгоритмы типа Якоби), или вполне конкретные значения, полученные как на предыдущей, так и на текущей итерации (алгоритмы зейделевского типа). При параллельных реализациях возможно использование значений, полученных на предыдущей и на текущей итерации, но не обязательно указанных на какой из них. Такого типа параллельные сеточные алгоритмы называются алгоритмами хаотической релаксации. Подробнее смотрите в 5, 6.

2.2 Итерационный процесс с оценкой погрешности

В данном разделе мы рассмотрим параллельный алгоритм красно-черного упорядочения на 1, 2, 3, 4, 8 потоках, также будем менять количество итераций и шаг сетки. Вместе с ним мы также рассмотрим эффективность параллельного алгоритма Зейделя и сравним их. Попробуем найти оптимальное количество потоков для решения этой задачи. Формула для расчёта погрешности во всех примерах такая: $\delta = max|u_{i,j}^{n+1} - u_{i,j}^{n}|$

Каждый вычислительный эксперимент был проведён 50 раз, в графиках приведены средние значения (Рисунок 2.5, 2.6, 2.7):

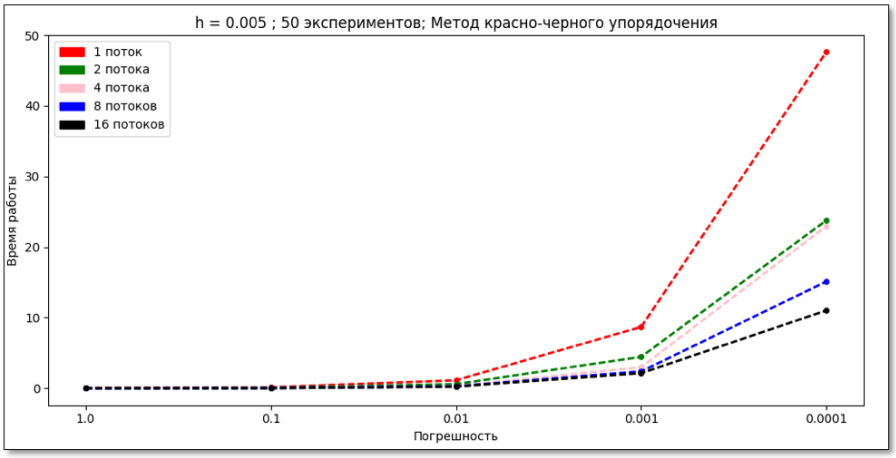

Рисунок 2.5 - Зависимость времени работы от погрешности и количества потоков (h = 0.005, Метод красно-черного упорядочения)

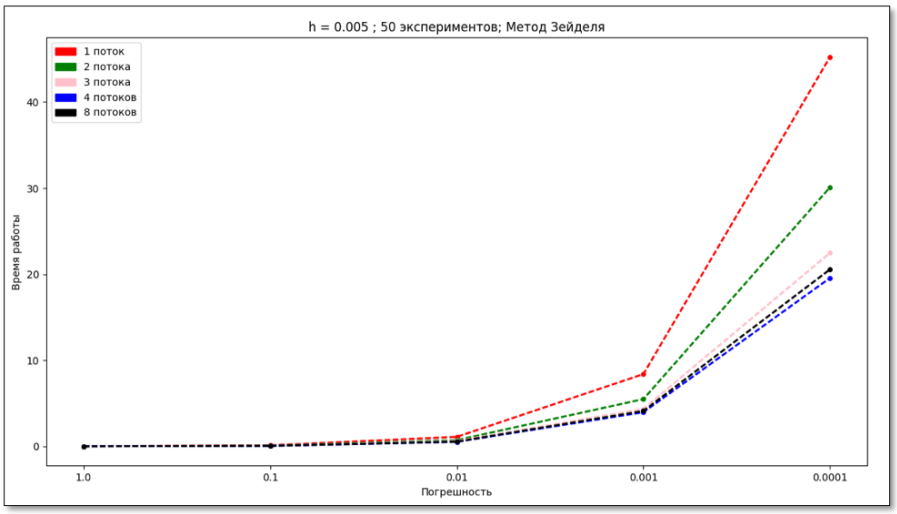

Рисунок 2.6 - Зависимость времени работы от погрешности и количества потоков (h = 0.005, Метод Зейделя)

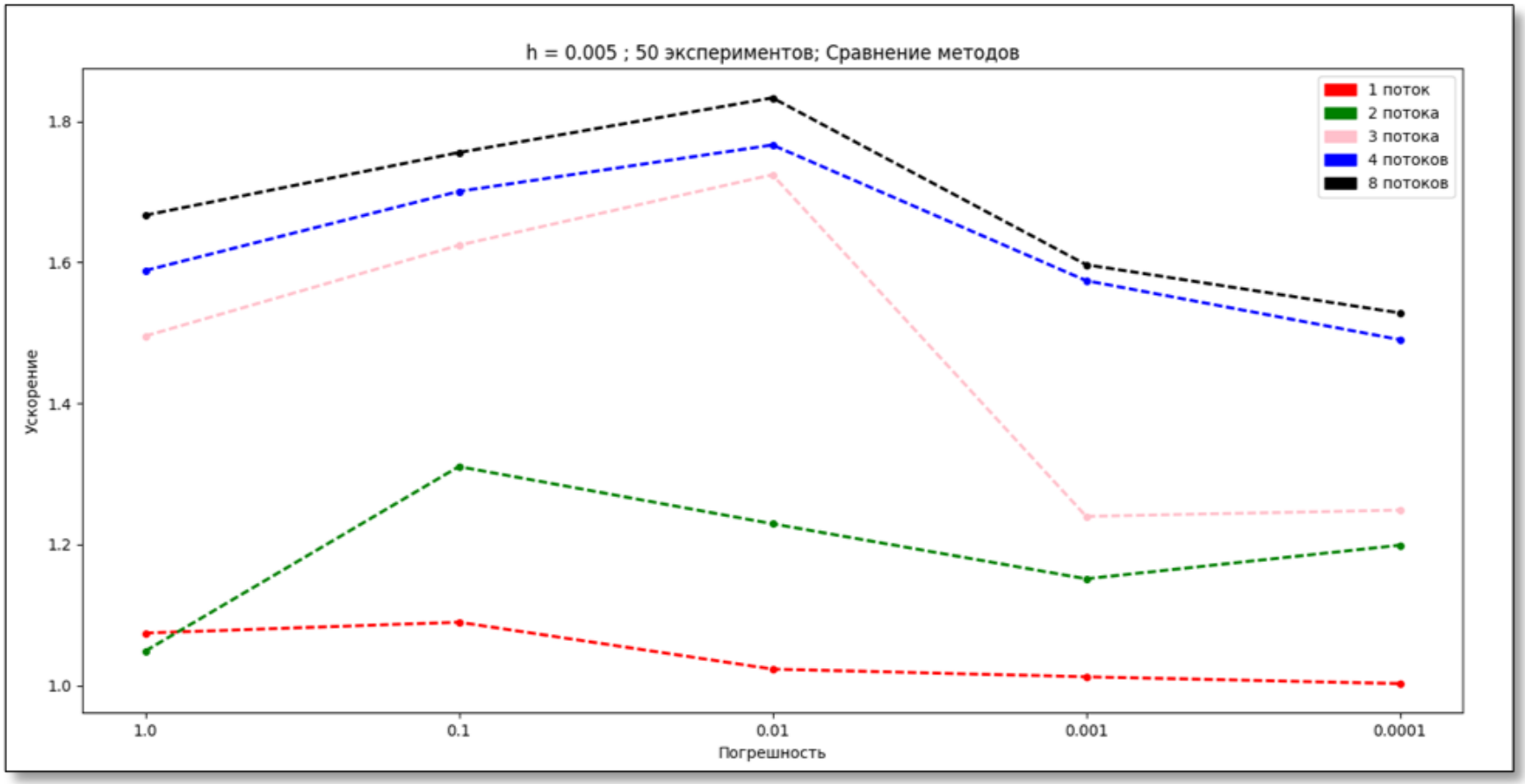

Рисунок 2.7 - $h = 0.005$, сравнение ускорения метода Красно-черного упорядочения в сравнении с методом Зейделя при разном количестве потоков и разной погрешности.

Из проведенных экспериментов можно сделать следующие выводы:

- В целом все выводы о наилучшем количестве потоков такие же, как и в прошлом разделе – целесообразно использовать количество потоков, равное количеству ядер компьютера.
- Ускорение метода красно-черного упорядочения при увеличении количества потоков в 2 раза чуть ухудшилось по сравнению с точечным параллельным алгоритмом и составило 1.7 – 1.8. Это связано с необходимостью подсчитывать погрешность вычислений на каждой итерации алгоритма.
- Ускорение метода красно-черного упорядочения по сравнению с методом Зейделя растёт с ростом числа потоков – так, например, для двух потоков оно составило в среднем 1.3, а для четырёх уже 1.65.

ЗАКЛЮЧЕНИЕ

При параллельной реализации метод красно-черного упорядочения особенно эффективен на большом количестве итераций (или на большой сетке и среднем/большом количестве итераций). При использовании пятиточечного шаблона он обладает большой по сравнению с методом Зейделя точностью на любом количестве потоков, что связано с особенностями многопоточной обработки. Для расчётов наиболее рационально использовать количество потоков, равное количеству ядер компьютера, это количество является наиболее эффективным их расчёта потраченой памяти / занятого времени. Также в ходе работы был рассмотрен метод Якоби, который даже по сравнению с методом Зейделя даёт худшее время.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Фалейчик, Б.В. Вычислительные методы алгебры: базовые понятия и алгоритмы : учеб.-метод. пособие / Б. В. Фалейчик. — Минск : БГУ, 2010. — 42 с.

2. Гринин, Л. Е. Социальная макроэволюция: генезис и трансформации Мир-Системы / Л. Е. Гринин, А. В. Коротаев. – Изд. 2-е. – М. : URSS, 2013. – 567 с.

3. Rips, L. J. Lines of thought: central concepts in cognitive psychology / L. J. Rips. – New York ; Oxford : Oxford Univ. Press, 2011. – XXII, 441 p.

4. Закономерности формирования и совершенствования системы движений спортсменов (на примере метания копья) / В. А. Боровая [и др.]. – Гомель: Гомел. гос. ун-т, 2013. – 173 с.

5. Агапов, Е. П. Методы исследования в социальной работе: учеб. пособие / Е. П. Агапов. – 2-е изд. – М. : Дашков и К ; Ростов н/Д : Наука-Спектр, 2013. – 223 с.

6. Reforming the United Nations for peace and security [Electronic resource]: proc. of a workshop to analyze the rep. of the High-level Panel on Threats, Challenges, a. Change / Yale Center for the Study of Globalization. – New Haven: Yale Center for the Study of Globalization, 2005. – Mode of access: http://www.ycsg.yale.edu/core/forms/Reforming$_un.pdf$ – Date of access: 20.02.2018.

7. Узел крепления крановых рельсов к стальным подкрановым балкам. Технические условия: СТБ 2135-2010. – Введ. 01.07.11 (с отменой на территории РБ ГОСТ 24741-81). – Минск: Белорус. гос. ин-т стандартизации и сертификации, 2011. – 6 с.

8. Национальный правовой Интернет-портал Республики Беларусь [Веб-сайт]. – Режим доступа: http://www.pravo.by. – Дата доступа: 24.06.2018.

9. ГОСТ 7.9 – 77. Реферат и аннотация. – М. : Изд-во стандартов, 1981. – 6 с..