**Министерство образования Республики Беларусь**

**БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ**

**Кафедра вычислительной математики**

ЛАВРЁНОВ

Иван Александрович

**УЛУЧШЕНИЕ ЛОКАЛЬНОСТИ И ПРОГРАММНАЯ РЕАЛИЗАЦИЯ ПАРАЛЛЕЛЬНОГО АЛГОРИТМА ПРОДОЛЬНО-ПОПЕРЕЧНОЙ ПРОГОНКИ**

Дипломная работа

|  |  |
| --- | --- |
|  | Научный руководитель:  доктор физико-математических наук,профессор Н.А. Лиходед |

Допущена к защите

“\_\_\_” \_\_\_\_\_\_\_\_\_\_\_\_\_ 2014 г

Зав. кафедрой вычислительной математики

кандидат физико-математических наук, доцент П.А. Мандрик

Минск, 2014

Белорусский государственный университет

Факультет Прикладной математики и информатики

Кафедра Вычислительной математики

“Утверждаю”

Заведующий кафедрой

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_П.А. Мандрик

“\_\_\_”  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  2013 г.

ЗАДАНИЕ

ПО ПОДГОТОВКЕ ДИПЛОМНОЙ РАБОТЫ

Студенту 5 курса 5 группы Лаврёнову Ивану Александровичу

1. Тема работы Улучшение локальности и программная реализация параллельного алгоритма продольно-поперечной прогонки

2. Срок сдачи студентом законченной работы \_\_ мая 2014 г.

3. Исходные данные к работе

* Метод переменных направлений (метод продольно-поперечной прогонки) численного решения двумерных параболических уравнений ([1, 2]).
* Метод организации зернистых вычислительных процессов на параллельных компьютерах с распределенной памятью типа белорусско-российской мультипроцессорной вычислительной системы СКИФ ([3, 4]).
* Известный параллельный алгоритм, реализующий двухшаговую разностную схему продольно-поперечной прогонки (рабочие материалы).
* Способ улучшение локальности параллельных алгоритмов, требующих групповых коммуникационных операций (обращение каждого процесса к локальной памяти всех других процессов) для «транспонирования» матрицы промежуточных результатов вычислений. ([5], рабочие материалы).
* Средства реализации параллельных программ ([6–8]).

Библиографические описания источников, рекомендуемых студентам к ознакомлению при выполнении работы:

* 1. Самарский А.А. Теория разностных схем. – М: Наука. 1989. – 616 c.
  2. Полевиков В.К. Численные методы математической физики. Курс лекций. – Мн: 2010. - 107 c. [Электрон. ресурс – \\fpmi-stud\Subfaculty\Каф. Выч. Мат\Polevikov]
  3. [Электрон. ресурс – Персональная страница Лиходеда Н.А. на wwwbsu.by] Лекции. [Электрон. ресурс – \\fpmi-stud\Subfaculty\Каф. Выч. Мат\Параллельные вычисления\Лекции], лекция «Тайлинг».
  4. [Электрон. ресурс – Персональная страница Лиходеда Н.А. на wwwbsu.by] Лекции. [Электрон. ресурс – \\fpmi-stud\Subfaculty\Каф. Выч.
  5. Мат\Параллельные вычисления\Лекции], лекция «Организация параллельных зернистых вычислительных процессов».
  6. Баханович С.В., Лиходед Н.А., Мандрик П.А. Параллельные алгоритмы численного решения двумерных квазилинейных параболических уравнений локально-одномерным методом // Международный конгресс по информатике: информационные системы и технологии CSIST'2013, 4–7 ноября 2013 г., Минск, Беларусь. Бел. гос. ун-т. – Минск, 2013. С. 356–360.
  7. Антонов А.С. "Параллельное программирование с использованием технологии MPI". М.: Изд-во МГУ, 2004. – 71 с. <http://parallel.ru/info/parallel/antonov/>
  8. Шпаковский Г.И., Серикова Н.В. Программирование для многопроцессорных систем в стандарте MPI. Пособие. – Мн.: БГУ, 2002. – 323 с.
  9. MPICH: Реализация MPI и среда разработки MPI-программ. – http://parallel.ru/Технологии

Базовый теоретический материал изучается в 5–8 семестрах в дисциплинах: Программирование, Уравнения математической физики, Численные методы математической физики, Компьютерные сети, а также в дисциплине специализации Лиходеда Н.А. – 7 семестр.

4. Перечень вопросов, подлежащих разработке или краткое содержание работы

* Изучить метод переменных направлений (метод продольно-поперечной прогонки) численного решения двумерных параболических уравнений ([1, 2]).
* Изучить основы теории организации параллельных зернистых вычислительных процессов ([3, 4])
* Изучить вычислительный алгоритм, ознакомиться с известным параллельным алгоритмом и программами, реализующими двухшаговую разностную схему продольно-поперечной прогонки (рабочие материалы).
* Изучить способ улучшение локальности параллельных алгоритмов, требующих групповых коммуникационных операций (обращение каждого
* процесса к локальной памяти всех других процессов) для «транспонирования» матрицы промежуточных результатов вычислений. ([5], рабочие материалы).
* Разработать новый параллельный алгоритм продольно-поперечной прогонки с улучшенной локальностью и, следовательно, с лучшими характеристиками коммуникационных операций. Параметрами алгоритма должны быть размеры задачи, число процессоров, используемых в вычислениях, размеры зерна вычислений.
* Программно реализовать (С, MPI) разработанный параллельный алгоритм. Отладить программу на персональном компьютере с помощью MPICH или аналогичной среды разработки MPI-программ.
* Провести и проанализировать вычислительные эксперименты на суперкомпьютере СКИФ.
* Оформить результаты. Составить компьютерную презентацию к докладу работы на защите.

5. Перечень графического материала

* Логотип БГУ для включения на слайды презентации.
* Схемы, поясняющие получение параллельного алгоритма.
* Графики и (или) таблицы зависимости ускорения и времени вычислений от параметров параллельного алгоритма.

6. Календарный графикработы

* октябрь-ноябрь – изучение задания, работа с научными статьями, электронными ресурсами;
* ноябрь-февраль – преддипломная практика;
* ноябрь-май – практическая реализация задач работы;
* февраль – отчет о преддипломной практике;
* апрель-май– оформление результатов работы (отчета DOC, презентации PPT, приложений), подготовка доклада и презентации на защиту;
* май –представление работы с отзывом руководителя на кафедру, рецензенту;
* май –предзащита на кафедре;
* июнь– защита на ГЭК.

Руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_ / Н.А. Лиходед / …. сентября 2013 г.

Задание принял к исполнению \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ .… октября 2013 г.

(подпись студента)

**РЕФЕРАТ**

Дипломная работа, 47 страниц, 8 иллюстраций, 2 таблицы, 7 источников.

***Ключевые слова:*** ПРОДОЛЬНО-ПОПЕРЕЧНАЯ ПРОГОНКА, ПАРАЛЛЕЛЬНЫЙ АЛГОРИТМ, ТАЙЛИНГ, ЛОКАЛЬНОСТЬ АЛГОРИТМА, КОММУНИКАЦИОННЫЕ ОПЕРАЦИИ.

***Объект исследования***  – параллельные реализации метода решения двумерного нелинейного уравнения теплопроводности с начальными и граничными условиями.

***Цель работы*** – улучшить известный параллельный алгоритм решения двумерного нелинейного уравнения теплопроводности методом продольно-поперечной прогонки.

***Методы исследования*** – поиск информационных зависимостей, выявление независимых частей алгоритма, тайлинг, организация параллельных вычислительных процессов, получение коммуникационных операций.

***В результате*** разработан параллельный алгоритм с улучшенной локальностью.

***Областью применения*** являются расчёты тепловых процессов, использующие продольно-поперечную прогонку.

**РЭФЕРАТ**

Дыпломная праца, 47 старонак, 8 малюнкаў, 2 табліцы, 7 крыніц.

***Ключавыя словы:*** ПАДОЎЖАНА-ПАПЯРЭЧНАЯ ПРАГОНКА, ПАРАЛЕЛЬНЫ АЛГAРЫТМ, ТАЙЛІНГ, ЛАКАЛЬНАСЦЬ АЛГАРЫТМУ, КАМУНІКАЦЫЙНЫЯ АПЕРАЦЫІ.

***Аб’ект даследвання***  – паралельныя рэалізацыі паралельнага алгарытму колькаснага рашэння двухмернага нелінейнага ўраўнення цеплаправоднасці з пачатковымі і межавымі умовамі.

***Мэта працы*** – палепшыць вядомы паралельны алгарытм колькаснага рашэння двухмернага нелінейнага ўраўнення цеплаправоднасці метадам падоўжана-папярэчнай прагонкі.

***Метады даследавання*** – пошук інфармацыйных залежнасцяў, выяўленне незалежных частак алгарытму, тайлінга, арганізацыя паралельных вылічальных працэсаў, атрыманне камунікацыйных аперацый.

***У выніку*** распрацаваны паралельны алгарытм палепшанай лакальнасці.

***Вобласцю ўжывання*** з'яўляюцца разлікі цеплавых працэсаў, якія выкарыстоўваюць падоўжана-папярэчную прагонку.

**ABSTRACT**

Diploma thesis, 47 pages, 8 figures, 2 tables, 7 sources.

***Keywords:*** LONGITUDINAL-TRANSVERSE SWEEP, PARALLEL ALGORITHMS, TILING, LOCAL ALGORITHMS, COMMUNICATION OPERATIONS.

***Object of study*** – parallel implementation of the method of two-dimensional solutions of the nonlinear heat equation with initial and boundary conditions.

***Purpose*** – improve the well-known parallel algorithm for solving the two-dimensional nonlinear heat equation method of longitudinal-transverse sweep.

***Research methods*** – search information dependencies, identify independent parts of the algorithm, the tiling, the organization of parallel computing processes, getting the communication operations.

***As a result*** the parallel algorithm is developed for improving locality.

***Area of applications*** is the calculation of thermal processes using longitudinal-transverse sweep.

**ОГЛАВЛЕНИЕ**

[ВВЕДЕНИЕ 6](#_Toc389708767)

[Глава 1 РАСЧЁТНАЯ СХЕМА ПРОГОНКИ ДЛЯ НЕЛИНЕЙНОГО УРАВНЕНИЯ ТЕПЛОПРОВОДНОСТИ 8](#_Toc389708768)

[1.1 Постановка разностной задачи 8](#_Toc389708769)

[1.2 Расчётная схема прогонки 9](#_Toc389708770)

[Глава 2 ПАРАЛЛЕЛЬНЫЙ АЛГОРИТМ 11](#_Toc389708771)

[2.1 Тайлинг 11](#_Toc389708772)

[2.2 Алгоритм 1 12](#_Toc389708773)

[2.3 Упрощение алгоритма 13](#_Toc389708774)

[2.4 Графическая интерпретация 13](#_Toc389708775)

[2.5 Результаты вычислительных экспериментов 14](#_Toc389708776)

[Глава 3 ПАРАЛЛЕЛЬНЫЙ АЛГОРИТМ С УЛУЧШЕННОЙ ЛОКАЛЬНОСТЬЮ 16](#_Toc389708777)

[3.1 Локальность алгоритма 16](#_Toc389708778)

[3.2 Алгоритм 2 16](#_Toc389708779)

[3.3 Графическая интерпретация 18](#_Toc389708780)

[3.4 Результаты вычислительных экпериментов 20](#_Toc389708781)

[3.5 Сравнительный анализ двух алгоритмов 22](#_Toc389708782)

[ЗАКЛЮЧЕНИЕ 24](#_Toc389708783)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 25](#_Toc389708784)

[ПРИЛОЖЕНИЕ А 26](#_Toc389708785)

[ПРИЛОЖЕНИЕ Б 36](#_Toc389708786)

# ВВЕДЕНИЕ

В настоящее время существует множество разностных методов для решения двумерных параболических уравнений. Одним из них является метод прогонки, позволяющий свести решение двумерной задачи к нахождению последовательности одномерных задач ([5],стр.250). Такой метод обладает устойчивостью и простотой решения. Он достаточно универсален, позволяет решать уравнения с переменными коэффициентами и не накладывает сильных ограничений на вид граничных условий. Однако для объёмных численных экспериментов не всегда хватает вычислительных мощностей. С целью увеличения производительности используется принцип параллельной обработки данных на вычислительных системах кластерного типа, одной из которых является белорусско-российская мультипроцессорная система СКИФ. Параллельная обработка данных в большинстве своём строится на естественном параллелизме используемого алгоритма решения поставленной задачи.

В данной работе предлагается отказаться от использования части естественного параллелизма и получить параллельные алгоритмы с улучшенной локальностью, с целью ускорения вычислений.

Для достижения поставленной цели необходимо:

1. Изучить метод переменных направлений (метод продольно-поперечной прогонки) численного решения двумерных параболических уравнений ([5]).
2. Изучить основы теории организации параллельных зернистых вычислительных процессов ([3, 7])
3. Изучить вычислительный алгоритм, ознакомиться с известным параллельным алгоритмом и программами, реализующими двухшаговую разностную схему продольно-поперечной прогонки (рабочие материалы).
4. Изучить способ улучшение локальности параллельных алгоритмов, требующих групповых коммуникационных операций (обращение каждого процесса к локальной памяти всех других процессов) для «транспонирования» матрицы промежуточных результатов вычислений ( рабочие материалы).
5. Разработать новый параллельный алгоритм продольно-поперечной прогонки с улучшенной локальностью и, следовательно, с лучшими характеристиками коммуникационных операций. Параметрами алгоритма должны быть размеры задачи, число процессоров, используемых в вычислениях, размеры зерна вычислений.
6. Программно реализовать (С, MPI) разработанный параллельный алгоритм. Отладить программу на персональном компьютере с помощью MPICH или аналогичной среды разработки MPI-программ.
7. Провести и проанализировать вычислительные эксперименты на суперкомпьютере СКИФ.

# РАСЧЁТНАЯ СХЕМА ПРОГОНКИ ДЛЯ НЕЛИНЕЙНОГО УРАВНЕНИЯ ТЕПЛОПРОВОДНОСТИ

## Постановка разностной задачи

Рассмотрим в области

|  |  |
| --- | --- |
|  | (1.1) |

двумерное нелинейное уравнение теплопроводности

|  |  |
| --- | --- |
|  | (1.2) |

с начальными

|  |  |
| --- | --- |
| , | (1.3) |

и граничными условиями

|  |  |
| --- | --- |
| . | (1.4) |

Введем в области  сетку узлов , , , , , , а на отрезке  сетку узлов .

Запишем разностную схему для заданной задачи:

|  |  |
| --- | --- |
|  | (1.5) |

где

|  |  |
| --- | --- |
| , | (1.6) |

|  |  |
| --- | --- |
| . | (1.7) |

Зафиксировав в первом из уравнений системы *j*, получим систему уравнений относительно значений , где , состоящую из линейного уравнения, которую можно решить методом прогонки.

В целом систему (1.5) на каждом половинном временном слое можно представить как  независимую задачу (для каждого фиксированного *j*), решаемую методом прогонки.

Аналогично решение второго из уравнений системы (1.5) на каждом слое  представляет собой решение  независимой задачи при фиксированном *i*. Каждая из указанных задач является системой линейных уравнений относительно значений сеточной функции по неявному направлению и решается методом прогонки.

Сеточная функция  является приближенным решением задачи (1.1)–(1.3).

## Расчётная схема прогонки

По каждому из неявных направлений разностная схема является линейной и может быть записана в следующем виде:

|  |  |
| --- | --- |
| , | (1.8) |

|  |  |
| --- | --- |
| , | (1.9) |

|  |  |
| --- | --- |
|  | (1.10) |

и соответственно

|  |  |
| --- | --- |
| , | (1.11) |

|  |  |
| --- | --- |
| , | (1.12) |

|  |  |
| --- | --- |
|  | (1.13) |

Для решения уравнений (1.8) и (1.11) воспользуемся формулами прогонки

|  |  |
| --- | --- |
|  | (1.14) |

где  – индекс неявного направления.

Из граничных условий при  и  определяются значения прогоночных коэффициентов. При этом  и  равны нулю, а значения  и  определяются из соответствующих краевых условий.

# ПАРАЛЛЕЛЬНЫЙ АЛГОРИТМ

## Тайлинг

Тайлинг (tiling) – это преобразование алгоритма для получения макроопераций. Получаемые в результате преобразования макрооперации называются зерном вычислений или тайлами. Множество операций алгоритма, составляющих зерно вычислений, выполняется атомарно, как одна единица вычислений; при параллельных вычислениях выполнение операций одного зерна не может прерываться синхронизацией или обменом данными. Разбиение множества операций алгоритма на тайлы позволяет при подходящем задании тайлов организовать параллельные вычислительные процессы и уменьшить накладные расходы на обмен данными при параллельных вычислениях и на использование памяти при последовательных вычислениях.

Использование тайлинга приводит, в частности, к известным блочным алгоритмам. Но идея тайлинга состоит не в разбиении на блоки массивов данных, а в разбиении множества операций алгоритма. Вариантов разбиения множества вычислений, вообще говоря, гораздо больше, чем вариантов разбиения массивов данных. Это связано с тем, что размерности итерационных пространств алгоритмов обычно превосходят размерности массивов данных.

При тайлинге каждый цикл разбивается на два цикла: глобальный, параметр которого определяет на данном уровне вложенности порядок вычисления тайлов, и локальный, в котором параметр исходного цикла изменяется в границах одного тайла. Допускается вырожденное разбиение цикла, при котором все итерации относятся к глобальному циклу или все итерации относятся к локальному циклу. Такие циклы будем называть соответственно глобальными неразбиваемыми и локальными неразбиваемыми.

По смыслу тайлинга, для каждого набора операторов локальные циклы должны быть самыми внутренними. Поэтому общие для операторов локальные циклы распределяются между циклами операторов, если только операторы не окружены одинаковым наборов циклов; локальные циклы переставляются с глобальными и становятся самыми внутренними.

В организации параллельных зернистых вычислительных процессов выделяют следующие этапы:

1. тайлинг;
2. запись параллельных зернистых вычислительных процессов;
3. запись операций тайла;
4. структурирование коммуникаций;
5. псевдокод параллельного зернистого алгоритма с обменом информации

## Алгоритм 1

Схематично основную вычислительную часть известного параллельного алгоритма [4, стр.3], реализующего двухшаговую разностную схему продольно-поперечной прогонки (т.е. метода переменных направлений) можно представить следующим образом (циклы, итерации которых заведомо можно выполнять независимо, запишем как **dopar**):

do *n =* 0*, k*

**dopar** *j=* 1*, N*y *–*1 // *Начало тайла первого типа*

do *i=* 1*, N*x *–*1





enddo

do *i=* 1*, N*x *–*1



enddo

**enddopar**

**dopar** *i=* 1*, N*x *–*1 // *Начало тайла второго типа*

do *j=* 1*, N*y *–*1





enddo

do *j=* 1*, N*y *–*1



enddo

**enddopar**

enddo(*n*)

Здесь  и  – коэффициенты прогонки, которые при подстановки формул (1.8)–(1.10) и (1.11)–(1.13) в (1.14) с учётом неявного направления текущей прогонки, выражаются через следующие функции:

|  |  |
| --- | --- |
| , | (2.1) |

|  |  |
| --- | --- |
|  | (2.2) |

|  |  |
| --- | --- |
|  | (2.3) |

|  |  |
| --- | --- |
|  | (2.4) |

Для приведённого псевдокода опущены операции пересылки данных между процессами, а также инициализации начальных и граничных значений с их последующим пересчётом по ходу выполнения алгоритма.

Конечные значения сеточной функции  и будут являются приближённым решением исходной задачи (1.1)–(1.3).

## Упрощение алгоритма

Отметим, что для первой («продольной») прогонки при вычисления прогоночных коэффициентов для  используем граничные и начальные условия, в то время как во второй прогонке при вычислении коэффициентов для  необходимо знать значения . Нетрудно заметить, что операциях пересылки между слоями можно использовать не сами значения сеточной функции на новом слое, а заранее вычисленные значения  и , что значительно уменьшит объём пересылаемых данных.

## Графическая интерпретация

Покажем графическую взаимосвязь между процессом решения каждой из прогонок на двух последовательных итерациях работы алгоритма. На рисункеРисунок 2.1 схематично изображены «повороты» матриц промежуточных результатов вычислений, порождающие в параллельном алгоритме обращение каждого процесса к локальной памяти всех других процессов.

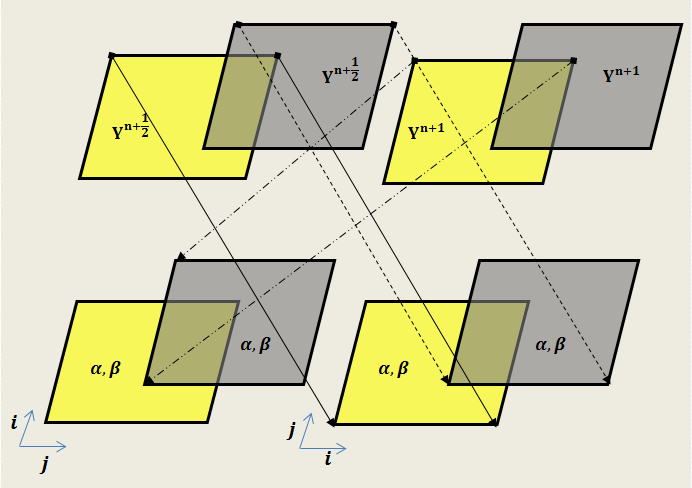


Рисунок 2.1- Процесс решения прогонки алгоритмом 1 (разными цветами изображены процессы, идущие на разных временных слоях)

## Результаты вычислительных экспериментов

Алгоритм 1 для численного решения двумерного нелинейного уравнения теплопроводности программно реализован на языке Си с использованием библиотеки MPI (ПРИЛОЖЕНИЕ А). Эксперименты, проводились на модернизируемом суперкомпьютере СКИФ (БГУ) для области  при следующих начальных и граничных условия:

|  |  |
| --- | --- |
|  | (2.5) |

Значение коэффициента теплопроводности было задано как:

|  |  |
| --- | --- |
|  | (2.6) |

В качестве параметров алгоритма были взяты:

|  |  |
| --- | --- |
|  | (2.7) |

Результаты вычислительного эксперимента представлены на рисункеРисунок 2.2:

Рисунок 2.2 - График зависимости времени выполнения от количества процессоров

# ПАРАЛЛЕЛЬНЫЙ АЛГОРИТМ С УЛУЧШЕННОЙ ЛОКАЛЬНОСТЬЮ

## Локальность алгоритма

Локальность алгоритма – это вычислительное свойство алгоритма, отражающее степень использования памяти с быстрым доступом. При многопроцессорной обработке памятью с быстрым доступом считается локальная память процессора, при однопроцессорной – кэш.

Локальность параллельного алгоритма, предназначенного для реализации на компьютерах с распределенной памятью, характеризует коммуникационные затраты: чем меньше при заданном числе вычислительных ядер суммарный объем операций обмена данными, тем лучше локальность. Задачей исследования локальности параллельного алгоритма является оценка числа и объема коммуникационных операций. Улучшения локальности можно достичь путем преобразования алгоритмов и/или удачного распределения операций и данных между процессорами.

## Алгоритм 2

В известном параллельном алгоритме 1, основанном на естественном параллелизме, каждый процесс перед выполнением каждого тайла получает данные от всех других процессов, а после выполнения тайла пересылает данные всем другим процессам. С ростом числа используемых процессов накладные расходы на коммуникационные операции резко возрастают.

Предложим параллельный алгоритм, в котором операции и данные перераспределены между процессами таким образом, что значительная часть данных приватизирована процессами и не требует коммуникационных операций.

Заметим, что коэффициенты прогонки и последующий пересчёт значений искомой функции происходит рекурсивно, и для заданной произвольным тайлом части сеточной функции не требуется обмена с соседними процессами, если высчитываемые значения локализованы в памяти текущего процессора. Единственным условием, налагаемым на данное предположение, является необходимость получения начальных данных для выполнения рекурсивного подсчёта. Его можно реализовать путём пересылки *point-to-point* (от одного процессора другому). Данная реализация подразумевает процессы «разгона» и «торможения» алгоритма, т.к. только получив данные от предыдущего процесса текущий начинает подсчёт. Это увеличивает время выполнения алгоритма, однако уменьшение операций пересылки между процессами должно дать больший прирост в скорости, чем замедление за счёт указанного выше недостатка. Также ускорение вычислений должен вызвать тот факт, что для второй прогонки пересылка данных не потребуется вовсе, т.к. все искомые значения уже будут в локальной памяти каждого из процессов.

Будем для простоты изложения предполагать, что  является целым числом, где *Р* – это общее число процессов, предназначенных для реализации алгоритма. Нумерацию процессов будем начинать с нуля; произвольный *p*-й процесс будем обозначать *Prp.* Количество тайлов, выполняемые каждым процессом обозначим как *Q*, а также введём величины  и , которые представляют собой выражения  и  соответственно. Величину  будем называть размером тайла.

Выделим три типа зёрен вычислений (тайлов):

1. В тайле 0-го типа происходят вычисления прогоночных коэффициентов  и (условия: 1+*pr*1  *i*  (*p +*1)*r*1, 1+*q*2*r*2  *j* ** min((*q*2*+*1)*r*2,  *N*y*–*1), 0  *q* ** *Q –*1).После выполнения тайла нулевого типа каждый процесс *Prp* должен передать «граничные»  процессу *Prp+1*,в качестве исходных для рекурсивного вычисление прогоночных коэффицентов (условие: );
2. В тайле 1-го типа происходит вычисления значений сеточной функции на промежуточном временном слое, а затем «граничные» (условие:1+*q*2*r*2  *j* ** min((*q*2*+*1)*r*2,*Nx-*1)) пересылаются каждым процессом *Prp* процессу *Prp–1* (условие: );
3. В тайле 2-го типа происходит решение второй прогонки для получения значений функций на новом временном слое, и пересылка здесь не требуется.

Представим описанный алгоритм в виде псевдокода с пояснениями.

Для каждого процесса , *0pP–1*:

do *n =*0*, k*

do *q=* 0, *Q –*1

**dopar** *j=* 1+*q r*2,min((*q +*1)*r*2, *N*y *–*1) *// Начало тайла 0-го типа.*

do *i=* 1+*p r*1,(*p +*1)*r*1





enddo

**enddopar**(*j*) *// Конец тайла 0-го типа. Обмен «граничными» данными.*

enddo(*q*)

do *q=* 0,*Q –*1

**dopar** *j=* 1+*q r*2,min((*q +*1)*r*2,*N*y *–*1) *// Начало тайла 1-го типа*

do *i=* 1+(*P*–*p*–1) *r*1,(*P*–*p*)*r*1



enddo

**enddopar**(*j*) *// Конец тайла 1-го типа. Обмен «граничными» данными.*

enddo(*q*)

**dopar** *i=* 1+*p r*1,(*p +*1)*r*1 *// Начало тайла 2-го типа.*

do *j=* 1*, N*y *–*1





enddo

do *j=* 1*, N*y *–*1



enddo

**enddopar**(*i*) *// Конец тайла 2-го типа.*

enddo(*n*)

Явный вид функций *F1, F2, F3, F4* аналогичен приведённым ранее формулам (2.1)–(2.4).

## Графическая интерпретация

Чтобы избежать излишней загромождённости ввиду большого количества отображаемых операций, для наглядности изобразим каждый тип тайла по отдельности:

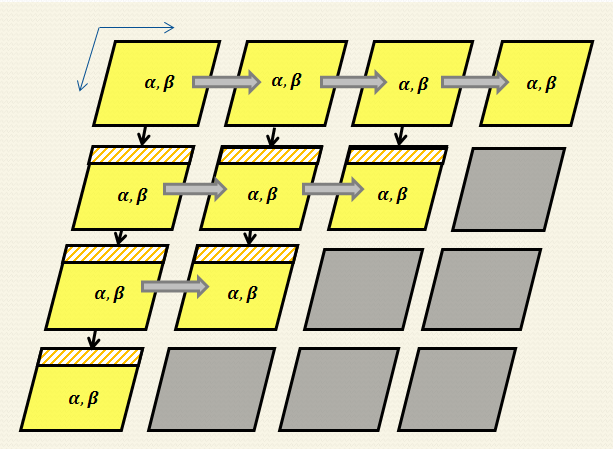


Рисунок 3.1 – Тайл 0-го типа

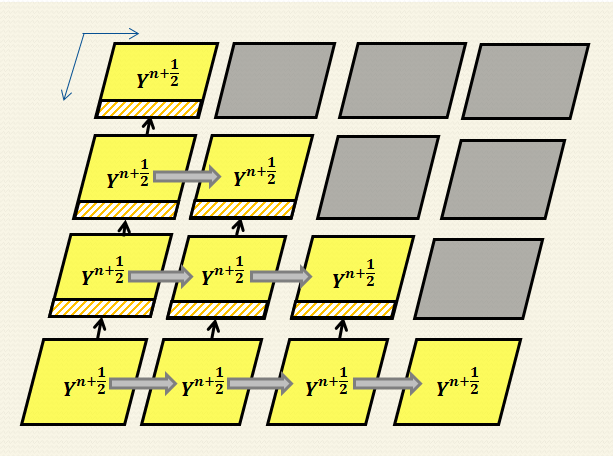


Рисунок 3.2 – Тайл 1-го типа

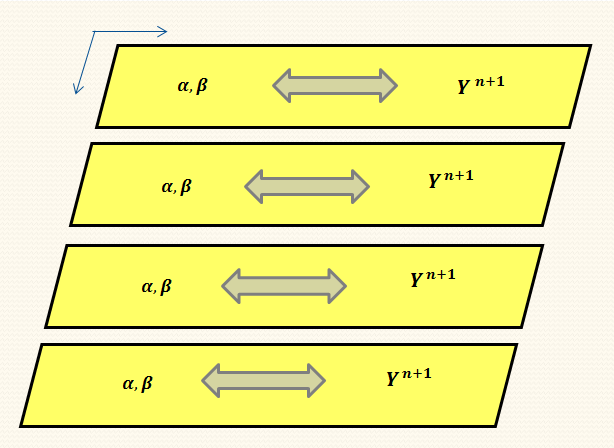


Рисунок 3.3 – Тайл 2-го типа

## Результаты вычислительных экпериментов

Алгоритм 2 программно реализован на языке Си с использованием библиотеки MPI (ПРИЛОЖЕНИЕ Б). Эксперименты, проводились на модернизируемом суперкомпьютере СКИФ (БГУ) для области  при условиях (2.5)–(2.7) и различном размере тайлов.

Для алгоритма 2 необходимо проверить быстродействие для различных размеров тайла и числа используемых процессов. Для начала зафиксируем количество процессов, взяв его достаточно большим, например, равным 20, и установим зависимость времени выполнения алгоритма от размера тайла

Таблица 3.1 - Зависимость времени выполнения от размера тайла

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Размер тайла** | 144 | 72 | 36 | 18 | 10 | 6 | 2 | 1 |
| **Время выполнения** | 1,4056 | 1,1573 | 1,0557 | 1,0276 | 0,9734 | 1,0067 | 1,1758 | 1,4722 |

Из таблицы Таблица 3.1 видно, что наименьшее время работы второго алгоритма приходится на заданный размер тайла в интервале от 6 до 18. Проведём дополнительный эксперимент, и сравним полученные результаты, оформив их в виде таблицы Таблица 3.2:

Таблица 3.2- Зависимость времени выполнения от размера тайла

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Размер тайла** | 18 | 12 | 10 | 9 | 8 | 6 |
| **Время выполнения** | 1,0276 | 0,9742 | 0,9734 | 0,9882 | 0,9902 | 1,0067 |

Наибольшую скорость показал алгоритм с размером тайла .

Исследуем зависимость ускорения от количества процессов для данного размера тайла, сравнивая с произвольными параметрами размера зерна вычислений (рисунок Рисунок 3.4):

Рисунок 3.4 – График зависимости ускорения от числа процессов

Из графика видно, что наибольшее ускорение также достигается для найденного значения тайла . Ниже приведён отдельный график зависимости времени выполнения от числа процессов с фиксированным оптимальным значением размера тайла для алгоритма 2:

Рисунок 3.5 - График зависимости времени выполнения алгоритма от числа процессов для оптимального размера тайла

## Сравнительный анализ двух алгоритмов

Рисунок 3.6 - Сравнительный график зависимости времени выполнения от числа процессов для алгоритма 1 и алгоритма 2

Как видно из рисунка Рисунок 3.6, алгоритм 2 с улучшенной локальностью обладает большим ускорением и лучшей скоростью работы, чем алгоритм 1, особенно при достаточно небольшом (порядка 10) количестве процессов. Поэтому можно сделать вывод, что целесообразнее использовать алгоритм 2 для расчётов – как на кластерах типа СКИФ (БГУ), так и на малом количестве процессов, которые могут быть представлены соединёнными в общую сеть несколькими персональными компьютерами.

# ЗАКЛЮЧЕНИЕ

Для отображения алгоритмов на параллельные компьютеры с распределенной памятью требуется распределить операции алгоритма между процессами, распределить данные между процессами, установить порядок выполнения операций в каждом процессе и организовать обмены данными. Однако с ростом числа используемых процессов накладные расходы на операции пересылки резко возрастают. Для решения данной проблемы в дипломной работе удалось использовать замену коммуникационных операций между процессами на приватизацию данных, необходимых для вычислений.

В результате работы:

1. изучен метод продольно-поперечной прогонки для решения двумерного нелинейного уравнения теплопроводности;
2. изучены основы теории организации параллельных зернистых вычислительных процессов;
3. изучена методика построения параллельных алгоритмов, основанных на естественном параллелизме; разработан алгоритм для решения поставленной задачи, проведены необходимые вычисления и сравнения результатов;
4. изучена методика построения параллельных алгоритмов, основанных на отказе от части коммуникационных операций в пользу приватизации данных процессами;
5. разработан параллельный алгоритм с улучшенной локальностью, реализующий метод продольно-поперечной прогонки для решения двумерного нелинейного уравнения теплопроводности.
6. произведён сравнительный анализ двух алгоритмов.
7. сделан вывод о целесообразности использования в расчётах алгоритма 2.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Баханович С.В. Параллельные алгоритмы численного решения двумерных квазилинейных параболических уравнений локально-одномерным методом // Лиходед Н.А., Мандрик П.А. Международный конгресс по информатике: информационные системы и технологии CSIST'2013, 4–8 ноября 2013 г., Минск, Беларусь. Бел. гос. ун-т. – Минск, 2013. Т. 2 C 78 -81.
2. Воеводин, В.В. Параллельные вычисления / В.В. Воеводин, Вл.В. Воеводин. – СПб. : БХВ-Петербург, 2002. – 600 с.
3. Лиходед, Н.А. Методы распараллеливания гнезд циклов : Курс лекций / Н.А. Лиходед. Минск : БГУ, 2008. – 100 с.
4. Рычков А.Д. Курсовые работы по параллельным вычислительным технологиям. Пособие [Электронный ресурс] - 2013. - Режим доступа: <http://csc.sibsutis.ru/sites/csc.sibsutis.ru/files/courses/pvt/%20работы.pdf>. - Дата доступа: 25.05.2014.
5. Самарский, А.А. Теория разностных схем / А.А. Самарский. – М. : Наука, 1989. – 616 c.
6. Шпаковский Г.И. Программирование для многопроцессорных систем в стандарте MPI. Пособие. // Шпаковский Г.И., Серикова Н.В. – Мн.: БГУ, 2002. – 323 с.
7. MPICH: Реализация MPI и среда разработки MPI-программ. –[Электронный ресурс] – 2014. - Режим доступа: <http://parallel.ru/tech/tech_dev/mpi.html>. - Дата доступа: 25.05.2014.

# ПРИЛОЖЕНИЕ А

Параллельная реализация на языке C решения двумерного нелинейного уравнения теплопроводности методом продольно–поперечной прогонки

#include <mpi.h>

#include <math.h>

#include <stdio.h>

#include <stdlib.h>

double \*\*Y, \*\*Ynew;

double \*\*Lambda;

#define L1 0.001

#define L2 0.00001

#define x\_0 0.0

#define x\_L 1.0

#define y\_0 0.0

#define y\_L 0.5

#define T 3.0

#define N 1441

#define N\_t 10

int N\_p, N\_q, jt;

#define h\_x (x\_L-x\_0)/N

#define h\_y (y\_L-y\_0)/N

#define tau 0.1

double u(double x, double y, double t) {

return exp(L\*t+(x+y)/sqrt(2.)); }

double u0(double x, double y,double t) {

return exp((x+y)/sqrt(2.)); }

double u\_x0(double x,double y, double t) {

return exp(L\*t+y/sqrt(2.)); }

double u\_xN(double x,double y, double t) {

return exp(L\*t+(y+x\_L)/sqrt(2.)); }

double u\_y0(double x,double y, double t) {

return exp(L\*t+x/sqrt(2.)); }

double u\_yN(double x,double y, double t) {

return exp(L\*t+(x+y\_L)/sqrt(2.)); }

double \* vektor\_progonki (double \* F,double \*Lambda, double h, int NN)

{

double \*alpha = (double\*)malloc(sizeof(double)\*(NN+1));

double \*betta = (double\*)malloc(sizeof(double)\*(NN+1));

int i;

for(i=0;i<NN+1;i++)

{

alpha[i]=0;

betta[i]=0;

}

alpha[1]=0;

betta[1]=F[0];

for(i=1;i<NN;i++)

{

double A;

A = -(Lambda[i-1]+Lambda[i])/(4\*h\*h);

double B ;

B = -(Lambda[i+1]+Lambda[i])/(4\*h\*h);

double C;

C = 1/tau-A-B;

alpha[i+1]=-B/(C+A\*alpha[i]);

betta[i+1]=(F[i]-A\*betta[i])/(C+A\*alpha[i]);

}

double \* Y =(double\*)malloc(sizeof(double)\*(NN+1));

for(i=0;i<N+1;i++)

Y[i]=0;

Y[NN]=F[N];

int k;

for(k=NN-1;k>=0;k--)

{

Y[k] = alpha[k+1]\*Y[k+1]+betta[k+1];

}

free(alpha);

free(betta);

return Y;

}

void print\_matrix(double \*\*Y)

{

printf("\n Matrix:\n");

int i;

int j;

for(i=0;i<N+1;i++)

{

for(j=0;j<=N\_p+1;j++)

{

printf("%.5f\t", Y[i][j]);

}

printf("\n");

}

}

int main(int argc,char \*argv[])

{

int myid, proc;

int i,j,ii,jj;

int ia,ja,iia,jja,iter;

double t1,t2;

MPI\_Init(&argc,&argv);

MPI\_Comm\_size(MPI\_COMM\_WORLD, &proc);

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &myid);

MPI\_Status status;

if(proc == 1) {

N\_p=N/proc-1;

}

else{

N\_p=N/proc;

}

N\_q = N\_p;

iter = 0;

while(iter\*tau<=T)

{

t1 = MPI\_Wtime();

Y = (double\*\*)malloc(sizeof(double\*)\*(N+1));

Ynew = (double\*\*)malloc(sizeof(double\*)\*(N+1));

Lambda =(double\*\*)malloc(sizeof(double\*)\*(N+1));

int i\_r;

int j\_r;

for (i\_r = 0; i\_r < N+1; i\_r++)

{

Y[i\_r] = (double\*)malloc(sizeof(double)\*(N+1));

Ynew[i\_r] =(double\*)malloc(sizeof(double)\*(N+1));

Lambda[i\_r] =(double\*)malloc(sizeof(double)\*(N+1));

}

for(i\_r=0;i\_r<N+1;i\_r++){

for(j\_r=0;j\_r<=N\_p+1;j\_r++){

i=i\_r;

j=myid\*N\_p+j\_r;

Lambda[i\_r][j\_r]=L1;

if (iter== 0){

Y[i\_r][j\_r]=u0(i\*h\_x,j\*h\_y,0);

}

else{

Y[i\_r][j\_r]=Ynew[i\_r][j\_r];

}

}

}

for(j\_r=1;j\_r<=N\_p;j\_r++){

double \*myF = (double\*)malloc(sizeof(double)\*(N+1));

double \*LL = (double\*)malloc(sizeof(double)\*(N+1));

int id;

for(id=0;id<N+1;id++){

LL[id]=Lambda[id][j\_r];

}

j=myid\*N\_p+j\_r;

myF[0]=u\_x0(x\_0,j\*h\_y,tau\*iter);

myF[N]=u\_xN(x\_L,j\*h\_y,tau\*iter);

int i;

for(i=1;i<N;i++){

double F = Y[i][j\_r]/tau+((Lambda[i][j\_r+1]+

Lambda[i][j\_r])/2 \* (Y[i][j\_r+1]-Y[i][j\_r]) -

(Lambda[i][j\_r-1]+Lambda[i][j\_r])/2 \* (Y[i][j\_r]-Y[i][j\_r-1]))/(2\*h\_y\*h\_y);

myF[i]=F;

}

double \* Y\_p=(double\*)malloc(sizeof(double)\*(N+1));

double \* Y\_qs=(double\*)malloc(sizeof(double)\*(N\_q+2));

double \* Y\_qr =(double\*)malloc(sizeof(double)\*(N\_q+2));

Y\_p = vektor\_progonki (myF,LL, h\_x, N);

int i\_s;

int j\_s;

for(i\_s=0;i\_s<proc;i\_s++){

for(j\_s=0;j\_s<=N\_q+1;j\_s++){

ii=i\_s\*N\_q+j\_s;

j=myid\*N\_p+j\_r;

Y\_qs[j\_s]=Y\_p[ii];

if (myid == i\_s) {

Ynew[j][j\_s]=Y\_qs[j\_s];

}

else{

jj=j\_r\*(N\_p+3)+j\_s;

MPI\_Send(Y\_qs, N\_q+2, MPI\_DOUBLE, i\_s, jj, MPI\_COMM\_WORLD);

MPI\_Recv(Y\_qr, N\_q+2, MPI\_DOUBLE, i\_s, jj,MPI\_COMM\_WORLD, &status);

ja=i\_s\*N\_p+j\_r;

Ynew[ja][j\_s]=Y\_qr[j\_s];

}

}

}

}

MPI\_Barrier(MPI\_COMM\_WORLD);

int j\_s;

for(j\_s=0;j\_s<=N\_q+1;j\_s++){

jj=myid\*N\_q+j\_s;

Ynew[0][j\_s]=u\_y0(jj\*h\_x,y\_0,tau\*iter);

Ynew[N][j\_s]=u\_yN(jj\*h\_x,y\_L,tau\*iter);

}

for(i\_r=0;i\_r<N+1;i\_r++){

for(j\_r=0;j\_r<=N\_p+1;j\_r++){

Y[i\_r][j\_r]=Ynew[i\_r][j\_r];

}

}

for(j\_r=1;j\_r<=N\_p;j\_r++){

double \*myF = (double\*)malloc(sizeof(double)\*(N+1));

double \*LL = (double\*)malloc(sizeof(double)\*(N+1));

int m;

for(m=0;m<N+1;m++){

LL[m]=Lambda[m][j\_r];

}

i=myid\*N\_p+j\_r;

myF[0]=u\_y0(i\*h\_x,y\_0,tau\*iter);

myF[N]=u\_yN(i\*h\_x,y\_L,tau\*iter);

int i;

for(i=1;i<N;i++){

double F = Y[i][j\_r]/tau+((Lambda[i][j\_r+1]+

Lambda[i][j\_r])/2 \* (Y[i][j\_r+1]-Y[i][j\_r]) -

(Lambda[i][j\_r-1]+Lambda[i][j\_r])/2 \* (Y[i][j\_r]-Y[i][j\_r-1]))/(2\*h\_x\*h\_x);

myF[i]=F;

}

double \* Y\_p=(double\*)malloc(sizeof(double)\*(N+1));

double \* Y\_qs=(double\*)malloc(sizeof(double)\*(N\_q+2));

double \* Y\_qr =(double\*)malloc(sizeof(double)\*(N\_q+2));

Y\_p = vektor\_progonki (myF,LL, h\_x, N);

int i\_s;

int j\_s;

for(i\_s=0;i\_s<proc;i\_s++){

for(j\_s=0;j\_s<=N\_q+1;j\_s++){

ii=i\_s\*N\_q+j\_s;

j=myid\*N\_p+j\_r;

Y\_qs[j\_s]=Y\_p[ii];

if (myid == i\_s) {

Ynew[j][j\_s]=Y\_qs[j\_s];

}

else{

jj=j\_r\*(N\_p+3)+j\_s;

MPI\_Send(Y\_qs, N\_q+2, MPI\_DOUBLE, i\_s, jj,MPI\_COMM\_WORLD);

MPI\_Recv(Y\_qr, N\_q+2, MPI\_DOUBLE, i\_s, jj,MPI\_COMM\_WORLD, &status);

ja=i\_s\*N\_p+j\_r;

Ynew[ja][j\_s]=Y\_qr[j\_s];

}

}

}

}

MPI\_Barrier(MPI\_COMM\_WORLD);

jj=0;

for(j\_s=0;j\_s<=N\_q+1;j\_s++){

jj=myid\*N\_q+j\_s;

Ynew[0][j\_s]=u\_x0(x\_0,jj\*h\_y,tau\*iter);

Ynew[N][j\_s]=u\_xN(x\_L,jj\*h\_y,tau\*iter);

}

iter++;

}

MPI\_Barrier(MPI\_COMM\_WORLD);

t2 = MPI\_Wtime();

printf("All time in current proccess is = %f\n", t2-t1);

MPI\_Finalize();

return 0;

}

# ПРИЛОЖЕНИЕ Б

Параллельная реализация на языке C решения двумерного нелинейного уравнения теплопроводности методом продольно–поперечной прогонки с улучшенной локальностью

#include <mpi.h>

#include <math.h>

#include <stdio.h>

#include <stdlib.h>

#define Lambda 0.001

#define Lambda\_2 0.00001

#define x\_0 0.0

#define x\_L 1.0

#define y\_0 0.0

#define y\_L 0.5

#define T 3.0

#define tile 144

#define N 1441

#define h\_x (x\_L-x\_0)/N

#define h\_y (y\_L-y\_0)/N

#define tau 0.1

double u(double x, double y, double t){

return exp(Lambda\*t+(x+y)/sqrt(2.)); }

double u0(double x, double y,double t){

return exp((x+y)/sqrt(2.)); }

double u\_x0(double x,double y, double t){

return exp(Lambda\*t+y/sqrt(2.)); }

double u\_xN(double x,double y, double t){

return exp(Lambda\*t+(y+x\_L)/sqrt(2.));}

double u\_y0(double x,double y, double t){

return exp(Lambda\*t+x/sqrt(2.)); }

double u\_yN(double x,double y, double t){

return exp(Lambda\*t+(x+y\_L)/sqrt(2.));}

void print\_Y\_0(double \*\*Y, int horizontal, int vertical, int rank)

{

printf("Y in process = %d\n",rank);

for(int i=0;i<horizontal;i++)

{

for(int j=0;j<vertical;j++)

{

printf("%.5f\t", Y[i][j]);

}

printf("\n");

}

}

void print\_Y\_1(double \*\*Y, int horizontal, int vertical, int rank)

{

printf("Y in process = %d\n",rank);

for(int i=0;i<horizontal;i++)

{

for(int j=0;j<vertical;j++)

{

printf("%.5f\t", Y[i][j]);

}

printf("\n");

}

}

void print\_Y\_3(double \*\*Y, int horizontal, int vertical, int rank)

{

printf("Y in process = %d\n",rank);

for(int i=0;i<horizontal;i++)

{

for(int j=0;j<vertical;j++)

{

printf("%.5f\t", Y[i][j]);

}

printf("\n");

}

}

int main(int argc, char \*\*argv )

{

int previousProcess;

int nextProcess;

int lastNumberOfProcess;

int rank, size;

int iter;

double t0,t1,t2;

MPI\_Status status;

MPI\_Init(&argc, &argv);

MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);

t0 = MPI\_Wtime();

previousProcess = rank - 1;

nextProcess = rank + 1;

lastNumberOfProcess = size - 1;

int r1 ;

r1 = (N - 1) / size;

int r2;

r2 = (N - 1) / tile;

iter = 0;

double \*\*Y;

Y = (double\*\*)malloc(sizeof(double\*)\*(r1 + 2));

int inner\_j;

for(inner\_j = 0 ; inner\_j < r1 + 2; inner\_j++) {

Y[inner\_j] = (double\*)malloc(sizeof(double)\*(N+1));

}

int inner\_i;

for(inner\_i = 0; inner\_i < r1+2; inner\_i++) {

for(inner\_j = 0; inner\_j < N+1; inner\_j++) {

Y[inner\_i][inner\_j] = u0((inner\_i+r1\*rank)\*h\_x,inner\_j\*h\_y,0);

}

}

iter = 0;

while(iter\*tau<=T)

{

if(rank == 0){

for(inner\_j = 0; inner\_j < N+1; inner\_j++) {

Y[0][inner\_j] = u\_x0(x\_0,inner\_j\*h\_y,tau\*iter);

}

}

if(rank == lastNumberOfProcess) {

for(inner\_j = 0; inner\_j < N+1; inner\_j++) {

Y[r1 + 1][inner\_j] = u\_xN(x\_L,inner\_j\*h\_y,tau\*iter);

}

}

for(inner\_i = 0; inner\_i < r1+2; inner\_i++) {

Y[inner\_i][0] = u\_y0((inner\_i + rank\*r1)\*h\_x,y\_0,tau\*iter);

}

for(inner\_i = 0; inner\_i < r1+2; inner\_i++) {

Y[inner\_i][N] = u\_yN((inner\_i + rank\*r1)\*h\_x,y\_L,tau\*iter);

}

double\*\* alpha;

alpha = (double\*\*)malloc(sizeof(double\*)\*(r1 + 2));

int i;

for(i=0;i<r1+2;i++) {

alpha[i]=(double\*)malloc(sizeof(double)\*(r2\*tile));

}

double\*\* betta ;

betta = (double\*\*)malloc(sizeof(double\*)\*(r1 + 2));

for(i=0;i<r1+2;i++) {

betta [i]=(double\*)malloc(sizeof(double)\*(r2\*tile));

}

int IteratorByTile;

for(IteratorByTile = 0;IteratorByTile < tile; IteratorByTile++)

{

double \*firstAlpha = (double\*)malloc(sizeof(double)\*(r2));

double \*firstBetta = (double\*)malloc(sizeof(double)\*(r2));

if(rank != 0)

{

MPI\_Recv(firstAlpha, r2, MPI\_DOUBLE, previousProcess, IteratorByTile, MPI\_COMM\_WORLD, &status);

MPI\_Recv(firstBetta, r2, MPI\_DOUBLE, previousProcess, IteratorByTile, MPI\_COMM\_WORLD, &status);

int j;

for(j=0;j<r2;j++) {

alpha[0][j+r2\*IteratorByTile] = firstAlpha[j];

betta[0][j+r2\*IteratorByTile] = firstBetta[j];

}

}

else {

int j;

for(j=0;j<r2;j++) {

alpha[0][j+r2\*IteratorByTile] = 0;

betta[0][j+r2\*IteratorByTile] = Y[0][j+r2\*IteratorByTile+1];

}

}

int j;

for(j=0;j<r2;j++) {

double A ;

A = -1\*(Lambda+Lambda\_2)/(4\*h\_x\*h\_x);

double B ;

B = -1\*(Lambda+Lambda\_2)/(4\*h\_x\*h\_x);

double C ;

C = 1/tau-A-B;

int i;

for(i=1;i<r1+2;i++) {

alpha[i][j+r2\*IteratorByTile] = -B / (C+A\*alpha[i-1][j+r2\*IteratorByTile]);

betta[i][j+r2\*IteratorByTile] = (Y[i][j+IteratorByTile\*r2+1]/tau +((Lambda+Lambda\_2)/2 \* (Y[i][j+IteratorByTile\*r2+2]-Y[i][j+IteratorByTile\*r2+1]) - (Lambda+Lambda\_2)/2 \* (Y[i][j+IteratorByTile\*r2+1]-Y[i][j+IteratorByTile\*r2]))/(2\*h\_y\*h\_y) - A \* betta[i-1][j+r2\*IteratorByTile])/(C + A \* alpha[i-1][j+r2\*IteratorByTile]);

}

}

if(rank != lastNumberOfProcess)

{

double \*lastAlpha = (double\*)malloc(sizeof(double)\*(r2));

int temp;

for(temp =0;temp<r2;temp++) {

lastAlpha[temp]=alpha[r1][temp+r2\*IteratorByTile];

}

double \*lastBetta= (double\*)malloc(sizeof(double)\*(r2));

for(temp =0;temp < r2;temp++) {

lastBetta[temp]=betta[r1][temp+r2\*IteratorByTile];

}

MPI\_Send(lastAlpha, r2, MPI\_DOUBLE, nextProcess, IteratorByTile, MPI\_COMM\_WORLD);

MPI\_Send(lastBetta, r2, MPI\_DOUBLE, nextProcess, IteratorByTile, MPI\_COMM\_WORLD);

}

}

for(IteratorByTile = 0;IteratorByTile < tile; IteratorByTile++)

{

double \*lastY= (double\*)malloc(sizeof(double)\*(r2));

if(rank != lastNumberOfProcess)

{

MPI\_Recv(lastY, r2, MPI\_DOUBLE, nextProcess, IteratorByTile+5000, MPI\_COMM\_WORLD, &status);

int j;

for(j=0;j<r2;j++) {

Y[r1+1][j + IteratorByTile\*r2+1] = lastY[j];

}

}

else {

int j;

for(j=0;j<r2;j++) {

Y[r1+1][j + IteratorByTile\*r2+1] =u\_xN(x\_L,(j+IteratorByTile\*r2+1)\*h\_y,tau\*iter);

}

}

int j;

int i;

for(j = 0; j < r2; j++) {

for(i = r1; i > -1; i--) {

Y[i][j+IteratorByTile\*r2+1] = alpha[i][j+IteratorByTile\*r2]\*Y[i+1][j+IteratorByTile\*r2+1]+

betta[i][j+IteratorByTile\*r2];

}

}

if(rank != 0)

{

double \*firstY = (double\*)malloc(sizeof(double)\*(r2));

int temp;

for(temp =0; temp < r2; temp++) {

firstY[temp] = Y[1][temp + IteratorByTile\*r2+1];

}

MPI\_Send(firstY, r2, MPI\_DOUBLE, previousProcess, IteratorByTile+5000, MPI\_COMM\_WORLD);

}

}

int i\_horizontal;

for(i\_horizontal = 1; i\_horizontal < r1+1 ; i\_horizontal++)

{

double \*alpha = (double\*)malloc(sizeof(double)\*(N));

double \*betta = (double\*)malloc(sizeof(double)\*(N));

alpha[0]=0;

betta[0]=u\_y0((i\_horizontal+r1\*rank)\*h\_x,y\_0,tau\*iter);

int j\_horizontal;

for(j\_horizontal = 1; j\_horizontal < N; j\_horizontal++) {

double A ;

A = -1\*(Lambda+Lambda\_2)/(4\*h\_y\*h\_y);

double B ;

B = -1\*(Lambda+Lambda\_2)/(4\*h\_y\*h\_y);

double C;

C = 1/tau-A-B;

alpha[j\_horizontal]=-B/(C+A\*alpha[j\_horizontal-1]);

betta[j\_horizontal]=(Y[i\_horizontal][j\_horizontal]/tau+

((Lambda+Lambda\_2)/2 \* (Y[i\_horizontal+1][j\_horizontal]-Y[i\_horizontal][j\_horizontal]) -(Lambda+Lambda\_2)/2 \* (Y[i\_horizontal][j\_horizontal]-Y[i\_horizontal-1][j\_horizontal]))/(2\*h\_x\*h\_x)-

A \* betta[j\_horizontal-1])/(C + A \* alpha[j\_horizontal-1]);

}

Y[i\_horizontal][N]=u\_yN((i\_horizontal+r1\*rank)\*h\_x,y\_L,tau\*iter);;

int j;

for(j=N-1;j>=0;j--)

{

Y[i\_horizontal][j] = alpha[j]\*Y[i\_horizontal][j+1]+betta[j];

}

}

iter++;

}

MPI\_Barrier(MPI\_COMM\_WORLD);

t2 = MPI\_Wtime();

printf("ALL Time is = %.5f \n", t2 - t0);

MPI\_Finalize();

return 0;

}