МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ 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 ПМиП
_____ Т.Ю. Оленчикова
“___”_____ 2020 г.

ДОПУСТИТЬ К ЗАЩИТЕ

Заведующий кафедрой,
д.ф.-м.н., профессор
_____ Л.Б. Соколинский
“___”_____ 2020 г.

Разработка программного обеспечения для формирования Q-детерминантов численных алгоритмов с Q-термами большой длины

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
ЮУрГУ – 02.03.02.2020. 308-029.ВКР

Научный руководитель,
к.ф.-м.н., доцент кафедры СП
_____ В.Н. Алеева

Автор работы,
студент группы КЭ-402
_____ М.П. Соколов

Ученый секретарь
(нормоконтролер)
_____ И.Д. Володченко
“___”_____ 2020 г.

Челябинск-2020

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ 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

УТВЕРЖДАЮ
Зав. кафедрой СП
_____ Л.Б. Соколинский
09.02.2020

ЗАДАНИЕ
на выполнение выпускной квалификационной работы бакалавра
студенту группы КЭ-402
Соколову Михаилу Павловичу,
обучающемуся по направлению
02.03.02 «Фундаментальная информатика и информационные технологии»

1. **Тема работы** (утверждена приказом ректора от 24.04.2020 № 627)

Разработка программного обеспечения для формирования Q-детерминантов численных алгоритмов с Q-термами большой длины.

2. **Срок сдачи студентом законченной работы:** 05.06.2020.

3. **Исходные данные к работе**

3.1. Подсистема для формирования Q-детерминантов численных алгоритмов программной системы Q-система.

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

3.3. Aleeva V., Bogatyreva E., Skleznev A., Sokolov M., Shuppa A. Software Q-system for the Research of the Resource of Numerical Algorithms Parallelism. In: Voevodin V., Sobolev S. (eds) Supercomputing. RuSCDays 2019. Communications in Computer and Information Science, vol 1129, pp. 641–652, 2019. Springer, Cham.

4. **Перечень подлежащих разработке вопросов**

4.1. Исследование концепции Q-детерминанта.

4.2. Разработка программного обеспечения для формирования Q-детерминантов численных алгоритмов с Q-термами большой длины.

4.3. Использование разработанного ПО.

5. Дата выдачи задания: 08.02.2020.

Научный руководитель
к.ф.-м.н., доцент кафедры СП В.Н. Алеева

Задание принял к исполнению М.П. Соколов

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Параллельные вычисления – способ организации компьютерных вычислений, при котором программы разрабатываются, как набор взаимодействующих вычислительных процессов, работающих асинхронно и при этом одновременно. Разработки в области параллельного программирования важны, потому что параллельное программирование помогает более эффективно использовать имеющиеся ресурсы вычислительных систем. Концепция Q-детерминанта может решить проблему повышения эффективности выполнения программ на параллельных вычислительных системах.

Использование ресурса внутреннего параллелизма численных алгоритмов с применением концепции Q-детерминанта позволяет достигнуть максимального распараллеливания любого алгоритма [1]. С помощью подхода, описанного в данной работе, можно для любого численного алгоритма, допускающего распараллеливание, определить его максимально параллельную реализацию. Помимо Q-эффективной реализации, Q-детерминант показывает все возможные реализации алгоритма и делает его прозрачным с точки зрения структуры.

Повышение эффективности за счет распараллеливания алгоритмов приведет к увеличению быстродействия параллельных вычислительных систем, в связи с чем актуальна проблема исследования внутреннего параллелизма алгоритмов [2].

Задача распараллеливания программ и задача распараллеливания алгоритмов связаны, но существенно различаются. Использование распараллеливания программ может оказаться не правильным решением задачи достижения максимальной эффективности выполнения, т.к. программа может не содержать всех реализаций выполнения алгоритмов, в том числе самой параллельной. Поэтому целесообразно использовать для распараллеливания структуру алгоритма [1].

Несмотря на ориентированность концепции Q-детерминанта на теоретическую модель вычислительной системы с бесконечными ресурсами, полученные результаты являются основой для автоматизированного выполнения наиболее эффективной реализации алгоритма на практике [1].

На рисунке 1 представлен пример Q-эффективной реализации алгоритма.

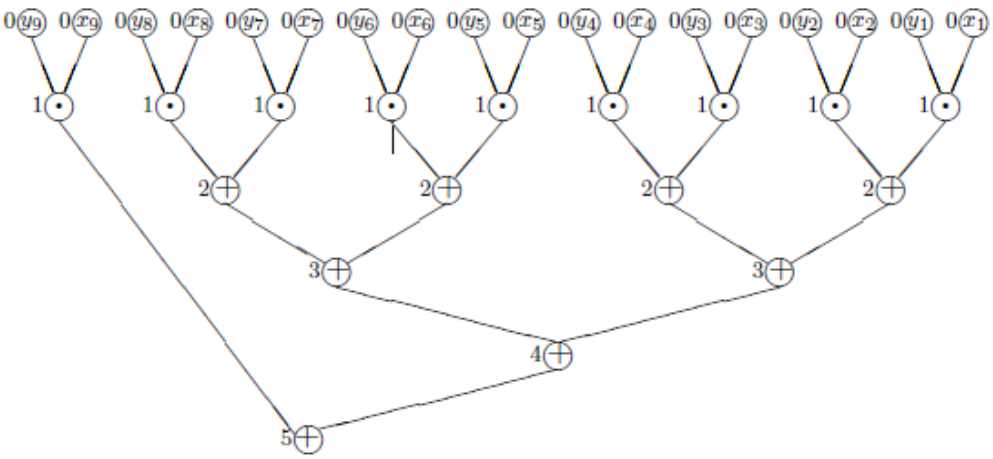

Рис. 1. Граф плана выполнения Q-эффективной реализации алгоритма скалярного произведения векторов размерности 9

Обзор литературы

Распараллеливанию алгоритмов посвящено много работ. Важным и развитым направлением является изучение параллельной структуры программ с целью их реализации на параллельных вычислительных системах. Одними из значимых работ этого направления являются [3, 4]. Среди работ по синтезу параллельных программ необходимо отметить [5], в которой описан метод синтеза параллельных программ и последующие работы, использующие этот метод. С помощью программной реализации алгоритмов проводятся исследования их ресурса распараллеливания [6].

Помимо этого проводятся исследования возможности повышения быстродействия параллельных программ за счет применения различных методов, использующих специфику алгоритмов и архитектуры вычислительных систем. Примерами таких работ являются [7, 8].

В работах [9, 10] приведено описание программной системы, которая дает возможность для любого численного алгоритма оценить такие характеристики параллельной сложности, как высота и ширина алгоритма, и из множества численных алгоритмов, решающих одну и ту же алгоритмическую проблему, выбрать алгоритм с лучшим ресурсом внутреннего параллелизма. Частью данной программной системы и является модернизируемая в рамках данной работы программа.

Таким образом, Q-детерминант делает алгоритм прозрачным с точки зрения структуры и реализации, он показывает все его реализации, в том числе Q-эффективную, позволяет выполнять алгоритмы на реальных вычислительных системах быстрее.

Цель и задачи исследования

Ранее разработанная подсистема [11] для генерации представления численных алгоритмов в форме Q-детерминанта позволяет формировать Q-детерминант алгоритма по соответствующей ему блок-схеме, которую задает пользователь в виде входного файла.

При завершении работы программного обеспечения формируется документ, содержащий представление алгоритма в форме Q-детерминанта, состоящего из Q-термов. Количество Q-термов в Q-детерминанте равняется количеству выходных данных алгоритма. При этом имеется ограничение на длину Q-термов: возникает связанное с переполнением памяти исключение в случае, если Q-термы имеют достаточно большую длину. В данной работе такие Q-термы будем называть Q-термами большой длины. Аналогично, Q-термы, которые программа формирует, будем называть Q-термами небольшой длины. Соответственно, невозможно с помощью данной программной системы получить представления некоторых алгоритмов, содержащих Q-термы большой длины, в форме Q-детерминанта. Для того чтобы снять ограничение на длину Q-термов, необходимо существенно модернизировать имеющееся программное обеспечение.

Целью данной работы является разработка программного обеспечения для формирования Q-детерминантов численных алгоритмов с Q-термами большой длины.

Для достижения этой цели необходимо решить следующие задачи:

1) изучить концепцию Q-детерминанта;

2) создать проект модернизации имеющегося программного обеспечения для генерации представлений численных алгоритмов в форме Q-детерминанта;

3) разработать ПО для генерации представлений численных алгоритмов в форме Q-детерминанта, не имеющее ограничений на длину Q-термов;

4) протестировать разработанное ПО;

5) использовать разработанное ПО для формирования Q-детерминантов алгоритмов, содержащих Q-термы большой длины.

Структура и объем работ

Курсовая работа состоит из введения, пяти разделов, заключения, библиографии. Объем работы составляет 32 страницы, объем библиографии – 15 наименований.

Содержание работы

Первый раздел «Концепция Q-детерминанта» описывает теоретические сведения о концепции Q-детерминанта.

Второй раздел «Проектирование программного обеспечения» содержит постановку задачи и требования к программе.

Третий раздел «Реализация программного обеспечения» включает описание разработки программы.

Четвертый раздел «Тестирование программного обеспечения» содержит результаты тестирования.

Пятый раздел «Эксплуатация» описывает опыт эксплуатации программной системы для обработки Q-детерминантов, содержащих Q-термы большой длины.

1. КОНЦЕПЦИЯ Q-ДЕТЕРМИНАНТА

1.1. Основные понятия

Концепция Q-детерминанта дает возможность исследовать ресурс внутреннего параллелизма алгоритма, а именно: получать все возможные реализации алгоритма, в том числе, Q-эффективную.

В рамках данной концепции Q-эффективной называется реализация алгоритма, которая в полной мере использует ресурс параллелизма алгоритма. Такая реализация также носит название максимально быстрой реализации алгоритма, так как все операции выполняются тогда, когда они готовы к выполнению. К тому же, представление алгоритма в форме Q-детерминанта позволяет проводить анализ характеристик ресурса параллелизма алгоритма – высота и ширина, которые характеризуют число тактов работы и число процессоров вычислительной системы соответственно, необходимых для выполнения Q-эффективной реализации.

Одним из главных понятий концепции Q-детерминанта является представление алгоритма в форме Q-детерминанта. В основе данной концепции лежит преобразование последовательности действий алгоритма в Q-термы – выражения над множеством арифметических или логических переменных и множеством операций над этими переменными. Вычисление полученных Q-термов и есть реализация алгоритма, представленного в форме Q-детерминанта. При этом параллельной называется такая реализация, что две или более операций выполняются одновременно.

Для лучшего понимания работы дадим более подробное описание основных понятий концепции.

α – алгоритм, решающий алгоритмическую проблему (формула 1):

$$\bar{y} = F(N, B) \tag{1}$$

- $N = \{n_1, \dots, n_k\}$ – множество параметров размерности алгоритмической проблемы, которое может быть пустым множеством;

- B – множество входных данных алгоритма;

- $\bar{y} = (y_1, \dots, y_m)$ – множество выходных данных.

$\bar{N}$ – вектор $(\bar{n}_1, \dots, \bar{n}_k)$, где $\bar{n}_i$ – некоторое заданное значение параметра n_i $(i = 1, \dots, k)$, множество всевозможных векторов $\bar{N}$ обозначим $\{\bar{N}\}$. Множество операций, которые используются в алгоритме α обозначим Q.

Пусть V – множество всех выражений над B и Q. Любое однозначное отображение $w: \{\bar{N}\} \to V$ будет являться безусловным Q-термом. Если при любой интерпретации переменных B и при любом $\bar{N} \in \{\bar{N}\}$ выражение $w(\bar{N})$ над множеством B и Q будет принимать значение логического типа, то w назовем безусловным логическим Q-термом.

Пусть $u_1, \dots, u_l$ – безусловные логические Q-термы, а $w_1, \dots, w_l$ – безусловные Q-термы. Множество пар (u_i, w_i), где $i = 1, \dots, l$ будем обозначать (формула 2):

$$(\bar{u}, \bar{w}) = \{(u_i, w_i)\}_{i=1,\dots,l} \tag{2}$$

и называть условным Q-термом длины l. Счетное множество пар безусловных Q-термов $(\bar{u}, \bar{w}) = \{(u_i, w_i)\}_{i=1,2,\dots}$ называется условным бесконечным Q-термом, если $\{(u_i, w_i)\}_{i=1,\dots,l}$ является условным Q-термом для любого конечного l. Если не имеет значения, является Q-терм условным, безусловным или условным бесконечным, то будем называть его Q-термом.

Вычисление безусловного Q-терма w при интерпретации B будем считать как вычисление выражения $w(\bar{N})$ при некотором $\bar{N} \in \{\bar{N}\}$. При заданной интерпретации B и некотором $\bar{N} \in \{\bar{N}\}$, для вычисления условного Q-терма необходимо, вычисляя выражения $u_i(\bar{N}), w_i(\bar{N})$, найти такие $u_{i_0}(\bar{N}), w_{i_0}(\bar{N})$, что $u_{i_0}(\bar{N})$ принимает значение $true$, а значение $w_{i_0}(\bar{N})$ определено. При этом $w_{i_0}(\bar{N})$ необходимо взять в качестве значения $(\bar{u}, \bar{w})$. Если известно, что выражений $u_{i_0}(\bar{N}), w_{i_0}(\bar{N})$ не существует, то для дан-

ной интерпретации B и данного $\overline{N}$ значение $(\overline{u}, \overline{w})$ не определено. Условный бесконечный Q-терм вычисляется аналогичным образом.

Предположим, что I_1, I_2, I_3 – подмножества множества $I = (1, \dots, m)$ такие, что (формула 3):

$$I_1 \cup I_2 \cup I_3 = I, I_i \cap I_j = \emptyset \tag{3}$$

при этом из множеств I_1, I_2, I_3 одно или два могут быть пустыми. Рассмотрим такое множество Q-термов $\{f_i\}_{i \in I}$, что:

- $f_{i_1}(i_1 \in I_1)$ – 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, $f_{i_3} = \left\{\left(u_j^{i_3}, w_j^{i_3}\right)\right\}_{j=1,2,\dots}$.

Предположим, что алгоритм состоит в том, что для определения y_i $(i \in I)$ требуется вычислить Q-терм f_i. Тогда множество Q-термов f_i $(i \in I)$ будем называть Q-детерминантом алгоритма, а представление алгоритма в виде $y_i = f_i$ $(i \in I)$ представлением алгоритма в форме Q-детерминанта.

Вычисление Q-термов f_i $(i \in I)$ при заданной интерпретации B и некотором $\overline{N} \in \{\overline{N}\}$ является реализацией алгоритма, представленного в форме Q-детерминанта $y_i = f_i(i \in I)$.

В случае, когда при реализации алгоритма выражения вычисляются одновременно, и во время их вычисление операции выполняются по мере готовности, то эта реализация носит название Q-эффективной (формула 4):

$$W(\bar{N}) = \Big\{ w^{i_1}(\bar{N})(i_1 \in I_1);\ u_j^{i_2}(\bar{N}), w_j^{i_2}(\bar{N})\big(i_2 \in I_2, j = 1, \dots, l_{i_2}\big); u_j^{i_3}(\bar{N}), w_j^{i_3}(\bar{N}), (i_3 \in I_3, j = 1,2, \dots) \Big\} \quad (4)$$

С формальной точки зрения Q-эффективная реализация алгоритма является максимально быстрой.

В случае, когда реализация алгоритма подразумевает одновременное выполнение конечного числа операций, такая реализация называется выполнимой реализацией алгоритма. При этом есть алгоритмы, Q-эффективная реализация которых не является выполнимой.

Q-эффективная реализация имеет следующие характеристики параллельной сложности:

- $D_\alpha(\bar{N})$ – максимальное число уровней вложенности выражений (4), вычисляется по формуле (формула 5):

$$D_\alpha(\bar{N}) = \max_{w(\bar{N}) \in W(\bar{N})} T^{w(\bar{N})}; \quad (5)$$

- $P_\alpha(\bar{N})$ – максимальное количество операций всех уровней вложенности всех выражений (4), вычисляется по формуле (формула 6):

$$P_\alpha(\bar{N}) = \max_{1 \le r \le D_\alpha(\bar{N})} \sum_{w(\bar{N}) \in W(\bar{N})} O_r^{w(\bar{N})}, \quad (6)$$

где $O_r^{w(\bar{N})}$ – количество операций уровня вложенности r выражения $w(\bar{N})$.

$D_\alpha(\bar{N})$ характеризует время выполнения Q-эффективной реализации алгоритма, а $P_\alpha(\bar{N})$ количество процессоров, необходимое для выполнения Q-эффективной реализации алгоритма. $D_\alpha(\bar{N})$ будем называть высотой алгоритма, а $P_\alpha(\bar{N})$ его шириной.

У выражения и операций, входящих в него, имеются уровни вложенности. В качестве обозначения количества уровней вложенности выраже-

ния $w(\bar{N})$ используем $T^{w(\bar{N})}$. Выражение, которое получено из n выражений с применением $(n-1)$ раз одной из ассоциативных операций множества Q без использования скобок в качестве указания порядка их выполнения, будем называть цепочкой выражений длины n.

В процессе определения уровня вложенности выражения и его подвыражений порядок выполнения операций цепочки задается с помощью схемы сдваивания. Например, для вычисления цепочки $a_1 + a_2 + a_3 + a_4$ по схеме сдваивания сначала нужно вычислить $b_1 = a_1 + a_2$ *и* $b_2 = a_3 + a_4$, а затем $c = b_1 + b_2$.

1.2. Исследование ресурса параллелизма алгоритмов

Для исследования внутреннего ресурса параллелизма численных алгоритмов в рамках концепции Q-детерминанта были рассмотрены следующие методы:

1) метод построения Q-детерминанта алгоритма на основе его блок-схемы;

2) метод получения Q-эффективной реализации алгоритма на основе Q-детерминанта;

3) метод вычисления характеристик ресурса параллелизма реализации алгоритма;

4) метод сравнения характеристик параллельной сложности Q-эффективных реализаций алгоритмов.

Для лучшего понимания смысла и значения этих методов рассмотрим их более подробно.

Поскольку алгоритмы очень редко представлены в форме Q-детерминанта, необходимо было использовать общепринятое представление алгоритма, например, блок-схему. Анализ блок-схемы происходит сверху вниз, что позволяет последовательно обойти ее и следить за процессом построения Q-детерминанта. Генерация на основе блок-схемы представления алгоритма в форме Q-детерминанта, включающего в себя условные, без-

условные или условные бесконечные Q-термы имеет свои особенности в реализации.

Метод получения Q-эффективной реализации алгоритма на основе Q-детерминанта заключается в определении уровня вложенности операций выражений $w(\overline{N})$. Множество выражений $w(\overline{N})$ с вычисленными уровнями вложенности операций называется планом выполнения Q-эффективной реализации.

Метод вычисления характеристик ресурса параллелизма алгоритма заключается в том, что:

- для вычисления высоты алгоритма нужно найти максимальное значение уровня вложенности операций выражений, входящих в Q-детерминант;

- для вычисления ширины алгоритма нужно найти количество операций каждого из уровней вложенности всех выражений, входящих в Q-детерминант, а затем вычислить максимальное из них.

Метод сравнения характеристик ресурса параллелизма алгоритмов позволяет сравнить для одинаковых наборов значений параметров размерности характеристики ресурса параллелизма двух алгоритмов, решающих одну и ту же алгоритмическую проблему. Это дает возможность определить алгоритм с лучшей характеристикой.

Программная система Q-system основана на описанных методах. Всего Q-system состоит из трех подсистем:

- первая подсистема, предназначенная для формирования представления численных алгоритмов в форме Q-детерминанта;

- вторая подсистема, реализующая интерфейс между первой подсистемой и третьей подсистемами;

- третья подсистема, необходимая для вычисления ресурса параллелизма численных алгоритмов.

2. ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

В работе была спроектирована модернизация первой подсистемы программной системы Q-system, которая реализует построение Q-детерминанта алгоритма по соответствующей алгоритму блок-схеме.

В основе имеющегося программного обеспечения лежит алгоритм, формирующий набор строк из Q-термов, которые записываются в выходной файл. Если Q-детерминант обрабатываемого алгоритма содержит Q-термы большой длины, то возникает исключение, связанное с переполнением памяти. Для решения этой проблемы необходимо модернизировать ПО, чтобы было возможно обрабатывать любые численные алгоритмы.

Программное обеспечение, которое требуется модернизировать, написано на языке программирования C# на базе платформы .NET. Модернизация этого ПО должна осуществляться с использованием того же языка программирования.

2.1. Диаграмма вариантов использования

На диаграмме вариантов использования имеющегося программного обеспечения был выделен единственный актер – «Пользователь», в качестве которого может выступать любой человек. Диаграмма вариантов использования изображена на рисунке 2.

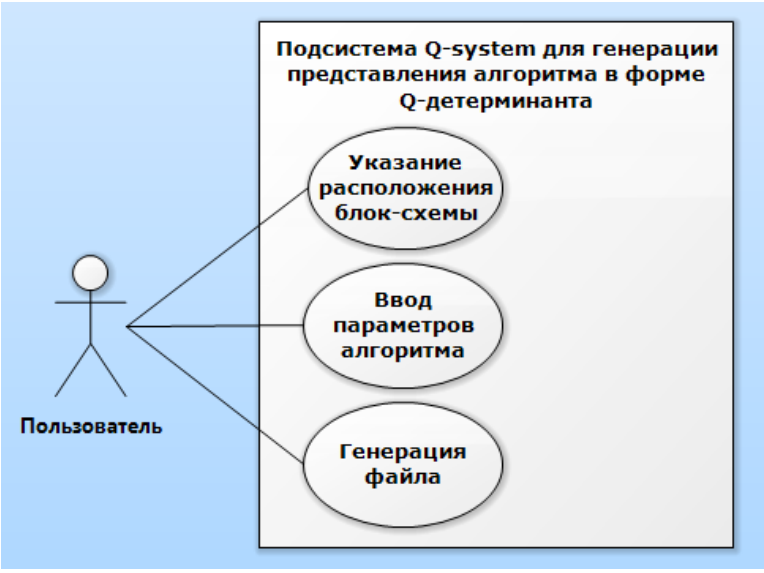

Рис. 2. Диаграмма вариантов использования

Для актера «Пользователь» имеются следующие прецеденты: «Указание расположения блок-схемы», «Ввод параметров алгоритма», «Генерация файла». В таблицах 1-3 представлены спецификации вариантов использования:

Табл. 1. Вариант использования «Указание расположения блок-схемы»

UseCase: Указание расположения блок-схемы
ID: 1
Аннотация: Предоставление пользователем информации о расположении и имени входного файла, содержащего блок-схему алгоритма
Главные актеры: Пользователь
Второстепенные актеры: Нет
Предусловия: Нет
Основной поток: 1) Пользователь вводит имя входного файла, содержащего блок-схему алгоритма.
Постусловия: Файл готов к обработке

Табл. 2. Вариант использования «Ввод параметров алгоритма»

UseCase: Ввод параметров алгоритма
ID: 2
Аннотация: Указание пользователем параметров алгоритма
Главные актеры: Пользователь
Второстепенные актеры: Нет
Предусловия: 1) Файл, содержащий блок-схему алгоритма, найден.
Основной поток: 1) Пользователь вводит название алгоритма, необходимого для формирования имени результирующего файла; 2) Пользователь вводит значения параметров размерности задачи; 3) Пользователь вводит значение количества итераций алгоритма.
Постусловия: Программа готова формировать файл.

Табл. 3. Вариант использования «Генерация файла»

UseCase: Генерация файла
ID: 3
Аннотация: Формирование представления алгоритма в форме Q-детерминанта
Главные актеры: Пользователь
Второстепенные актеры: Нет
Предусловия: 1) Пользователь указал необходимые параметры алгоритма.
Основной поток: 1) Программа формирует текстовый файл, содержащий представление алгоритма в форме Q-детерминанта.
Постусловия: Нет

2.2. Диаграмма классов

Суть модернизации заключается в том, что формируемые в ходе работы программы строковое переменные хранятся в автоматически создаваемых временных файлах.

Для обработки строк любой длины необходимо разработать новые классы: класс LargeDictionary для обработки коллекций, содержащих в качестве значений строки большой длины, и класс LargeString для обработки непосредственно больших строк. Для описания этих классов был создан новый файл BigData.cs, присоединенный к проекту.

Диаграмма классов проектируемого программного обеспечения представлена на рисунке 3.

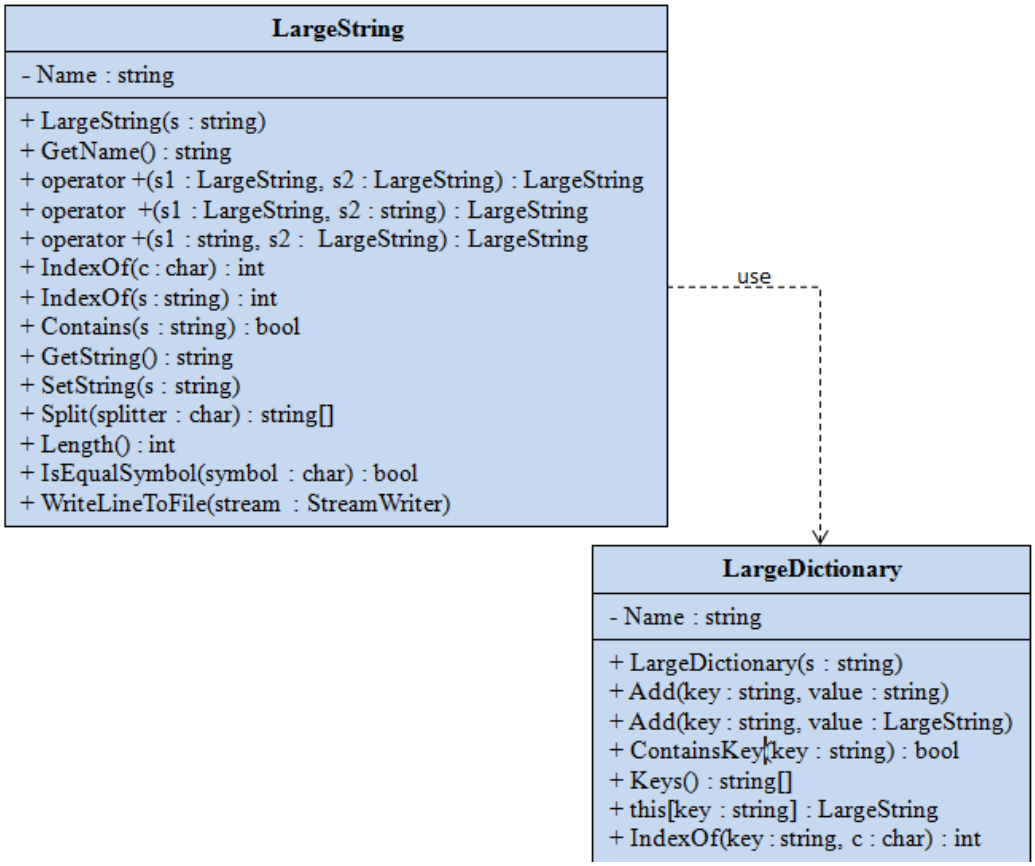

Рис. 3. Диаграмма классов

2.3. Класс *LargeString*

Суть класса LargeString заключается в том, что значения строк записываются в файлы, хранящиеся в папке LargeStrings корневого каталога. Операции над строками становятся операциями над содержимым файлов. При этом используются посимвольные чтение из файла и запись в файл.

Класс содержит следующие элементы:

- Поле *private string Name* – наименование строки;
- Конструктор *public LargeString(string s)* – генерирует для строки уникальное наименование (поле *Name*), создает файл с названием *Name* и содержанием *s* в папке *LargeString* корневого каталога;
- Метод *public string GetName()* – возвращает наименование строки *Name*;
- Метод *public static LargeString operator +(LargeString s1, LargeString s2)* – позволяет складывать значения двух строк-экземпляров дан-

ного класса с помощью оператора +. Сложение осуществляется посимвольной записью в результирующий файл сначала данных из файла первой строки, затем данных из файла второй строки;

- Метод *public static LargeString operator +(LargeString s1, string s2)* – также позволяет складывать строки, но второе слагаемое является стандартной строкой типа *string*;

- Метод *public static LargeString operator +(string s1, LargeString s2)* – аналогичен прошлым для сложения строк, но первое слагаемое является стандартной строкой типа *string*;

- Метод *public int IndexOf(char c)* – возвращает индекс символа *c* в строке;

- Метод *public int IndexOf(string s)* – возвращает индекс первого вхождения строки *s*;

- Метод *public bool Contains(string s)* – проверяет, содержит ли строка подстроку *s*;

- Метод *public string GetString()* – возвращает значение строки, т.е. содержимое файла *Name*;

- Метод *public void SetString(string s)* – изменяет значение строки на *s*, .т.е. перезаписывает содержимое файла *Name*;

- Метод *public string[] Split(char splitter)* – разделяет строку на подстроки по сепаратору *splitter*, возвращает массив подстрок;

- Метод *public int Length()* – возвращает длину строки, т.е. количество символов в файле *Name*;

- Метод *public bool IsEqualSymbol(char symbol)* – проверяет, состоит ли строка из единственного символа *symbol*;

- Метод *public void WriteLineToFile(StreamWriter stream)* - осуществляет посимвольную запись (посимвольно считывая содержимое файла *Name*) строки в выходной файл.

Пример реализации одного из методов класса – *IndexOf* – представлен на рисунке 4.

```
public int IndexOf(string s)
{
   int index = -1;
   int num = 0;
   string part = "";
   StreamReader reader = new StreamReader(Name);
   for (var i = 0; i < s.Length; i++)
   {
      part += (char)reader.Read();
   }
   while ((reader.Peek() >= 0) && (index == -1))
   {
      if (part == s)
         index = num;
      part = part.Substring(1);
      part += (char)reader.Read();
      num++;
   }
   reader.Close();
   if (part == s)
      index = num;
   return index;
}
```

Рис. 4. Реализация метода *IndexOf*

2.4. Класс *LargeDictionary*

Суть класса LargeDictionary заключается в том, что значения коллекции хранятся в файлах, названиями которых являются ключи соответствующих значений. Эти файлы хранятся в автоматически создаваемых временных папках, которые автоматически создаются для каждой коллекции. Операции над строками становятся операциями над содержимым файлов. При этом используется посимвольные чтение из файла и запись в файл.

Класс содержит следующие элементы:

- Поле *private string Name* – название коллекции;
- Конструктор *public LargeDictionary()* – создает папку в корневом каталоге программы;
- Метод *public void Add(string key, string value)* – добавляет в эту папку файл с названием *key*, содержащий текст *value*;
- Метод *public void Add(string key, LargeString value)* – аналогично предыдущему, но текст задан не строкой типа *string*, а экземпляром класса *LargeString*;

- Метод *public bool ContainsKey(string key)* – проверяет наличие в папке файла с именем *key*, т.е. имеется ли такой ключ;

- Метод *public string[] Keys()* – возвращает массив имен файлов в папке, т.е. массив ключей;

- Метод *public LargeString this[string key]* – позволяет обращаться к элементу по ключу *key* для чтения (возвращает содержимое файла *key* как экземпляр класса *LargeString*) и записи (записывает в файл *key* новый текст);

- Метод *public int IndexOf(string key, char c)* – возвращает индекс символа *c* в строке с ключом *key*.

Пример реализации одного из методов класса, позволяющего обращаться к элементу коллекции с помощью ключа, представлен на рисунке 5.

```
public LargeString this[string key]
{
   get
   {
      LargeString s = new LargeString("");
      File.Copy(Name + "\\" + key, s.GetName(), true);
      return s;
   }
   set
   {
      File.Copy(value.GetName(), Name + "\\" + key, true);
   }
}
```

Рис. 5. Реализация метода *this[string key]*

После создания новых классов для обработки строк большой длины, необходимо внести изменения в текст программы. Тип тех переменных строк, которые в ходе работы программы могли принимать критически большие значения, должны быть изменены со *string* на *LargeString*. Аналогичные изменения нужно осуществить с коллекциями: тип тех коллекций, значения которых могут принимать критически большие значения, необходимо изменить на *LargeDictionay*. Также, где это необходимо, нужно изменить синтаксис обработки этих строк и коллекций, тип которых был изменен, в соответствии с интерфейсом элементов новых классов.

3. РЕАЛИЗАЦИЯ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

3.1. Входные данные программы

Входные данные программы - текстовый файл в формате JSON, описывающий блок-схему алгоритма, название алгоритма, а также параметры размерности задачи и количества итераций алгоритма. На рисунке 6 отображена блок-схема алгоритма, реализующего метод Гаусса-Зейделя.


```
{"Vertices":[
{"Id":1,"Type":0,"Content":"Start"},{"Id":2,"Type":4,"Content":"A[n,n]"},
{"Id":3,"Type":4,"Content":"B[n]"},{"Id":4,"Type":4,"Content":"X0[n]"},
{"Id":5,"Type":4,"Content":"e"},{"Id":6,"Type":4,"Content":"iterations"},
{"Id":7,"Type":2,"Content":"it=1"},{"Id":8,"Type":2,"Content":"i=1"},
{"Id":9,"Type":3,"Content":"i<=n"},{"Id":10,"Type":2,"Content":"X(i)=X0(i)
"},{"Id":11,"Type":2,"Content":"i=i+1"},{"Id":12,"Type":2,"Content":"i=1"}
,{"Id":13,"Type":3,"Content":"i<=n"},{"Id":14,"Type":2,"Content":"newX(i)=
B(i)"},{"Id":15,"Type":2,"Content":"j=1"},{"Id":16,"Type":3,"Content":"j<=
n"},{"Id":17,"Type":3,"Content":"i!=j"},{"Id":18,"Type":3,"Content":"j<i"}
,{"Id":19,"Type":2,"Content":"D=A(i,j)*newX(j)"},{"Id":20,"Type":2,"Conten
t":"D=A(i,j)*X(j)"},{"Id":21,"Type":2,"Content":"newX(i)=newX(i)-
D"},{"Id":22,"Type":2,"Content":"j=j+1"},{"Id":23,"Type":2,"Content":"newX
(i)=newX(i)/A(i,i)"},{"Id":24,"Type":2,"Content":"i=i+1"},{"Id":25,"Type":
2,"Content":"i=1"},{"Id":26,"Type":2,"Content":"D=X(i)-
newX(i)"},{"Id":27,"Type":2,"Content":"norm=abs(D)"},{"Id":28,"Type":2,"Co
ntent":"X(i)=newX(i)"},{"Id":29,"Type":2,"Content":"i=i+1"},{"Id":30,"Type
":3,"Content":"i<=n"},{"Id":31,"Type":2,"Content":"D=X(i)-
newX(i)"},{"Id":32,"Type":2,"Content":"D=abs(D)"},{"Id":33,"Type":2,"Conte
nt":"norm=norm+D"},{"Id":34,"Type":3,"Content":"norm<e"},{"Id":35,"Type":2
,"Content":"it=it+1"},{"Id":36,"Type":3,"Content":"it<=iterations"},{"Id":
37,"Type":5,"Content":"X[n]"},{"Id":38,"Type":1,"Content":"End"}],
"Edges":[
      {"From":1,"To":2,"Type":2},   {"From":2,"To":3,"Type":2},
      {"From":3,"To":4,"Type":2},   {"From":4,"To":5,"Type":2},
      {"From":5,"To":6,"Type":2},   {"From":6,"To":7,"Type":2},
      {"From":7,"To":8,"Type":2},   {"From":8,"To":9,"Type":2},
      {"From":9,"To":10,"Type":1},  {"From":9,"To":12,"Type":0},
      {"From":10,"To":11,"Type":2}, {"From":11,"To":9,"Type":2},
      {"From":12,"To":13,"Type":2}, {"From":13,"To":14,"Type":1},
      {"From":13,"To":25,"Type":0}, {"From":14,"To":15,"Type":2},
      {"From":15,"To":16,"Type":2}, {"From":16,"To":17,"Type":1},
      {"From":16,"To":23,"Type":0}, {"From":17,"To":18,"Type":1},
      {"From":17,"To":22,"Type":0}, {"From":18,"To":19,"Type":1},
      {"From":18,"To":20,"Type":0}, {"From":19,"To":21,"Type":2},
      {"From":20,"To":21,"Type":2}, {"From":21,"To":22,"Type":2},
      {"From":22,"To":16,"Type":2}, {"From":23,"To":24,"Type":2},
      {"From":24,"To":13,"Type":2}, {"From":25,"To":26,"Type":2},
      {"From":26,"To":27,"Type":2}, {"From":27,"To":28,"Type":2},
      {"From":28,"To":29,"Type":2}, {"From":29,"To":30,"Type":2},
      {"From":30,"To":31,"Type":1}, {"From":30,"To":34,"Type":0},
      {"From":33,"To":28,"Type":2}, {"From":34,"To":37,"Type":1},
      {"From":34,"To":35,"Type":0}, {"From":35,"To":36,"Type":1},
      {"From":36,"To":12,"Type":1}, {"From":36,"To":37,"Type":0},
      {"From":37,"To":38,"Type":2}]}
```


Рис. 6. Блок-схема алгоритма, реализующего метод Гаусса-Зейделя

3.2. Выходные данные программы

Результат работы программы - текстовый файл, содержащий представление алгоритма в форме Q-детерминанта. Каждая строка этого файла содержит результирующую переменную и Q-терм, описывающий её вычисление. На рисунке 7 представлен пример для алгоритма метода Гаусса-Зейделя (блок-схема этого алгоритма была приведена в п. 3.1): представление этого алгоритма в форме Q-детерминанта для значения размерности 2 и количества итераций 1. Q-детерминант в данном случае состоит из 2 условных Q-термов, в выходном файле 2 строки. Длина и количество формирующихся Q-термов зависят от указанных параметров размерности задачи и количества итераций алгоритма.


```
X(1)={"op":"<","fO":{"op":"+","fO":{"op":"abs","od":{"op":"-
","fO":"X0(1)","sO":{"op":"/","fO":{"op":"-
","fO":"B(1)","sO":{"op":"*","fO":"A(1,2)","sO":"X0(2)"}},"sO":"A(1,1)"}}},
"sO":{"op":"abs","od":{"op":"-","fO":"X0(2)","sO":{"op":"/","fO":{"op":"-
","fO":"B(2)","sO":{"op":"*","fO":"A(2,1)","sO":{"op":"/","fO":{"op":"-
","fO":"B(1)","sO":{"op":"*","fO":"A(1,2)","sO":"X0(2)"}},"sO":"A(1,1)"}}},
"sO":"A(2,2)"}}}},"sO":"e"};{"op":"/","fO":{"op":"-
","fO":"B(1)","sO":{"op":"*","fO":"A(1,2)","sO":"X0(2)"}},"sO":"A(1,1)"}

X(2)={"op":"<","fO":{"op":"+","fO":{"op":"abs","od":{"op":"-
","fO":"X0(1)","sO":{"op":"/","fO":{"op":"-
","fO":"B(1)","sO":{"op":"*","fO":"A(1,2)","sO":"X0(2)"}},"sO":"A(1,1)"}}},
"sO":{"op":"abs","od":{"op":"-","fO":"X0(2)","sO":{"op":"/","fO":{"op":"-
","fO":"B(2)","sO":{"op":"*","fO":"A(2,1)","sO":{"op":"/","fO":{"op":"-
","fO":"B(1)","sO":{"op":"*","fO":"A(1,2)","sO":"X0(2)"}},"sO":"A(1,1)"}}},
"sO":"A(2,2)"}}}},"sO":"e"};{"op":"/","fO":{"op":"-
","fO":"B(2)","sO":{"op":"*","fO":"A(2,1)","sO":{"op":"/","fO":{"op":"-
","fO":"B(1)","sO":{"op":"*","fO":"A(1,2)","sO":"X0(2)"}},"sO":"A(1,1)"}}},
"sO":"A(2,2)"}
```


Рис. 7. Представление в форме Q-детерминант алгоритма, реализующего метод Гаусса-Зейделя

3.3. Ограничение на длину Q-термов

При достаточно больших значениях параметров размерности и количества итераций программа в ходе получения представления алгоритма в форме Q-детерминанта формирует Q-термы слишком большой длины, из-за чего выдается исключение, связанное с недостаточным объемом памяти для выполнения программы. Например, невозможно получить с помощью имеющегося ПО представление в форме Q-детерминанта алгоритма, реа-

лизующего метод Гаусса-Зейделя (см. п. 3.1), указав размерность 10 и количество итераций 2 (рисунок 8). Таким образом, имеющееся ПО имеет ограничение на длину Q-термов.

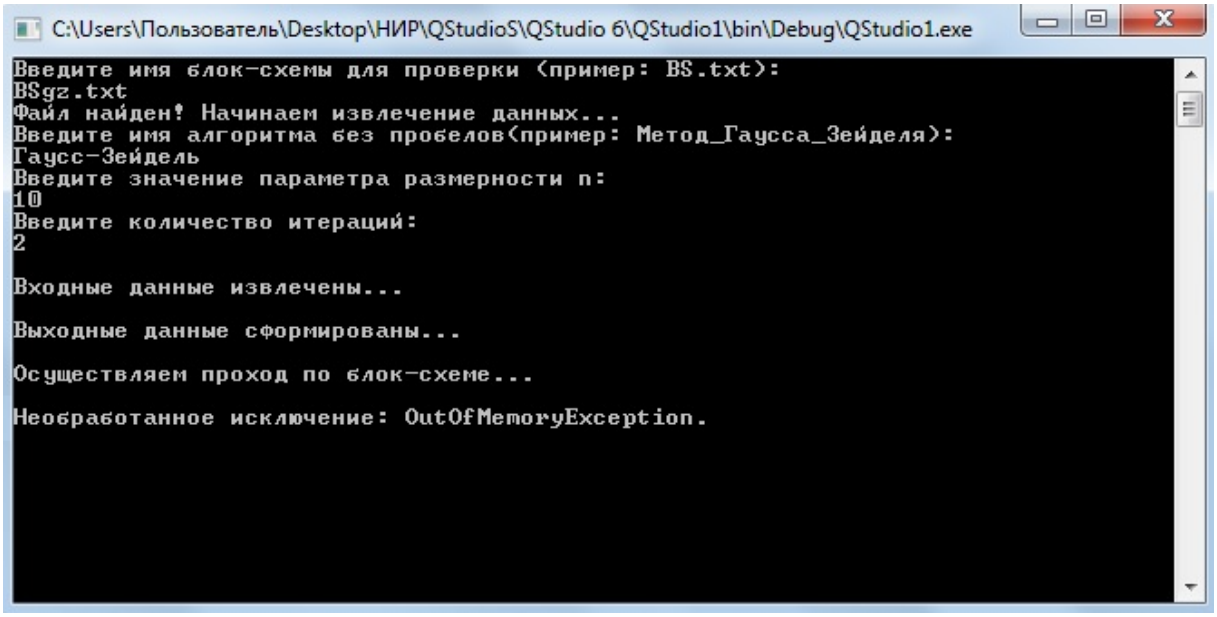

Рис. 8. Исключение при формировании Q-термов большой длины

3.4. Модернизация программного обеспечения

В рамках данной работы на основе результатов проектирования была произведена модернизация программного обеспечения, позволяющая строить представление алгоритма в форме Q-детерминанта с Q-термами большой длины. Во время написания кода в качестве справочных материалов использовались [12, 13].

В результате, например, стало возможно получить представление алгоритма, реализующего метод Гаусса-Зейделя со значением размерности 10 и количества итераций 2, который невозможно было получить ранее (рисунок 9). Размер выходного файла – 6.4 ГБ.

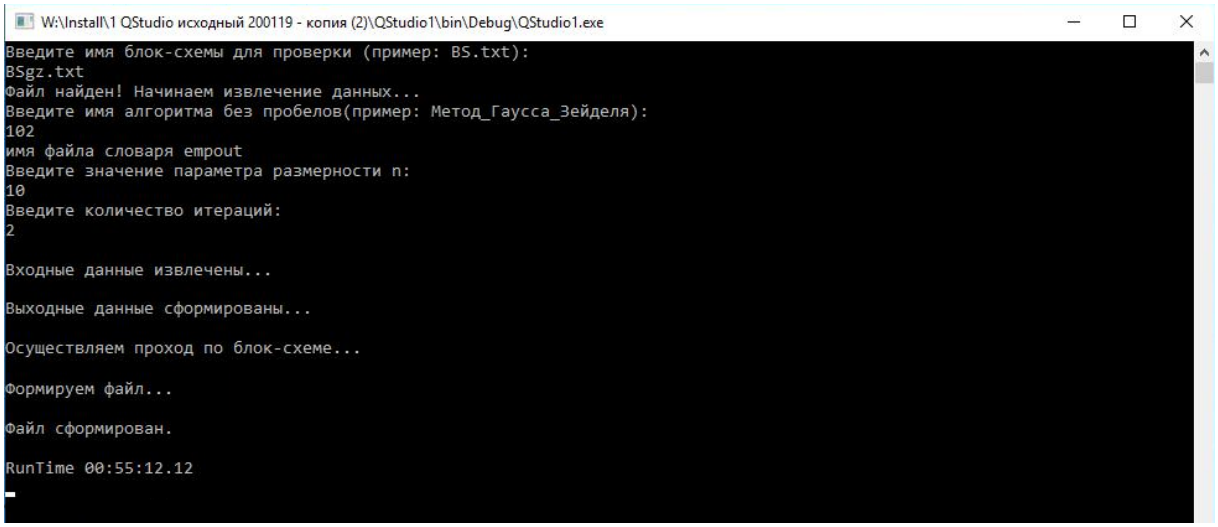

Рис. 9. Работа программы после модернизации

После модернизации существенно увеличилось время работы программы. Однако, это не имеет большого значения, поскольку Q-детерминанты формируются один раз, затем используются.

4. ТЕСТИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Для тестирования разработанного программного обеспечения необходимо использовать алгоритмы, Q-детерминанты которых:

- было возможно получить до модернизации;
- стало возможно получить после модернизации.

4.1. Алгоритмы, Q-детерминанты которых содержат Q-термы небольшой длины

Начать следует с тестирования на алгоритмах, Q-детерминанты которых содержат Q-термы небольшой длины. Результат работы программы можно будет сравнить с результатом работы ранее разработанного ПО. Это тестирование позволит удостовериться в корректности работы ПО после модернизации. Тестирование нужно осуществить с использованием алгоритмов, имеющих Q-детерминанты с разными типами Q-термов.

Пример. Рассмотрим алгоритм, реализующий метод Якоби для решения СЛАУ размерности 2, количество итераций 1. Q-детерминант данного алгоритма содержит Q-термы небольшой длины. Представление алгоритма в форме Q-детерминанта, сформированное ранее разработанным ПО, представлено на рисунке 10. Результат работы ПО после модернизации представлен на рисунке 11.

```
X(1)={"op":"<","fO":{"op":"+","fO":{"op":"abs","od":{"op":"-
","fO":"X0(1)","sO":{"op":"/","fO":{"op":"-
","fO":"B(1)","sO":{"op":"*","fO":"A(1,2)","sO":"X0(2)"}},"sO":"A(1,1)"}}},
"sO":{"op":"abs","od":{"op":"-","fO":"X0(2)","sO":{"op":"/","fO":{"op":"-
","fO":"B(2)","sO":{"op":"*","fO":"A(2,1)","sO":"X0(1)"}},"sO":"A(2,2)"}}}}
,"sO":"e"};{"op":"/","fO":{"op":"-
","fO":"B(1)","sO":{"op":"*","fO":"A(1,2)","sO":"X0(2)"}},"sO":"A(1,1)"}

X(2)={"op":"<","fO":{"op":"+","fO":{"op":"abs","od":{"op":"-
","fO":"X0(1)","sO":{"op":"/","fO":{"op":"-
","fO":"B(1)","sO":{"op":"*","fO":"A(1,2)","sO":"X0(2)"}},"sO":"A(1,1)"}}},
"sO":{"op":"abs","od":{"op":"-","fO":"X0(2)","sO":{"op":"/","fO":{"op":"-
","fO":"B(2)","sO":{"op":"*","fO":"A(2,1)","sO":"X0(1)"}},"sO":"A(2,2)"}}}}
,"sO":"e"};{"op":"/","fO":{"op":"-
","fO":"B(2)","sO":{"op":"*","fO":"A(2,1)","sO":"X0(1)"}},"sO":"A(2,2)"}
```

Рис. 10. Представление в форме Q-детерминант алгоритма, реализующего метод Якоби, полученное до модернизации ПО

```
X(1)={"op":"<","fO":{"op":"+","fO":{"op":"abs","od":{"op":"-
","fO":"X0(1)","sO":{"op":"/","fO":{"op":"-
","fO":"B(1)","sO":{"op":"*","fO":"A(1,2)","sO":"X0(2)"}},"sO":"A(1,1)"}}},
"sO":{"op":"abs","od":{"op":"-","fO":"X0(2)","sO":{"op":"/","fO":{"op":"-
","fO":"B(2)","sO":{"op":"*","fO":"A(2,1)","sO":"X0(1)"}},"sO":"A(2,2)"}}}}
,"sO":"e"};{"op":"/","fO":{"op":"-
","fO":"B(1)","sO":{"op":"*","fO":"A(1,2)","sO":"X0(2)"}},"sO":"A(1,1)"}

X(2)={"op":"<","fO":{"op":"+","fO":{"op":"abs","od":{"op":"-
","fO":"X0(1)","sO":{"op":"/","fO":{"op":"-
","fO":"B(1)","sO":{"op":"*","fO":"A(1,2)","sO":"X0(2)"}},"sO":"A(1,1)"}}},
"sO":{"op":"abs","od":{"op":"-","fO":"X0(2)","sO":{"op":"/","fO":{"op":"-
","fO":"B(2)","sO":{"op":"*","fO":"A(2,1)","sO":"X0(1)"}},"sO":"A(2,2)"}}}}
,"sO":"e"};{"op":"/","fO":{"op":"-
","fO":"B(2)","sO":{"op":"*","fO":"A(2,1)","sO":"X0(1)"}},"sO":"A(2,2)"}
```


Рис. 11. Представление в форме Q-детерминант алгоритма, реализующего метод Якоби, полученное после модернизации ПО

Результаты работы программ идентичны (см. рисунок 10 и рисунок 11), из чего следует, что логика формирования Q-детерминанта данного алгоритма не нарушена. Таким же образом была проверена корректность формирования Q-детерминантов других алгоритмов с различными типами Q-термов небольшой длины:

- алгоритм умножения матриц со схемой сдваивания;
- алгоритм умножения матриц без схемы сдваивания;
- алгоритм поиска наибольшего элемента массива;
- алгоритм решения системы линейных уравнений методом Якоби;
- алгоритм нахождения решения квадратного уравнения без унарного минуса;
- алгоритм нахождения решения квадратного уравнения с унарным минусом;
- алгоритмы вычисления скалярного произведения векторов со схемой сдваивания;
- алгоритм вычисления скалярного произведения векторов без схемы сдваивания.

4.2. Алгоритмы, Q-детерминанты которых содержат Q-термы большой длины

Q-детерминанты алгоритмов, содержащие Q-термы большой длины, формируются согласно такой же логике, по которой формируются Q-детерминанты алгоритмов, содержащие Q-термы небольшой длины. Следовательно, если Q-детерминанты, содержащие Q-термы небольшой длины, формируются корректно, что подтверждено в предыдущем подразделе, Q-детерминанты, содержащие Q-термы большой длины, также должны формироваться корректно. С помощью разработанного ранее программного обеспечения такие Q-детерминанты не могли быть сформированы.

Пример. Рассмотрим алгоритм, реализующий метод Гаусса-Зейделя размерности 20, количество итераций 1. Q-детерминант данного алгоритма содержит Q-термы большой длины, поэтому не мог быть сформирован ранее разработанным ПО. В предыдущем примере этот алгоритм имел другие параметры, поэтому его Q-детерминант содержал Q-термы небольшой длины. Блок-схема алгоритма была представлена ранее (см. рисунок 4). Результатом работы программы является текстовый файл, содержащий Q-детерминант алгоритма, размером 25,2 ГБ.

5. ЭКСПЛУАТАЦИЯ

После разработки и тестирования программное обеспечение стало использоваться для формирования представления в форме Q-детерминантов алгоритмов, которые содержат Q-термы большой длины.

На данный момент сформированы представления в форме Q-детерминантов следующих алгоритмов (таблица 4):

Табл. 4. Начало эксплуатации ПО

Название алгоритма	Размерность	Итерации
Метод Гаусса-Зейделя	2	40
Метод Гаусса-Зейделя	2	100
Метод Гаусса-Зейделя	5	4
Метод Гаусса-Зейделя	5	5
Метод Гаусса-Зейделя	9	2
Метод Гаусса-Зейделя	10	2
Метод Гаусса-Зейделя	11	2
Метод Гаусса-Зейделя	15	1
Метод Гаусса-Зейделя	20	1
Метод Гаусса-Зейделя	21	1
Метод Якоби	5	8
Метод Якоби	2	100

ЗАКЛЮЧЕНИЕ

Данная работа посвящена разработке программного обеспечения для формирования Q-детерминантов численных алгоритмов с Q-термами большой длины.

В рамках работы были получены следующие результаты:

1) изучена концепция Q-детерминанта;

2) создан проект модернизации имеющегося программного обеспечения для генерации представлений численных алгоритмов в форме Q-детерминанта;

3) разработано программное обеспечение для генерации представлений численных алгоритмов в форме Q-детерминанта, не имеющее ограничений на длину Q-термов;

4) разработанное ПО протестировано;

5) начато использование разработанного программного обеспечения для формирования Q-детерминантов алгоритмов, содержащих Q-термы большой длины.

Модернизированное программное обеспечение является подсистемой Q-system и будет использоваться совместно с другими подсистемами.

Работа выполнена при финансовой поддержке РФФИ в рамках научного проекта № 17-07-00865 а.

По теме данной работы имеется публикация [10], а также сделан доклад на Международной конференции «Суперкомпьютерные дни в России 2019».

ЛИТЕРАТУРА

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

2. Алеева В.Н., Иванов Н.А. Исследование внутреннего параллелизма численных алгоритмов // Параллельные вычислительные технологии – XII международная конференция, ПаВТ'2018, г. Ростов-на-Дону, 2–6 апреля 2018 г. Короткие статьи и описания плакатов. Челябинск: Издательский центр ЮУрГУ, 2018.

3. Voevodin V.V., Voevodin Vl.V., The V-Ray Technology of Optimizing Programs to Parallel Computers, Proc. of the 1st workshop on numerical analysis and applications, Russe, Bulgary, 24-27 June, 1999.

4. Воеводин В.В., Воеводин Вл.В., Параллельные вычисления, изд. БХВ-Петербург, СПб., 2002.

5. Вольковский В.А., Малышкин В.Э., Синтез параллельных программ и систем на вычислительных моделях. изд. Наука, Новосибирск, 1988.

6. Легалов А.И., "Функциональный язык для создания архитектурно-независимых параллельных программ", Журн. "Вычислительные технологии", 1:10 (2005), 71-89.

7. Гервич Л.Р. Автоматизация распараллеливания программ с блочным размещением данных. / Л.Р. Гервич, Е.Н. Кравченко, Б. Я. Штейнберг, М.В. Юрушкин. // Сиб. журн. вычисл. матем., 18:1 (2015) – С. 41–53.

8. 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 л.

9. Алеев Р.Ж., Алеева В.Н., Богатырева Е.С. Логический анализ численных алгоритмов на основе концепции Q-детерминанта и его применение для исследования ресурса параллелизма численных алгоритмов. // Международная конференция Мальцевские чтения 2018, г. Новосибирск,

19–22 ноября 2018 г. Сборник тезисов. Новосибирск: Изд-во Института математики СО РАН, 2018. – С. 28.

10. Aleeva V., Bogatyreva E., Skleznev A., Sokolov M., Shuppa A. Software Q-system for the Research of the Resource of Numerical Algorithms Parallelism. In: Voevodin V., Sobolev S. (eds) Supercomputing. RuSCDays 2019. Communications in Computer and Information Science, vol 1129, pp. 641–652, 2019. Springer, Cham.

11. Богатырева Е.С. Разработка программного обеспечения для генерации представления численных алгоритмов в форме Q-детерминанта: Вып. квалиф. работа магистра по направлению «Фундаментальная информатика и информационные технологии»: 02.04.02 / Южно-Уральский государственный университет. Челябинск, 2019. 41 л.

12. Mark Michaelis. Essential C# 7.0. Addison-Wesley Professional, 2018.

13. Троелсен Э. Язык программирования C# 2010 и платформа .NET 4, 5-е изд. – М.: Вильямс, 2011. – 1392 с.

14. Свирихин Д.И., Алеева В.Н. Определение максимально эффективной реализации алгоритма на основе концепции Q-детерминанта. // Параллельные вычислительные технологии (ПаВТ'2013): труды международной научной конференции (1 – 5 апреля 2013 г., г. Челябинск). Челябинск: Издательский центр ЮУрГУ, 2013. – С. 617.

15. Фаулер М. UML. Основы, 3-е издание. – СПб.: Символ-Плюс, 2004. – 192 с.