МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное
учреждение высшего образования
**«Южно-Уральский государственный университет
(национальный исследовательский университет)»**
Высшая школа электроники и компьютерных наук
Кафедра системного программирования

РАБОТА ПРОВЕРЕНА
Рецензент
к.т.н., доцент кафедры прикладной математики ЮУрГУ
_____Т.Ю. Оленчикова
“___”_____ 2018 г.

ДОПУСТИТЬ К ЗАЩИТЕ
Заведующий кафедрой,
д.ф.-м.н., профессор
_____Л.Б. Соколинский
«___»_____2018 г.

РАЗРАБОТКА Q-ЭФФЕКТИВНОЙ ПРОГРАММЫ ДЛЯ РЕШЕНИЯ СЛАУ МЕТОДОМ ГАУССА–ЗЕЙДЕЛЯ

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
ЮУрГУ – 02.03.02.2018.115-036.ВКР

Научный руководитель
к.ф.-м.н., доцент кафедры СП
_____В.Н. Алеева
Автор работы
студент группы КЭ-402
_____А.Д. Нечепоренко

Ученый секретарь
(нормоконтроллер)
_____О.Н. Иванова
«___»_____2018 г.

Челябинск-2018

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

АКТУАЛЬНОСТЬ ТЕМЫ ИССЛЕДОВАНИЯ

В настоящее время многоядерная архитектура используется в большинстве ЭВМ. Преимущество данной архитектуры заключается в том, что она позволяет исполнять алгоритмы параллельно.

Современный подход к распараллеливанию численных алгоритмов не подразумевает решения этой задачи в общем виде. В настоящее время нет общепринятых критериев для оценки ресурса параллелизма алгоритма и степени параллелизма программы не существует. Исследования в этих областях могут помочь решить эту проблему. Так, зная характеристики и ограничения вычислительной системы и ресурс распараллеливания алгоритма, можно получить самую параллельную реализацию алгоритма.

Одним из подходов для решения данной задачи является концепция Q-детерминанта [2, 4, 10, 13]. Программа, использующая полностью параллелизм алгоритма, в рамках концепции Q-детерминанта называется Q-эффективной.

ЦЕЛЬ И ЗАДАЧИ ИССЛЕДОВАНИЯ

Целью данной работы является разработка Q-эффективных программ для решения СЛАУ методом Гаусса–Зейделя для общей и распределенной памяти.

Для достижения поставленной цели было необходимо решить следующие задачи:

1) изучить метод проектирования параллельных программ на основе концепции Q-детерминанта;
2) изучить метод Гаусса–Зейделя для решения СЛАУ;
3) представить метод Гаусса–Зейделя для решения СЛАУ в виде Q-детерминанта;
4) описать Q-эффективную реализацию метода Гаусса–Зейделя;
5) изучить технологию OpenMP;
6) изучить технологию MPI;

7) разработать Q-эффективные программы для решения СЛАУ методом Гаусса–Зейделя для общей и распределенной памяти;

8) провести вычислительные эксперименты на суперкомпьютере «Торнадо ЮУрГУ» с целью получения динамических характеристик разработанных программ.

СТРУКТУРА И ОБЪЕМ РАБОТЫ

Выпускная квалификационная работа состоит из введения, пяти разделов, заключения, библиографии, приложений. Объем работы составляет 30 страниц, объем библиографии – 15 наименований, объем приложения 2 страницы.

СОДЕРЖАНИЕ РАБОТЫ

Первый раздел «Анализ предметной области» содержит описание концепции Q-детерминанта и технологий параллельного программирования.

Во втором разделе «Теоретическая часть» содержится описание алгоритма, Q-детерминанта и Q-эффективной реализации алгоритма.

Третий «Проектирование» включает функциональные и нефункциональные требования, варианты использования системы, общее описание архитектуры системы.

В четвертом разделе «Реализация системы» содержится описание реализации разрабатываемой системы.

В пятом разделе описано тестирование Q-эффективной программы и результаты вычислительных экспериментов, проведенных на суперкомпьютере «Торнадо ЮУрГУ».

В заключении представлены основные результаты выполненной работы.

Приложения содержат дополнительные материалы работы.

1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1. Концепция Q-детерминанта

Пусть α – алгоритм для решения алгоритмической проблемы $\overline{y} = F(N, B)$, где $N = \{n_1, \dots, n_k\}$ – множество параметров размерности проблемы или пустое множество, B – множество входных данных, $\overline{y} = (y_1, \dots, y_m)$ – множество входных данных. $\overline{N}$ – вектор $(\overline{n}_1, \dots, \overline{n}_k)$, где $\overline{n}_i$ – некоторое заданное значение параметра n_i. $\{\overline{N}\}$ – множество всевозможных векторов $\overline{N}$. Q – множество операций используемых алгоритмом α.

Любое однозначное отображение $w: \{\overline{N}\} \to V$, где V – множество всех выражений B над Q, называется безусловным Q-термом. Если при любом $\overline{N} \in \{\overline{N}\}$ и любой интерпретации переменных B $w(\overline{N})$ принимает значение логического типа, то w называется безусловным логическим Q-термом. Пусть $u_1, \dots, u_l$ – безусловные логические Q-термы, $w_1, \dots, w_l$ – безусловные Q-термы. Множество пар (u_i, w_i), где $i = 1, \dots, l$, обозначается $(\overline{u}, \overline{w}) = \{(u_i, w_i)\}_{i=1,\dots,l}$ и называется условным Q-термом длинны l. Счетное множество пар безусловных Q-термов $(\overline{u}, \overline{w}) = \{(u_i, w_i)\}_{i=1,2,\dots}$ называется условным бесконечным Q-термом, если $\{(u_i, w_i)\}_{i=1,\dots,l}$ является условным Q-термом для любого $l < \infty$. Если не имеет значения, является ли Q-терм безусловным, условным или условным бесконечным, то его можно назвать Q-термом. Под вычислением безусловного Q-терма w при интерпретации B следует понимать вычисления выражений $w(\overline{N})$ при некотором $\overline{N} \in \{\overline{N}\}$. Для вычисления при заданной интерпретации B и некотором $\overline{N} \in \{\overline{N}\}$ условного Q-терма $(\overline{u}, \overline{w}) = \{(u_i, w_i)\}_{i=1,\dots,l}$ необходимо найти такие $u_{i_0}(\overline{N}), w_{i_0}(\overline{N})$, что $u_{i_0}(\overline{N})$ принимает значение *true*, а значение $w_{i_0}(\overline{N})$ определено. В качестве значения $(\overline{u}, \overline{w})$ нужно взять $w_{i_0}(\overline{N})$. Если установлено, что выражение $u_{i_0}(\overline{N}), w_{i_0}(\overline{N})$ не существует, то значение $(\overline{u}, \overline{w})$ для данной интерпретации B и данного $\overline{N}$ не определено. Вычисление условного бесконечного Q-терма определяется аналогично.

Предположим, что I_1, I_2, I_3 – подмножества множества $I = (1, \dots, m)$ такие, что: одно или два из множеств $I_i (i = 1,2,3)$ могут быть пустыми, $I_1 \cup I_2 \cup I_3 = I$, $I_i \cap I_j = \emptyset (i \neq j; i,j = 1,2,3)$. Множество Q-термов $\{f_i\}_{i \in j}$ удовлетворяет условиям: $f_{i_1}(i_1 \in I_1)$ – безусловный Q-терм, $f_{i_1} = w^{i_1}$; $f_{i_2}(i_2 \in I_2)$ – условный Q-терм, $f_{i_2} = \left\{\left(u_j^{i_2}, w_j^{i_2}\right)\right\}_{j=1,\dots,l_{i_2}}$, l_{i_2} является вычислимой функцией параметров N; $f_{i_3}(i_3 \in I_3)$ – условный бесконечный Q-терм, , $f_{i_3} = \left\{\left(u_j^{i_3}, w_j^{i_3}\right)\right\}_{j=1,2,\dots}$. Если алгоритм α состоит в том, что для определения $y_i (i \in I)$ называется Q-детерминантом алгоритма α, а представление алгоритма в виде $y_i = f_i (i \in I)$ представлением в форме Q-детерминанта.

Реализацией алгоритма α, представленного в форме Q-детерминанта $y_i = f_i (i \in I)$, называется вычислением Q-термов $f_i (i \in I)$ при заданной интерпретации B и некотором $\overline{N} \in \{\overline{N}\}$. Если реализация такова, что выражения $W(\overline{N}) = \{w^{i_1}(\overline{N})(i \in I); u_j^{i_2}(\overline{N}), w_j^{i_2}(\overline{N})(i_2 \in I_2, j = 1, \dots, l_{i_2}); u_j^{i_3}(\overline{N}), w_j^{i_3}(\overline{N})(i_3 \in I_3, j = 1,2, \dots)\}$ вычисляются одновременно и при их вычислении операции выполняются по мере готовности, то реализация называется Q-эффективной. Если алгоритм допускает распараллеливание, то его Q-эффективная реализация полностью использует ресурс параллелизма алгоритма, поэтому является максимально параллельной реализаций алгоритма. Реализация алгоритма α называется выполнимой, если одновременно необходимо выполнять конечное число операций [3].

1.2. Проектирование параллельных программ на основе концепции Q-детерминанта

Подход к разработке программы, выполняющей Q-эффективную реализацию алгоритма, основан на следующих утверждениях:

1) Q-детерминант можно построить для любого численного алгоритма;

2) Q-детерминант позволяет описать Q-эффективную реализацию алгоритма;

3) если Q-эффективная реализация алгоритма является выполнимой, то можно разработать программу для ее выполнения.

Процесс разработки программы состоит из следующих этапов:

1) построение Q-детерминанта алгоритма;

2) описание Q-эффективной реализации алгоритма;

3) если Q-эффективная реализация выполнима, то для нее разрабатывается программа.

Разработанную программу будем называть Q-эффективной, а процесс ее разработки Q-эффективным программированием. Q-эффективная программа полностью использует ресурс параллелизма алгоритма, т.к. выполняет его Q-эффективную реализацию. В связи с этим она не допускает дальнейшего распараллеливания.

Q-эффективная программа – программа, выполняющая Q-эффективную реализацию алгоритма на определенной вычислительной архитектуре. В данной работе рассматриваются следующие особенности архитектуры: наличие общей либо распределенной памяти, а также количество независимых вычислительных узлов.

Если готовы к выполнению несколько операций цепочки, то они выполняются по схеме сдваивания, которая ускоряет вычисления, изображенной на рис. 1.

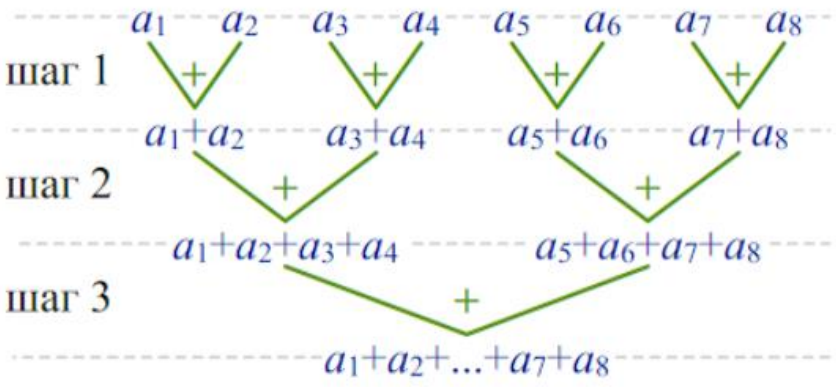

Рис. 1. Схема сдваивания

В соответствии с Q-эффективной реализацией при вычислении каждого из Q-термов на каждом такте работы программы выполняются все готовые к выполнению операции.

1.3. OpenMP

OpenMP – это стандарт программного интерфейса приложения для параллельных компьютерных систем с общей памятью. Данный стандарт реализован для языков программирования Fortran и C/C++ и состоит из набора директив для компилятора, библиотек функция и набора переменных окружения [1, 6, 9, 11].

Параллельная программ, написанная при помощи OpenMP, состоит из цепочек последовательных и параллельных участков кода. Параллельные участки кода создаются посредствам директив. Например, директива *#pragma omp parallel for* перед циклом указывает на то, что цикл будет разделен на дочерние потоки. Количество потоков можно указывать в коде. Если не указывать количество, то будет использовано максимально возможное количество потоков.

Иллюстрация отличий последовательной и параллельной программы изображен на рис. 2.

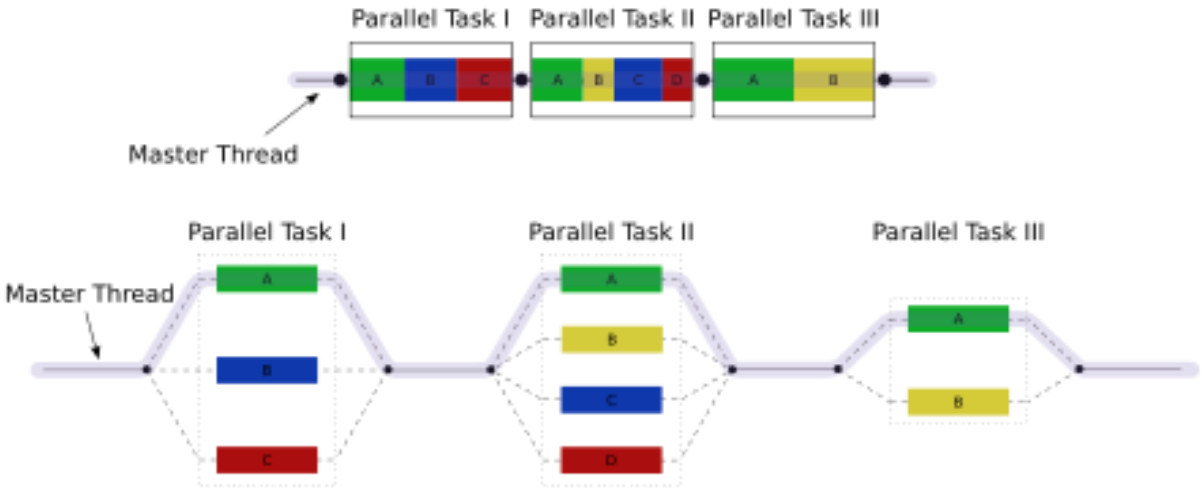

Рис. 2. Пример различия последовательной и распараллеленной программы при помощи OpenMP

На рисунке наглядно показана разница между последовательным выполнением и параллельным. Основной потом «Master Thread» разделяется по мере выполнения на несколько потоков «Parallel Task». При этом в каждом параллельном регионе может быть разное количество дочерних потоков.

1.4. MPI

MPI является наиболее распространенным стандартом интерфейса обмена данными в параллельном программировании, существуют его реализации для большого числа компьютерных платформ. Используется при разработке программ для кластеров и суперкомпьютеров. Основным средством коммуникации между процессами в MPI является передача сообщений друг другу.

Технология MPI используется для написания параллельных программ с распределенной памятью.

Под параллельной программой в рамках MPI понимается множество одновременно выполняемых процессов. Процессы могут выполняться на разных процессорах, но на одном процессоре могут располагаться и несколько процессов (в этом случае их исполнение осуществляется в режиме разделения времени). В предельном случае для выполнения параллельной программы может использоваться один процессор – как правило, такой способ применяется для начальной проверки правильности параллельной программы [5].

Поскольку для коммуникации между процессами в MPI используется передача сообщений, на кластерных системах для написания эффективных параллельных программ чаще всего используется гибридное программирование MPI+OpenMP, т.е. для пересылки данных между узлами и взаимодействия между ними используется технология MPI, а для более быстрого взаимодействия между потоками на самих узлах (каждый из которых является системой с общей памятью) используется технология OpenMP [7, 8, 15].

Вывод по главе 1

В главе рассмотрена концепция Q-детерминанта и способ проектирования Q-эффективных программ. Также рассмотрены технологии параллельного программирования OpenMP и MPI. OpenMP применяется для написания программ с использованием общей памяти. MPI применяется для написания программ с использованием распределенной памяти.

2. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

2.1. Метод Гаусса–Зейделя

Пусть дана система $A\bar{x} = \bar{b}$, где

$$A = \begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{pmatrix},$$

$$\bar{b} = \begin{pmatrix} b_1 \\ \vdots \\ b_n \end{pmatrix}.$$

Чтобы пояснить суть метода, перепишем задачу в виде:

$$\begin{cases} a_{11}x_1 = -a_{12}x_2 - a_{13}x_3 - \ldots - a_{1n}x_n + -b_1 \\ a_{21}x_1 + a_{22}x_2 = -a_{23}x_3 - \ldots - a_{2n}x_n + -b_2 \\ \ldots \\ a_{(n-1)1}x_1 + a_{(n-1)2}x_2 + \ldots + a_{(n-1)(n-1)}x_{n-1} = -a_{(n-1)n}x_n + b_{n-1} \\ a_{n1}x_1 + a_{n2}x_2 + \ldots + a_{n(n-1)}x_{n-1} + a_{nn}x_n = b_n \end{cases}$$

Здесь в j-м уравнении мы перенесли в правую часть все члены, содержащие x_i, для i>j. Эта запись может быть представлена:

$$(L + D)\bar{x} = -U\bar{x} + \bar{b},$$

где в принятых обозначениях D означает матрицу, у которой на главной диагонали стоят соответствующие элементы матрицы A, а все остальные элементы – нули.

При этом матрицы U и L содержат верхнюю и нижнюю треугольные части A, на главной диагонали которых нули.

Итерационный процесс в методе Гаусса–Зейделя [12] строится по формуле:

$$(L + D)\,\overline{x}^{(k+1)} = -U\overline{x}^{(k)} + \overline{b},\ k=0,1,2,\ldots$$

после выбора соответствующего начального приближения $\overline{x}^{(0)}$.

Метод Гаусса–Зейделя можно рассматривать как модификацию метода Якоби.

Основная идея модификации состоит в том, что в методе Гаусса–Зейделя новые значения $\overline{x}^{(i)}$ используются по мере получения, в то время как в методе Якоби они не используются до следующей итерации:

$$\begin{cases} x_1^{(k+1)} = c_{12}x_2^{(k)} + c_{13}x_3^{(k)} + \ldots + c_{1n}x_n^{(k)} + d_1 \\ x_2^{(k+1)} = c_{21}x_1^{(k)} + c_{23}x_3^{(k)} + \ldots + c_{2n}x_n^{(k)} + d_2 \\ \ldots \\ x_n^{(k+1)} = c_{n1}x_1^{(k)} + c_{n2}x_2^{(k)} + \ldots + c_{n(n-1)}x_{n-1}^{(k+1)} + d_n, \end{cases}$$

где:

$$c_{ij} = -\frac{a_{ij}}{a_{ii}},\ d_i = \frac{b_i}{a_{ii}},\ i = 1, \ldots, n.$$

Таким образом, i-тая компонента $(k+1)$-го приближения вычисляется по формуле:

$$x_i^{(k+1)} = \sum_{j=1}^{i-1} c_{ij}x_j^{(k+1)} + \sum_{j=i+1}^{n} c_{ij}x_j^{(k)} + d_i,\ i = 1, \ldots, n.$$

Условие окончания итерационного процесса метода Гаусса–Зейделя решения СЛАУ в момент достижении точности величины ε имеет следующий вид:

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

2.2. Проектирование параллельной программы для Q-эффективной реализации алгоритма

Теперь построим Q-детерминант метода Гаусса–Зейделя для решения СЛАУ и опишем Q-эффективную реализацию данного метода.

Этап 1

Количество Q-термов в Q-детерминанте алгоритма зависит от количества выходных данных.

В случае с методом Гаусса–Зейделя решения системы линейных алгебраических уравнений вектор x состоит из n элементов. Это значит, что Q-детерминант метода Гаусса–Зейделя состоит из n Q-термов, все Q-термы являются условными бесконечными, так как вычисление выходных данных зависит от точности вычислений ε.

Представление метода Гаусса–Зейделя в форме Q-детерминанта имеет вид:

$$x_i = \left\{ (\|x^1 - x^0\| < \varepsilon, x_i^1), \ldots, \left(\left\|x^k - x^{k-1}\right\| < \varepsilon, x_i^k\right), \ldots \right\} (i = 1, \ldots, n).$$

Этап 2

Для упрощения описания Q-эффективной реализации введем обозначения:

$$u^l = \left|\left|x^1 - x^{l-1}\right|\right| < \varepsilon \ (l = 1,2,\dots).$$

Тогда Q-детерминант имеет вид:

$$x_j = \left\{\left(u^1, x_j^1\right), \dots, \left(u^k, x_j^k\right), \dots\right\} (i = 1, \dots, n)$$

В соответствии с определением Q-эффективной реализации все безусловные Q-термы $\{u^l, x_j^l\}(i = 1, \dots, n; l = 1,2\dots)$ должны вычисляться одновременно.

Сначала должны быть вычислены одновременно следующие Q-термы:

$$x_i^1 (i = 1, \dots, n),$$

т.к. их операции готовы к выполнению.

После этого нужно вычислить одновременно следующие безусловные Q-термы:

$$u^1 \ , x_1^2 (i = 1, \dots, n).$$

Если u^1 имеет значение true, то вычисления заканчиваются, а решением системы линейных уравнений является:

$$x_i = x_i^1 (i = 1, \dots, n).$$

Если вычисление будет продолжаться, то Q-термы:

$$u^k, x_i^{k+1} (i = 1, \dots n)$$

будут вычисляться одновременно при любом значении $k \geq 2$.

Если значение безусловного Q-терма u^k является истинным, то выполнение Q-эффективной реализации алгоритма заканчиваются, а решением исходной системы линейных алгебраических уравнений выглядит следующим образом:

$$x_i = x_i^k (i = 1, \dots, n).$$

Из этого следует, что Q-эффективная реализация метода Гаусса–Зейделя выполнима.

Вывод по главе 2

Был рассмотрен метод Гаусса–Зейделя для решения СЛАУ. Кроме того, был построен Q-детерминант для метода Гаусса–Зейделя для решения СЛАУ и описана его Q-эффективная реализация.

3. ПРОЕКТИРОВАНИЕ

Для выполнения поставленной задачи было решено разработать программную систему, позволяющую произвести вычисление СЛАУ методом Гаусса–Зейделя последовательно, с использованием общей или распределенной памяти.

3.1. Требования и варианты использования системы

Разрабатываемая система должна удовлетворять следующим функциональным требованиям:

1) должна принимать на вход размерность СЛАУ;
2) должна решать СЛАУ методом Гаусса–Зейделя;
3) должна оценивать время решения СЛАУ методом Гаусса–Зейделя;
4) должна генерировать СЛАУ любой размерности.

Поскольку технологии OpenMP и MPI реализуются на ограниченном наборе языков программирования, система должна удовлетворять следующим нефункциональным требованиям: система должна быть реализована на языке C++.

Для проектирования системы был использован язык графического описания для объектного моделирования UML. Диаграмма вариантов использования представлена на рис. 3.

Актеры, взаимодействующие с системой

Пользователь использует систему для вычисления СЛАУ методом Гаусса–Зейделя.

Краткое описание вариантов использования

Пользователь может:

1) задать размерность СЛАУ и выбрать способ решения СЛАУ методом Гаусса–Зейделя.

2) Решить СЛАУ методом Гаусса–Зейделя последовательно, решить СЛАУ методом Гаусса–Зейделя с использованием общей памяти или с использованием распределённой памяти.

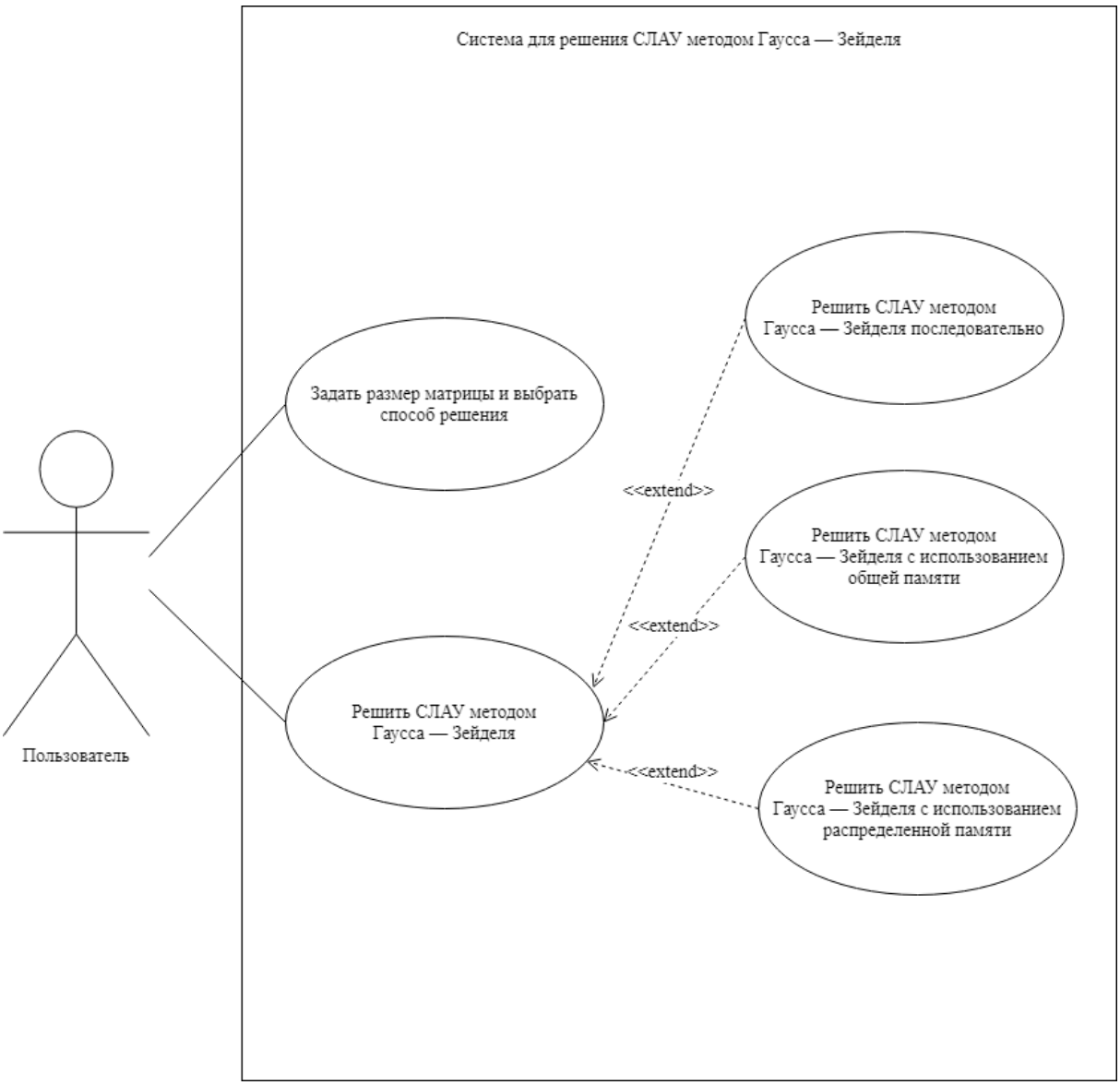

Рис. 3. Диаграмма вариантов использования

3.2. Общее описание архитектуры системы

На рис. 4 представлена диаграмма компонентов системы. Система содержит 4 компонента: MainScript, Simple, OpenMP, MPI.

1) *MainScript* – главный компонент системы, обеспечивает выбор других компонентов системы и считывание размера матрицы.

2) *Simple* – компонент, решающий СЛАУ методом Гаусса–Зейделя последовательно.

3) *OpenMP* – компонент, решающий СЛАУ методом Гаусса–Зейделя параллельно с использованием общей памяти.

4) *MPI* – компонент, решающий СЛАУ методом Гаусса–Зейделя параллельно с использованием распределенной памяти.

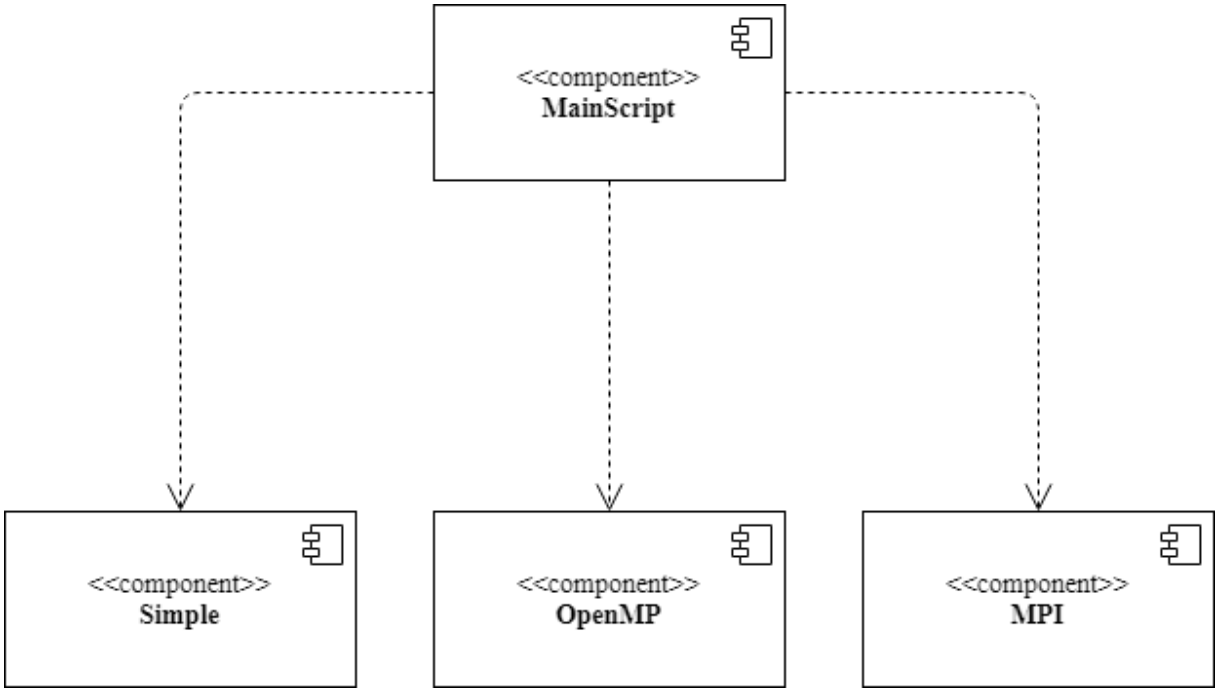

Рис. 4. Диаграмма компонентов

Вывод по главе 3

В ходе написания раздела были сформулированы функциональные и нефункциональные требования к системе, созданы диаграмма вариантов использования и диаграмма компонентов. Спроектирована система для решения СЛАУ методом Гаусса–Зейделя, состоящая из 4 компонентов.

4. РЕАЛИЗАЦИЯ СИСТЕМЫ

4.1. Главный модуль системы

Главный модуль системы представлен в виде скрипта для командной оболочки bash ОС Linux. Реализация этого компонента приведен на листинге 1.

Компонент принимает на вход два параметра:

– N: размерность матрицы;

– mode: запускаемый модуль.

Листинг 1. Главный модуль

```
#!/bin/bash
N=5000
mode=0
while getopts "n:m:" opt
do
      case $opt in
      n) N=$OPTARG ;;
      m) mode=$OPTARG ;;

      *) echo "invalid arguments" ;;
      esac
done
case $mode in
0) simple -N $N
1) openMP -N $N
2) MPI -N $N
*) echo "Invalid mode." ;;
esac
```

4.2. Последовательная реализация

В последовательную реализацию входит модуль *Simple*.

В модуле *Simple* производится расчет СЛАУ методом Гаусса–Зейделя последовательно.

Реализация модуля приведена на листинге 2.

Листинг 2. Последовательная реализация

```
main()
{
      int N;
```

```
      double runtime;
      float** a;

      bool converge(double *xk, double* xkp)
      {
          for (int i = 0; i < n; i++)
              if (fabs(xk[i]-xkp[i]) >= eps)
                  return false;
          return true;
      }

      a = new float *[N];
      for (int i = 0; i < N; i++)
      {
            a[i] = new float[N + 1];
            for (int j = 0; j < N + 1; j++)
            {
                  a[i][j] = (rand() % 100 - 50)*0.5;
            }
      }

      float *x = new float[N];
      float *p = new float[N];
      for (int i = 0; i < N; i++)
            x[i] = 0;
      double start_time = clock() / (float)CLOCKS_PER_SEC;
      do

      {
            for(int i = 0; i < n; i++)
            {
                  double var = 0;
                  for(int j = 0; j < n; j++)
                        if(j != i) var += (a[i][j]*x[j]);
                  p[i] = x[i];
                  x[i]=(b[i] - var)/a[i][i];
            }
      }
      while(!converge(x,p));
      double end_time = clock() / (float)CLOCKS_PER_SEC;
      runtime = end_time - start_time;
      runtime end_time - start_time << endl;
};
```

4.3. Параллельная реализация

В параллельную реализацию входят модули *OpenMP* и *MPI.*

В модуле *OpenMP* производится расчет СЛАУ методом Гаусса–Зейделя параллельно с использованием общей памяти с применением технологии OpenMP.

Листинг модуля приведен в приложении 1.

В модуле *MPI* производится расчет СЛАУ методом Гаусса–Зейделя параллельно с использованием распределенной памяти с применением технологии MPI+OpenMP.

Вывод по главе 4

В ходе написания главы был реализован скрип запуска программы, в котором выбирается размер матрицы и способ решения СЛАУ методом Гаусса–Зейделя. Были описаны последовательная и параллельные реализации решения СЛАУ методом Гаусса–Зейделя.

5. ТЕСТИРОВАНИЕ И ЭКСПЕРИМЕНТЫ

5.1. Тестирование и характеристики системы

Для проверки корректности работы разработанных программ было проведено функциональное тестирование.

Функциональное тестирование – это тестирование ПО в целях проверки реализуемости функциональных требований, то есть способности ПО в определенных условиях решать задачи, нужные пользователям. Функциональные требования определяют, что именно делает ПО, какие задачи оно решает.

В рамках данной работы были разработаны программы с единственной функциональной возможностью: решить СЛАУ методом Гаусса–Зейделя. Для тестирования работы алгоритмов в программы были добавлены функции ввода данных с клавиатуры или из конфигурационного файла и вывода полученного решения в результирующий файл.

Также были найдены СЛАУ малых размерностей с заранее известными решениями. И сгенерированы единичные матрицы для проверки работы системы на больших размерностях систем. Найти СЛАУ больших размерностей с известными решениями не удалось.

После верного решения достаточно большого количества тестовых СЛАУ, работу программ можно считать корректной и для других СЛАУ.

Для вычислений был выбран суперкомпьютер «Торнадо ЮУрГУ», входящий в рейтинги Green500 и СНГ ТОР50 [14]. Характеристики системы представлены в таблицах 1-2.

Табл. 1. Характеристики вычислительного узла «Торнадо ЮУрГУ»

Модель процессора	Intel Xeon X5680
Количество ядер/потоков	12/24
Частота ядер, GHz	3,33
Количество процессоров	2

Табл. 2. Технические характеристики «Торнадо ЮУрГУ»

Число вычислительных узлов/процессорных ядер	480/5760
Тип процессора:	Intel Xeon X5680 (Gulftown, 6 ядер по 3.33 GHz) – 960 шт.
Оперативная память:	16,9 TB
Пиковая производительность комплекса	473.6 TFlops
Производительность комплекса на тесте LINPACK:	288.2 TFlops
Операционная система:	Linux CentOS 6.2

5.2. Эксперименты

Одной из основных задач работы является получение данных о выполнении спроектированных программ на суперкомпьютере «Торнадо ЮУрГУ».

Т.к. показатель чистого времени работы программы t на разных параллельных вычислительных системах будет зависеть от характеристик конкретной системы, а для исследования нужно было получить данные в более общем виде, было решено использовать такие показатели как:

1) S_p – ускорение работы параллельной программы ($S_p = \frac{T_1}{T_p}$, где T_p – время работы параллельной программы, T_1 – время работы последовательной программы);

2) E_p – эффективность параллельной программы ($E_p = \frac{S_p}{p}$, где p – количество процессорных ядер).

Каждое значение времени, использованное для вычисления ускорения и эффективности Q-эффективных программ, являлось средним временем на основе 20 экспериментов (для повышения точности экспериментальных данных).

5.3. Результаты экспериментов

На рис. 7 и рис. 8 представлены графики ускорения и эффективности для общей памяти при использовании матрицы размером 45000.

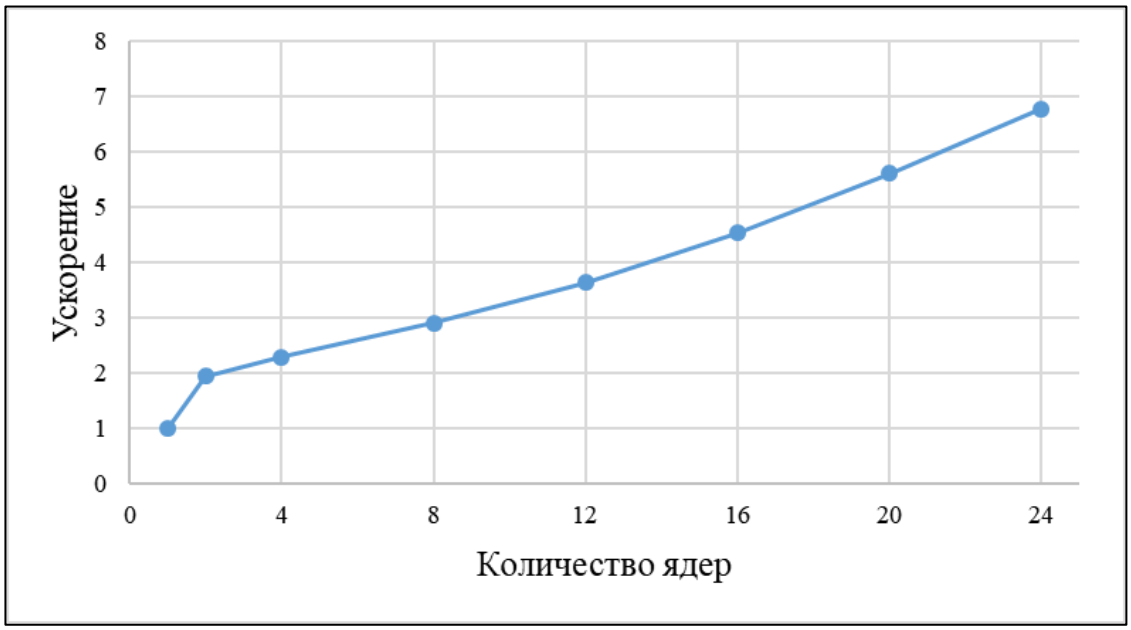

Рис. 7. Ускорение для Q-эффективной программы в случае общей памяти

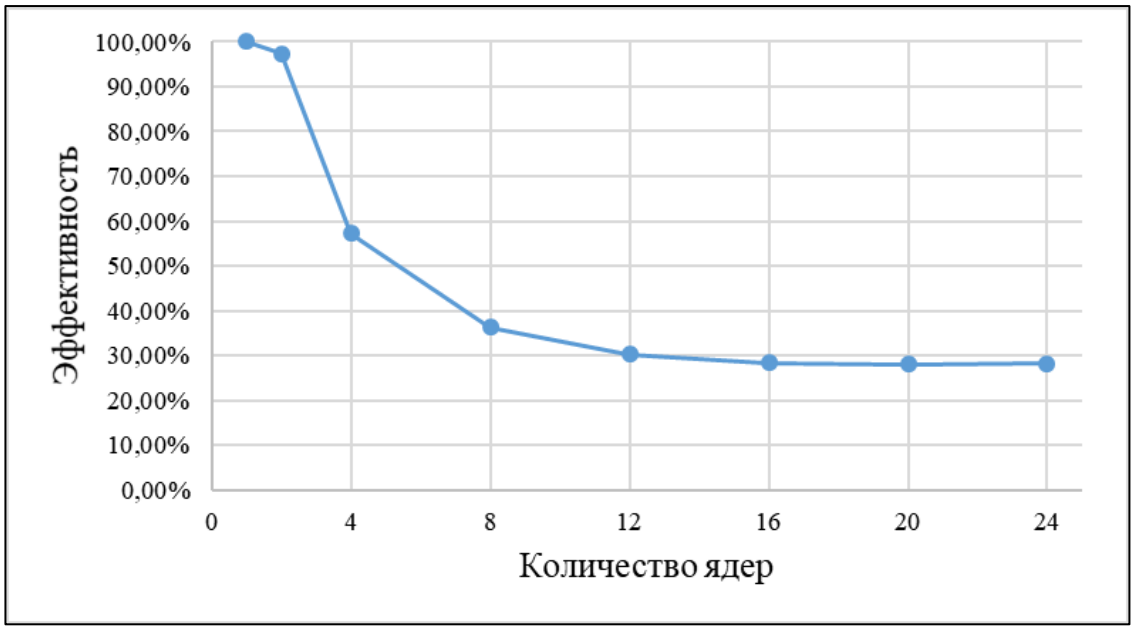

Рис. 8. Эффективность для Q-эффективной программы в случае общей памяти

На рис. 9 и рис. 10 представлены графики ускорения и эффективности для распределённой памяти при использовании матрицы размером 45000.

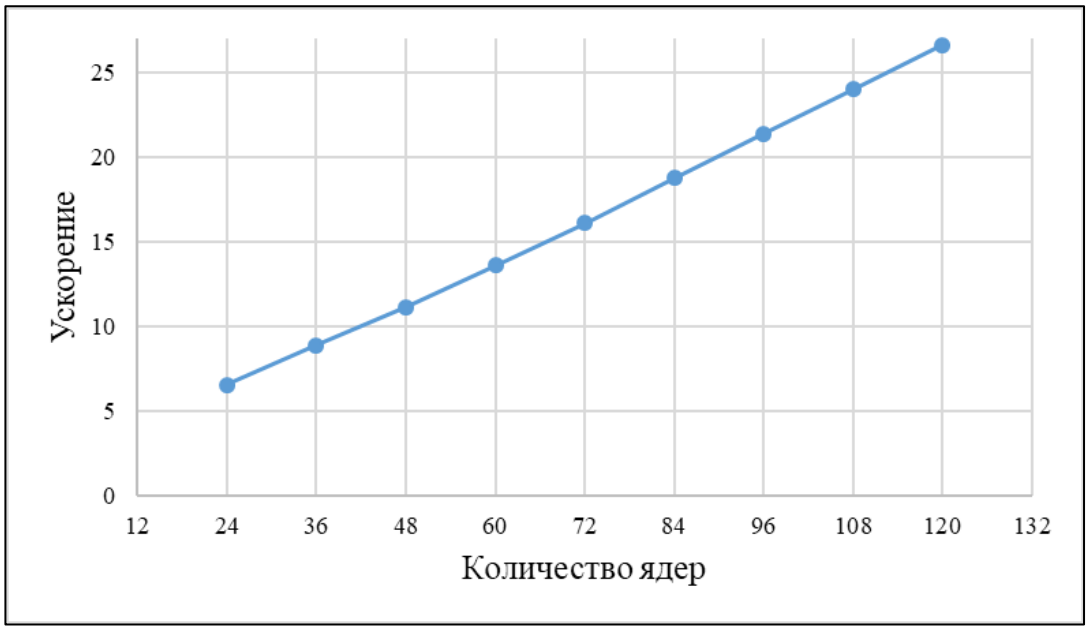

Рис. 9. Ускорение для Q-эффективной программы в случае распределенной памяти

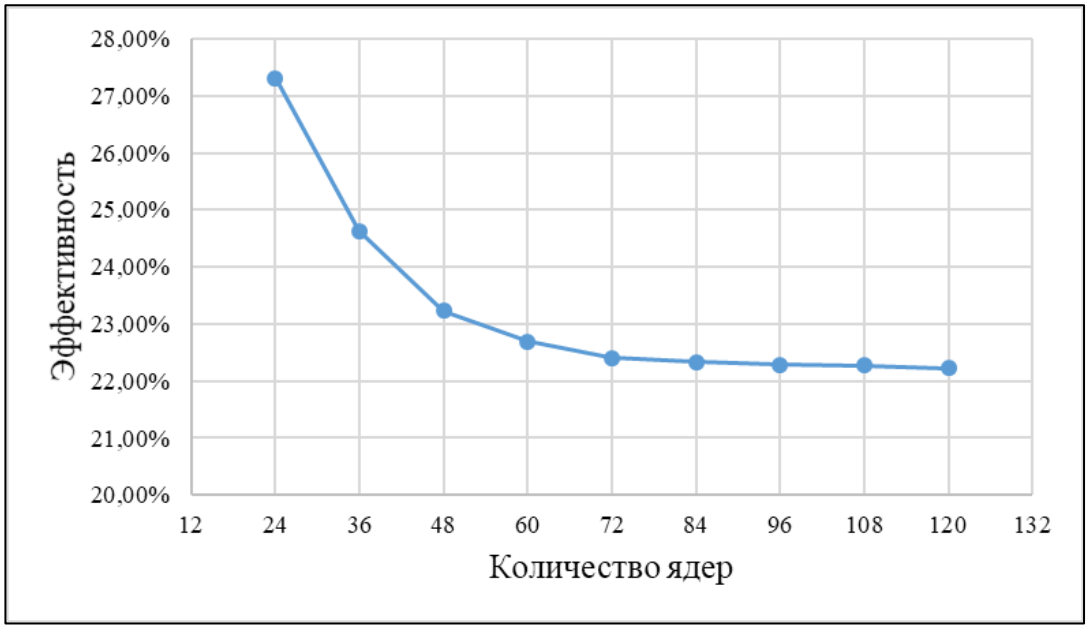

Рис. 10. Эффективность для Q-эффективной программы в случае распределенной памяти

Исходя из графиков, можно сделать вывод, что реализованные Q-эффективные программы являются масштабируемыми, т.к. при увеличении количества вычислительных ядер график эффективности стремится к постоянному значению, а ускорение растет.

Вывод по главе 5

Были проведены тесты, по результатам которых был сделан вывод, что реализованные Q-эффективные программы являются масштабируемыми, так как эффективность Q-эффективной программы в случае общей памяти стремится к 28 %, а в случае распределённой памяти к 22 %, а ускорение растет.

ЗАКЛЮЧЕНИЕ

В ходе выполнения выпускной квалификационной работы были разработаны Q-эффективные программы для решения СЛАУ методом Гаусса–Зейделя для общей и распределенной памяти.

Были решены следующие задачи.

1. Изучен метод проектирования параллельных программ на основе концепции Q-детерминанта.
2. Изучен метод Гаусса–Зейделя для решения СЛАУ.
3. Представлен метод Гаусса–Зейделя для решения СЛАУ в виде Q-детерминанта.
4. Описана Q-эффективная реализации метода Гаусса–Зейделя;
5. Изучена технология OpenMP.
6. Изучена технология MPI.
7. Разработаны Q-эффективные программы для решения СЛАУ методом Гаусса–Зейделя для общей и распределенной памяти.
8. Проведены вычислительные эксперименты на суперкомпьютере «Торнадо ЮУрГУ» с целью получения динамических характеристик разработанной программы.

Следует заметить, что Q-эффективная программа полностью использует ресурс параллелизма алгоритма, т.к. выполняет его Q-эффективную реализацию, поэтому дальнейшее распараллеливание такой программы невозможно.

Данная работа выполнена при поддержке Российского Фонда Фундаментальных Исследований, грант № 17-07-00865.

СПИСОК ЛИТЕРАТУРЫ

1. Chandra R. Parallel Programming in OpenMP Kindle Edition. / R. Chandra, Menon R., Dadum L., Kohr D. – UK: Morgan Kaufmann, 2001. – 163 p.

2. Алеева В.Н. Анализ параллельных численных алгоритмов: Препринт № 590 // Новосибирск: ВЦ СО АН СССР, 1985. – 23 с.

3. Алеева В.Н. Расширенная модель концепции Q-детерминанта и ее применение для реализации ресурса параллелизма численных алгоритмов. В.Н. Алеева, Н.В. Валькевич, Ю.С. Лаптева, Д.Е. Тарасов. // «Современные проблемы математического моделирования, обработки изображений и параллельных вычислений 2017» (СПММОИиПВ-2017): труды Международной научной конференции (пос. Дивноморское, 4–11 сентября 2017 г.). Том I; Донской гос. техн. ун-т. – Ростов-на-Дону: ООО «ДГТУ-Принт», 2017. – С. 23–32.

4. Алеева В.Н., Сулейманов Д.Э., Шарабура И.С. «Разработка программной системы QStudio для получения максимально быстрой реализации алгоритма». // Параллельные вычислительные технологии (ПаВТ’2015): труды международной научной конференции (31 марта – 2 апреля 2015 г., г. Екатеринбург), Издательский центр ЮУрГУ, Челябинск, 2015. – С. 523.

5. Антонов А.С. Параллельное программирование с использованием технологии MPI. Учебное пособие. – М.: Изд-во МГУ, 2004. – 71 с.

6. Антонов А.С. Параллельное программирование с использованием технологии OpenMP. Учебное пособие. – М.: Изд-во МГУ, 2009. – 77 с.

7. Антонов А.С. Технологии параллельного программирования MPI и OpenMP. Учеб. пособие. – М.: Издательство Московского университета, 2012. – 344 с.

8. Бахтин В.А. Гибридная модель параллельного программирования DVM/OpenMP: диссертация кандидата физико-математических наук:

05.13.11 // Институт прикладной математики им. М.В. Келдыша РАН. – Москва, 2008. – 122 с.

9. Борзунов С.В. Практикум по параллельному программированию. СПб.: Изд-во БВХ-Петербург, 2017. – 265 с.

10. Игнатьев С.В. Определение ресурса параллелизма алгоритмов на базе концепции Q-детерминанта: Вып. квалиф. работа магистра прикладной математики и информатики: 010500.68 /Южно-Уральский государственный университет. – Челябинск, 2009. – 75 с.

11. Левин М.П. Параллельное программирование с использование OpenMP. Учебное пособие. – М.: БИНОМ. Лаборатория знаний, Интернет-Университет Информационных Технологий (ИНТУИТ), 2008. – 120 с.

12. Осиновская, И.В. Численные методы решения алгебраических уравнений и их систем, электрон. учеб. пособие. [Электронный ресурс] URL: http://ssau.ru/files/education/uch_posob/Численные%20методы-Осиновская%20ИВ.pdf (дата обращения: 03.03.2018).

13. Свирихин Д.И. Построение эффективной реализации алгоритма на основе концепции Q-детерминанта, вып. квалиф. работа магистра: 010300.68 – Челябинск: Южно-Уральский государственный университет, 2013. – 47 с.

14. Суперкомпьютер «Торнадо ЮУрГУ» [Электронный ресурс]. URL: http://supercomputer.susu.ru/computers/tornado/ (дата обращения: 15.04.2018).

15. Шамов Е.А. Технологии достижения параллелизма MPI, CUDA, OpenMP и моделирование динамики электронного потока в скрещенных полях с применением гибрида технологий MPI и OpenMP. / Е.А. Шамов, В.С. Лукьянов, Д.Н. Жариков, Д.С. Попов // Известия ВолгГТУ. 2010. – № 6. С. – 33–37.

ПРИЛОЖЕНИЕ

Листинг 1.

```
main
{
	input = new float *[N];
	A = new float *[N];
	B = new float [N];
	for (int i = 0; i < N; i++)
	{
		input[i] = new float[N + 1];
		A[i] = new float[N];
		for (int j = 0; j < N + 1; j++)
		{
			input[i][j] = (rand() % 100 - 50)*0.5;
		}
	}
}
{
	for (int j = 0; j < N - 1; j++) {
		float max = Abs(input[j][j]);
		int maxN = j;
		for (int i = j + 1; i < N; i++) {
			if (Abs(input[i][j]) > max) {
				maxN = i;
				max = Abs(input[i][j]);
			}
		}
		for (int k = 0; k < N + 1; k++) {
			float temp = input[j][k];
			input[j][k] = input[maxN][k];
			input[maxN][k] = temp;
		}
	}
		for (int i = 0; i < N; i++) {
		for (int j = 0; j < N; j++) {
			A[i][j] = input[i][j];
		}
		B[i] = input[i][N];
	}
		float *x = new float[N];
	float *p = new float[N];
	for (int i = 0; i < N; i++)
		x[i] = 0;
	double start_time = clock() / (float)CLOCKS_PER_SEC;
	do
	{
		for (int i = 0; i < N; i++)
			p[i] = x[i];
		for (int i = 0; i < N; i++)
		{
			float sum = B[i] / A[i][i];
			int half = (N - (i + 1)) / 2;
#pragma omp parallel for reduction(+:sum)
			for (int j = i + 1; j < half; j++) {
				x[i] += p[j] * A[i][j] / A[i][i] + p[N - j] *
			A[i][N - j] / A[i][i];
			}
			if ((N - (i + 1)) % 2 != 0) {
				sum += p[half] * A[i][half] / A[i][i];
			}
```

```
                  x[i] = sum;
            }
            for (int i = 0; i < N - 1; i++)
            {
#pragma omp parallel for
            for (int j = i + 1; j < N; j++)
                  x[j] += x[i] * A[j][i] / A[j][j];
            }
      } while (!converge(x, p));
      double end_time = clock() / (float)CLOCKS_PER_SEC;
      runtime = end_time – start_time;
}
      runtime end_time – start_time << endl;
};
```