МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра Параллельных вычислительных технологий

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА

Ткачёвой Анастасии Александровны
(фамилия, имя, отчество автора)

Разработка и реализация фрагментированных алгоритмов для работы с разреженными матрицами

Направление подготовки **010500 - «Прикладная математика и информатика»**

Руководитель

Городничев М.А.
(фамилия, И.О.)

ассистент кафедры ПВт
(уч. степень, уч. звание)

.....
(подпись, дата)

Автор

Ткачёва А.А.
(фамилия, И.О.)

ФПМИ, ПМ-71
(факультет, группа)

.....
(подпись, дата)

Новосибирск, 2011 г.

Содержание

Введение 6

1. Обзор 8

1.1. Обзор библиотек для работы с разреженными матрицами 8

1.2. Система фрагментированного программирования LuNA 11

1.3. Постановка задачи 12

2. Реализация 13

2.1. Представление программы в системе LuNA. Язык LuNA 13

2.1.1. Фрагментированная программа в системе LuNA 13

2.1.2. Пример программы на языке LuNA. Умножение матриц 14

2.2. Правила фрагментации алгоритма 18

2.3. Фрагментация алгоритма метода сопряженных градиентов 19

3.3.1. Фрагментация симметричных разреженных матриц 20

2.3.1.1. Разбиение на вертикальные полоски 23

2.3.1.2. Разбиение на блоки 26

2.3.1.3. Сравнительный анализ форматов хранения 28

2.3.2. Фрагментация векторных операций 29

2.3.2.1. Вычисление скалярного произведения 29

2.3.2.2. Вычисление квадрата нормы вектора 29

3. Сравнительное тестирование фрагментированных программ 31

3.1. Дублирование фрагментов данных 31

3.1.1. Разбиение матрицы на блоки 32

3.1.2. Разбиение матрицы на вертикальные полоски 32

3.2. Зависимость времени исполнения от количества процессов 33

3.2.1. Разбиение матрицы на блоки 33

3.2.2. Разбиение матрицы на вертикальные полоски 35
3.3. Зависимость времени исполнения от количества потоков 36
3.3.1. Разбиение матрицы на блоки 36
3.3.2. Разбиение матрицы на вертикальные полоски 37
3.4. Сравнение результатов для вариантов программы с разбиением на блоки и на вертикальные полосы 38
Заключение 39
Список литературы 40
Приложение А. Текст программы для разбиения на блоки 41
Приложение Б. Текст программы для разбиения на вертикальные полоски .. 59

Введение

Постоянное развитие компьютерных технологий: рост количества ядер на процессоре, увеличение пропускной способности шин, производительность процессоров – всё это позволяет за приемлемое время решать задачи с большим объемом данных и вычислений, которые раньше невозможно было реализовать на последовательных компьютерах. Но возможности параллельных вычислений непосредственно связаны со сложностями параллельного программирования. Параллельной программе приходится управлять во времени системами, достигающими тысяч процессоров и терабайт оперативной памяти.

Поэтому для высокой производительности программа должна обладать следующими динамическими свойствами: настройка на имеющиеся ресурсы, динамическая балансировка загрузки, осуществление коммуникаций на фоне вычислений и т.д.

Сейчас в большинстве случаев эти свойства реализуются вручную с помощью MPI (Message Passing Interface), но делаются попытки реализации систем программирования и библиотек, в которых динамические свойства обеспечиваются автоматически, например, Charm++ [9], SMP Superscalar [10], PLASMA [11].

Фрагментированное программирование – это технология параллельного программирования, при которой программа представляется в виде множества отдельных фрагментов. Такое фрагментированное представление параллельной программы сохраняется до самого момента ее исполнения, что позволяет более просто решать задачи распределения ресурсов и управления. Фрагментированный подход позволяет использовать универсальные алгоритмы планирования и исполнения параллельных программ, автоматически обеспечивая переносимость и настройку на имеющиеся ресурсы.

В настоящее время разрабатывается система фрагментированного программирования LuNA [3], способная автоматически обеспечивать динамические свойства программы: параллельное исполнение, асинхронные пересылки данных, начальная настройка на ресурсы и динамическая балансировка. В системе LuNA программа представляется в специальном фрагментированном виде. Использование этой системы для решения численных задач делает необходимым разрабатывать фрагментированные алгоритмы для решения широко используемых численных методов.

Большинство задач численного моделирования не обходятся без решения СЛАУ большой размерности с разреженными матрицами. Разреженные матрицы встречаются при решении многих важных практических задач: структурного анализа, теории электрических сетей и энергосистем распределения энергии, численного решения дифференциальных уравнений, теории графов, а также генетики, социологии и поведенческих наук.

Разреженной называют матрицу, имеющую малый процент ненулевых элементов. При этом относительно местоположения нулей никаких предположений не выдвигается: они могут быть расположены совершенно произвольным образом.

Целью данной работы ставится использование принципов фрагментированного программирования при создании для системы LuNA алгоритмов и программ для работы с разреженными матрицами.

1. Обзор

Важными качествами, которыми должна обладать параллельная программа – переносимость, автоматическая настройка на ресурсы и осуществление коммуникаций на фоне вычислений.

Автоматическая настройка на ресурсы – это распределение или перераспределение данных и вычислений по узлам кластера перед началом счета с целью сбалансировать нагрузку, а значит и минимизировать время счета. Настройка выполняется в двух случаях:

1) начальная настройка перед началом счета;

2) перебалансировка в ходе вычислений, которая становится необходимой из-за особенностей алгоритма (например, на разных стадиях алгоритма требуется по-разному распределять данные), особенностей задачи (например, если распределение нагрузки напрямую зависит от моделируемого явления), или меняющейся обстановки в вычислительной среде (появляются и исчезают вычислительные узлы, запускаются и завершаются одновременно работающие задачи).

Переносимость – это критерий, который характеризует, насколько просто перенести параллельную программу с одной вычислительной системы на другую с сохранением всех ее необходимых свойств.

Коммуникации на фоне вычислений – выполнение межпроцессорных обменов на фоне вычислений путем разделения коммуникационной операции на две части: начало обмена и завершение, между которыми может быть выполнено некоторое вычисление.

1.1. Обзор библиотек для работы с разреженными матрицами

С точки зрения перечисленных выше качеств оценим существующие параллельные библиотеки для работы с разреженными матрицами.

Библиотека AZTEC

AZTEC – параллельная библиотека итерационных методов для решения систем линейных уравнений с разреженными матрицами. Она позволяет

описывать части распределенной матрицы с использованием глобальной адресации. Пакет изначально разрабатывался для того, чтобы облегчить перенос приложений с однопроцессорных систем на многопроцессорные [1]. Предоставляемые средства трансформации данных позволяют легко создавать разреженные матрицы для использования как на однопроцессорных, так и многопроцессорных системах. Из требуемых динамических свойств эта библиотека обеспечивает только выполнение обменов на фоне вычислений.

Библиотека BlockSolve95

BlockSolve95 – библиотека функций для решения больших систем линейных уравнений с симметричной разреженной матрицей на массивных параллельных системах с распределенной памятью и на сетях рабочих станций [2]. Библиотека является пакетом общего назначения, и его использование не предполагает ограничений на структуру матриц, кроме того, что они должны быть разреженными и симметричными по структуре (но не обязательно по значениям).

Библиотека Parallel Algorithms Project

Parallel Algorithms Project – пакет параллельных версий итерационных методов Крыловского типа для решения систем с разреженной матрицей [4].

Библиотека PSparselib

PSparselib (A Portable Library of Parallel Sparse Iterative Solvers) – библиотека параллельных итерационных методов для линейных систем с разреженной матрицей [7]. Эта библиотека обеспечивает выполнение обменов на фоне вычислений.

Библиотека PIM

PIM (The Parallel Iterative Methods) – набор процедур на Фортран-77, реализующих параллельные версии различных итерационных методов

решения систем линейных уравнений с разреженной матрицей [6]. Входит в состав библиотеки PETSc.

Библиотека PSPASES

PSPASES (Parallel Sparse Symmetric Direct Solver) – библиотека параллельных версий прямых методов для решения линейных систем с симметричной матрицей [8]. Эта библиотека обеспечивает выполнение обменов на фоне вычислений и начальное распределение ресурсов.

Библиотека PETSc

PETSc (Portable, Extensible Toolkit for Scientific Computation) – параллельная библиотека для численного решения дифференциальных уравнений в частных производных и сходных проблем высокопроизводительных вычислений [5]. PETSc состоит из нескольких библиотек. Каждая библиотека оперирует с определенным семейством распределенных объектов (например, векторами, матрицами), и распределенными операциями, которые применяются к этим объектам. Все программы PETSc используют стандарт MPI (Message Passing Interface) для обмена сообщениями. Библиотека поддерживает начальную настройку на ресурсы (автоматическая декомпозиция неструктурированной сетки), но не поддерживает динамическую балансировку загрузки. Также поддерживается выполнение межпроцессорных обменов на фоне вычислений.

Все перечисленные выше библиотеки используют конкретную архитектуру, конкретную топологию сети, и не позволяют эффективно подстраиваться под какие-то нестандартные условия. В тех случаях, когда какая-то библиотека поддерживает совмещение счета и вычислений, на самом деле она только предоставляет необходимые для этого средства – команды начала и окончания обменов. Фактическое совмещение обменов и счета выполняет программист, используя данные средства.

1.2. Система фрагментированного программирования LuNA

Для применения технологии фрагментированного программирования в ИВМиМГ СО РАН в Лаборатории синтеза параллельных программ разрабатывается экспериментальная система фрагментированного программирования LuNA [3]. Она включает следующие компоненты:

- язык фрагментированного программирования LuNA;
- компилятор;
- runtime-система.

Язык LuNA позволяет описать фрагментированную программу как множество фрагментов, а также содержит средства для прямых указаний и рекомендаций по распределения ресурсов. Задача компилятора – частичное статическое планирование вычислений на основе имеющейся информации о программе, о вычислительной системе и с учетом указаний и рекомендаций пользователя. Runtime-система обеспечивает исполнение фрагментированной программы на имеющемся оборудовании. В ходе работы она должна выполнять динамическое распределение ресурсов, автоматически обеспечивая программе необходимые динамические свойства: автоматическую настройку программы на имеющиеся ресурсы, динамическую балансировку загрузки, организация асинхронных обменов данными и вычислений.

Текущая реализация системы LuNA поддерживает исполнение фрагментированных программ на системах с распределенной памятью и многоядерным и узлами, исполняя на каждом узле вычислительной системы фрагменты вычислений несколькими рабочими потоками.

Использование фрагментированной системы LuNA, которая автоматически обеспечивает динамические свойства программы, во многом может упростить написание эффективной параллельной программы.

1.3. Постановка задачи

В качестве первого применения технологии фрагментированного программирования разрабатывается библиотека фрагментированных численных подпрограмм. Основные задачи при разработке библиотеки – это представление численных алгоритмов во фрагментированном виде и поиск для них хорошего распределения ресурсов.

Целью бакалаврской работы является создание фрагментированных алгоритмов для работы с разреженными матрицами и написание соответствующих фрагментированных программ на языке LuNA. В качестве конкретной задачи выдвигается создание фрагментированного алгоритма и фрагментированной программы, реализующей метод сопряженных градиентов (МСГ).

Выделим следующие подзадачи:

- разработать формат хранения симметричной разреженной матрицы;
- разработать фрагментированные алгоритмы и программы операций:
  - умножение разреженной матрицы на вектор;
  - вычисление скалярного произведения векторов;
  - вычисление нормы вектора;
- выполнить сравнение реализаций фрагментированной программы в системе LuNA для выбранных форматов хранения разреженных матриц.

2. Реализация

2.1. Представление программы в системе LuNA. Язык LuNA

Система фрагментированного программирования LuNA, включает следующие компоненты: язык LuNA, компилятор и runtime-систему. На первом этапе обработки фрагментированной программы компилятор преобразует код фрагментированной программы на языке LuNA во внутреннюю форму, выполняет некоторые глобальные оптимизации. Компилятор делает статическое распределение ресурсов для фрагментов данных и фрагментов вычислений, для которых это возможно, и накладывает некоторые ограничение на порядок выполнения фрагментов вычислений и дальнейшее динамическое распределение ресурсов. На втором этапе обработки фрагментированной программы runtime-система исполняет операции, определенные фрагментированным алгоритмом. Runtime-система динамически распределяет ресурсы мультикомпьютера для размещения фрагментов данных и фрагментов вычислений и выполняет все фрагменты вычислений в некотором допустимом порядке, который не противоречит информационным зависимостям между фрагментами вычислений, наложенными алгоритмом, и ограничениям, наложенным компилятором.

2.1.1. Фрагментированная программа в системе LuNA

Фрагментированный алгоритм на языке LuNA определяется через:

- DF – множество фрагментов данных;
- CF – множество фрагментов вычислений;
- ρ – отношение частичного порядка на множестве фрагментов вычислений.

Определение алгоритма через множества фрагментов данных, фрагментов вычислений и отношение частичного порядка над фрагментами вычислений эквивалентно множеству функциональных термов и делает возможным динамически создавать функциональные термы в ходе исполнения. Отношение порядка ρ включает информационные зависимости

вместе с другими зависимостями, добавленными компилятором на этапе глобальной оптимизации, а также явно определенные программистом. В ходе исполнения программы каждый фрагмент вычислений выполняется только один раз, тогда как фрагменты данных являются переменными множественного присваивания и обозначают область памяти, т.е. для них выполнено частичное распределение ресурсов. Это сделано с целью помочь компилятору и runtime-системе создать высококачественное распределение ресурсов.

Фрагментированная программа в системе LuNA – это фрагментированный алгоритм с частичным определением ресурсов. Такое представление программы не содержит ненужных зависимостей между фрагментами вычислений и допускает множество вариантов исполнения. Для некоторых фрагментов данных и фрагментов вычислений ресурсы задаются пользователем, для некоторых – статически при компиляции, а для остальных – динамически при исполнении программы.

2.1.2. Пример программы на языке LuNA. Умножение матриц

Описание конструкций языка LuNA будут продемонстрированы на примере фрагментированной программы умножения матриц. Фрагментированный алгоритм умножения матриц может быть записан следующим образом:

$$D^k_{i,j} = A_{i,k} B_{k,j}, \qquad C_{i,j} = \sum_{k=1}^{K} D^k_{i,j},$$

где $A_{i,k}$, $B_{k,j}$, $C_{i,j}$ – подматрицы соответствующих матриц A, B и C. $D^k_{i,j}$ – подматрицы, хранящие частичные суммы соответствующих подматриц $C_{i,j}$. Далее приводятся конструкции программы на языке LuNA, реализующие этот алгоритм.

```
df a[i,k] := block(4*M*M) | i=0..K-1, k=0..K-1;

df b[k,j] := block(4*M*M) | k=0..K-1, j=0..K-1;

df c[i,j] := block(4*M*M) | i=0..K-1, j=0..K-1;
```

```
df d[i,j,k] := block(4*M*M) | i=0..K-1, j=0..K-1, k=0..K-1;
```

Ключевое слово **df** обозначает описание фрагментов данных:

$$DF = \{ a_{i,k} \mid i=0..K-1, k=0..K-1 \} \cup \{ b_{k,j} \mid k=0..K-1, j=0..K-1 \} \cup$$

$$\cup \{ c_{i,j} \mid i=0..K-1, j=0..K-1 \} \cup \{ d_{i,j,k} \mid i=0..K-1, j=0..K-1, k=0..K-1 \}.$$

Фрагменты данных $a_{i,k}$ и $b_{k,j}$ входные, фрагменты данных $c_{i,j}$ выходные для фрагмента вычислений *mul*. Для каждого фрагмента ключевое слово **block** определяет его размер в байтах, M и K – это параметры алгоритма.

```
cf initc[i,j] := proc_zero<M,M> (out: c[i,j])
  | i=0..K-1, j=0..K-1;
cf mul[i,j,k] := proc_mmul<M,M,M> (in: a[i,k],b[k,j];
  out: d[i,j,k]) | i=0..K-1, j=0..K-1, k=0..K-1;
cf sum[i,j,k] := proc_add<M,M> (in: d[i,j,k],c[i,j];
  out: c[i,j]) | i=0..K-1, j=0..K-1, k=0..K-1;
```

Ключевое слово *cf* обозначает фрагмент вычислений:

$$CF = \{ initc_{i,j} \mid i=0..K-1, j=0..K-1 \} \cup \{ mul_{i,j,k} \mid i=0..K-1, j=0..K-1, k=0..K-1 \} \cup$$

$$\cup \{ sum_{i,j,k} \mid i=0..K-1, j=0..K-1, k=0..K-1 \}.$$

Каждому фрагменту вычислений соответствует функция, реализующая его, параметры которой задаются в угловых скобках. Входные и выходные фрагменты обозначаются ключевыми словами **in** и **out**. Для примера, каждый фрагмент вычислений `initc[i,j]` для разных `i=0..K-1` и `j=0..K-1` реализуется функцией `proc_zero`, а его выходом является один фрагмент данных `c[i,j]`.Функции `proc_zero`, `proc_mmul` и `proc_add` – это обычные последовательные функции, написанные на C/C++. Функция `proc_zero` заполняет матрицу размером $M \times M$ нулевыми значениями, `proc_mmul` выполняет умножение матриц $M \times M$, `proc_add` выполняет сложение матриц размером $M \times M$.

```
initc[i,j] < sum[i,j,k] | i=0..K-1, j=0..K-1, k=0..K-1;
mul[i,j,k] < sum[i,j,k] | i=0..K-1, j=0..K-1, k=0..K-1;
```

Отношение частичного порядка над фрагментами вычислений определяется как множество на пар, обозначенных символом «меньше». Приведенная выше фрагментированная программа задает следующее отношение порядка:

$$\rho = \{ (initc_{i,j}, sum_{i,j,k}) \mid i=0..K-1, j=0..K-1, k=0..K-1 \} \cup$$

$$\cup \{ (mul_{i,j,k}, sum_{i,j,k}) \mid i=0..K-1, j=0..K-1, k=0..K-1 \}.$$

В каждой паре второй фрагмент вычислений должен начать исполнение не раньше, чем первый завершит свое исполнение. Полностью исполнение фрагментированной программы завершится, когда все начавшие свое исполнение фрагменты вычислений завершились, и нет фрагментов вычислений, которые могут начать свое исполнение.

Приведенная выше фрагментированная программа в системе LuNA является переносимой. Она не зависит от компьютерной архитектуры и уже может быть исполнена runtime-системой. Отношение частичного порядка ρ определяет различные варианты исполнения фрагментированной программы, одни из которых являются эффективными, а другие нет. Для примера, runtime-система может первым исполнить все $mul_{i,j,k}$ и затем все $sum_{i,j,k}$, но это неизбежно приведет к необходимости хранить в памяти все $K \times K \times K$ фрагментов данных $d_{i,j,k}$. Но если для каждой пары i,j все фрагменты вычислений $mul_{i,j,k}$ и $sum_{i,j,k}$ объединить в отдельные множества $S_{i,j}$, и исполнить все фрагменты вычислений из одного $S_{i,jj}$, затем все фрагменты вычислений из другого $S_{i1,j1}$ и так далее, это приведет к необходимости хранить одновременно не более K фрагментов данных $d_{i,j,k}$ для каждой пары индексов i,j.

В [3] представлены средства оптимизации фрагментированной программы в системе LuNA. Они состоят из рекомендаций программиста для распределения ресурсов и задания порядка исполнения фрагментов вычислений:

- Задание приоритета выбора фрагмента вычисления для исполнения.

- Задание группы фрагментов вычислений со стратегией GMF (Group Member First). Она заключается в том, что если фрагменты некоторой группы начали исполнение, то все фрагменты вычислений внутри этой группы должны запуститься раньше других фрагментов вычислений. В предыдущем примере с программой умножения матриц, множества $S_{i,j}$ являются группами со стратегией GMF.
- Задание оценки вычислительной сложности фрагментов вычислений.
- Задание бинарного отношения соседства на фрагментах вычислений. Если два фрагмента вычисления определены как соседи, то runtime-система будет пытаться разместить их на одном или соседних кластерных узлах.

Используя перечисленные выше рекомендации, программист может предоставить runtime-системе дополнительную информацию о желательном варианте исполнения фрагментированной программы. Рекомендации не содержат предложений о строгом порядке фрагментов или ресурсов и поддерживают переносимость фрагментированной программы. Дополнительно программист может использовать прямые инструкции о порядке присвоения фрагментам вычисления для особых процессов или включение дополнительных элементов в отношении ρ, чтобы сократить множество различных вариантов исполнения фрагментированной программы.

Подходящий вариант исполнения для фрагментированной программы умножения матриц может быть определен, например, усилением отношения порядка и явным заданием соответствия фрагментов вычислений и процессов.

```
initc[i,j].location = i * PE_COUNT / K | i=0..K-1, j=0..K-1;
mul[i,j,k].location = i * PE_COUNT / K
  | i=0..K-1, j=0..K-1, k=0..K-1;
sum[i,j,k].location = i * PE_COUNT / K
```

```
  | i=0..K-1, j=0..K-1, k=0..K-1;
mul[i,(i+j)%K,k] < mul[i,(i+j+1)%K,k]
  | i=0..K-1, j=0..K-2, k=0..K-1;
```

Здесь параллельная программа умножения матриц определена на линейке вычислительных узлов. Матрица *A* распределена между узлами по строкам, и это распределение фиксировано. Матрица *B* распределена между узлами по столбцам, и ее фрагменты циклически смещаются из одного узла в другой (изображено на рисунке 1).

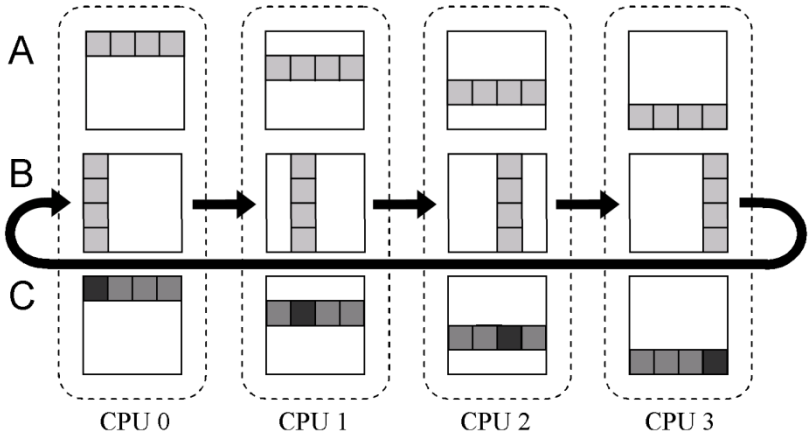

Рисунок 1 – Пример распределения фрагментов данных по процессорам в фрагментированной программе умножения матриц

2.2. Правила фрагментации алгоритма

Методика фрагментации алгоритмов еще разрабатывается, но сейчас уже можно выделить некоторые рекомендации, следование которым позволит обеспечить необходимое качество фрагментированной программы.

1. Начальную структуру данных следует разбивать на фрагменты данных, так чтобы они имели такую же структуру – матрицу следует разбить на подматрицы, сети – на подсети, дерево – на поддеревья и т.д. Это делает возможным использовать готовый последовательный код или оптимизированную библиотеку подпрограмм для реализации фрагментов вычислений.

2. Все фрагменты данных должны иметь одинаковый размер, и все фрагменты вычислений должны иметь приблизительно одинаковую вычислительную сложность, чтобы автоматически обеспечить хорошее распределение ресурсов и динамическую балансировку загрузки.

3. Размер фрагмента данных и вычислительная сложность фрагмента вычислений должна быть параметром фрагментированного алгоритма и не должны зависеть от размера задачи. В таком случае при изменении размера задачи изменится только число фрагментов, и можно будет использовать те же алгоритмы управления и распределения ресурсов.

4. Фрагменты данных должны формировать такую же структуру, как и начальная структура данных, и порядок исполнения фрагментов вычислений должен быть таким же, как порядок операций в исходном алгоритме. Это качество позволит изменять размер фрагментов вплоть до одного элемента исходного (нефрагментированного) алгоритма.

Фрагментация численных алгоритмов – непростая задача, и часто требует трансформации исходного алгоритма. Однако многие численные алгоритмы могут быть фрагментированы «естественным образом».

2.3. Фрагментация алгоритма метода сопряженных градиентов

Приведем последовательный алгоритм метода сопряженных градиентов решающий систему линейных алгебраических уравнений (СЛАУ) $Ax = f$ с начальным приближением x^0:

$$r^0 = f - Ax^0$$

$$z^0 = r^0$$

$$\mathrm{k} = 1,2, \ldots$$

$$\alpha^k = \frac{(r^{k-1}, r^{k-1})}{(Az^{k-1}, z^{k-1})}$$

$$x^k = x^{k-1} + \alpha^k z^{k-1}$$

$$r^k = r^{k-1} - \alpha^k Az^{k-1}$$

$$\beta^k = \frac{(r^k, r^k)}{(r^{k-1}, r^{k-1})}$$

$$z^k = r^k + \beta^k z^{k-1}$$

Выход из цикла, если $\frac{\|r^k\|}{\|f\|} < \varepsilon$

Здесь матрица A симметричная разреженная матрица, которая будет фрагментирована одним из выбранных ниже способов. Вектора r^k, z^k, x^k, f – фрагментированные векторы.

Как видно, в приведенном алгоритме можно выделить следующие подзадачи:

- умножение матрицы на вектор;
- векторные операции:
    - вычисление скалярного произведения;
    - вычисление нормы вектора.

Для этих подзадач необходимо реализовать фрагментированные алгоритмы.

3.3.1. Фрагментация симметричных разреженных матриц

Исходя из перечисленных выше рекомендаций к созданию фрагментированного алгоритма, требуется выбрать способ фрагментации данных в алгоритме.

Для разреженного столбцового формата хранения симметричной разреженной матрицы *A* размером *N* на *N* с количеством ненулевых элементов *K* в дальнейшем будут использоваться следующие обозначения:

di – массив размером *N* вещественных чисел, элементами которого являются диагональные элементы матрицы;

ja – массив целых чисел размером *K*, хранящий номера строк ненулевых наддиагональных элементов матрицы;

gg – массив вещественных чисел размером *K*, хранящий значения ненулевых наддиагональных элементов матрицы;

ia – массив размером *N+1* целых чисел, хранящий указатели на начала столбцов в массиве *ja* и *gg*.

При выборе способа фрагментации разреженной матрицы можно придерживаться одной из двух стратегий.

Первая стратегия заключается в том, что разреженную матрицу представляют в известном разреженном формате, а полученные массивы равномерно разделяют на части и работают уже с ними. Такое разбиение гарантирует одинаковый размер блоков, а, значит, и возможность равномерно загрузить ресурсы компьютера. Кроме того, мы заведомо не храним никакой лишней информации. К недостаткам можно отнести тот факт, что во многом усложняется написание фрагментированного алгоритма, так как невозможно по номеру фрагмента матрицы сказать к каким строкам и столбцам относятся элементы внутри него.

Фрагментированный алгоритм умножения матрицы, представленной таким способом, на вектор будет выглядеть на языке LuNA следующим образом:

```
df  x[i] :=block(SizeofDouble*BI) | i = 1..n;
df  f[i] :=block(SizeofDouble*BI) | i = 1..n;
df  ja[i] :=block(SizeofInt*BI) | i = 1..k;
df  ia[i] :=block(SizeofInt*BI) | i = 1..k;
df  gg[i] :=block(SizeofDouble*BI) | i = 1..k;
cf init[i] := init <k,i>(out: gg[i], ia[i], ja[i]) | i = 1..k;
cf init2[i] : =init <n,i>(out: x[i], f[i]) | i =1..n;
cf  mult[i,j,k] := mult <k,n,i,j,k>(in: gg[i], ia[i], ja[i], x[j], f[k]; out: f[k])  | i =1..k, j = 1..n, k=1..n;
init[i] < mult[i,j,k] | i =1..k, j = 1..n, k=1..n;
init2[j] < mult[i,j,k] | i =1..k, j = 1..n, k=1..n;
init2[k] < mult[i,j,k] | i =1..k, j = 1..n, k=1..n;
```

Здесь *n* – размерность матрицы, *k* – задаваемое количество фрагментов, *BI* – количество элементов во фрагменте. Фрагменты данных: *x[i]* и *f[i]* –

части умножаемого вектора и вектора результата соответственно; в *ia[i]*, *ja[i]*, *gg[i]* содержится информация о частях матрицы – строке, столбце и значении элемента. Фрагменты вычислений *init[i]* и *init2[i]* инициализируют фрагменты данных вектора и фрагменты данных матрицы соответственно. Фрагмент вычислений *mult[i,j,k]* делает умножение *i*-фрагмента матрицы на *j*-фрагмент вектора *x* и записывает его в *k*-фрагмент вектора *f*.

Схема, приведенная на рисунке 2, иллюстрирует использование фрагментов данных фрагментами вычислений *mult[i,j,k]* при выполнении операции умножения матрицы на вектор. Для простоты изложения и наглядности было взято *k=2*. Также введено следующее обозначение: фрагмент данных *mat[i]* есть объединение элементов фрагментов *gg[i]*, *ia[i]*, *ja[i]*. Все фрагменты вычислений *mult[i,j,k]* в таком фрагментированном алгоритме могут исполняться в произвольном порядке.

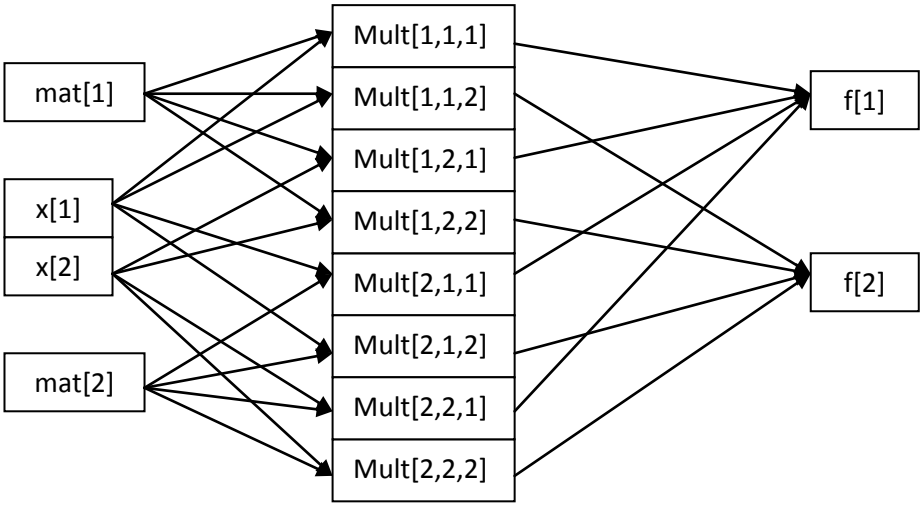

Рисунок 2 – Схема использования фрагментов данных фрагментами вычислений в алгоритме умножения матрицы на вектор с фрагментацией по способу 1

Как видно из схемы, при таком представлении матрицы необходимо использовать большое число фрагментов вычислений, во многих из которых вычисления, возможно, производиться не будут. В такой ситуации сильно

возрастает доля накладных расходов на исполнение фрагментированной программы. Общее количество фрагментов вычислений имеет порядок $k*k*k$. Количество требуемой памяти будет зависеть от формата хранения матрицы в самих фрагментах вычислений. В тоже время для такого разбиения матрицы выполнимы 1, 2, 3 рекомендации к фрагментации алгоритма.

При использовании второй стратегии матрица разбивается на подматрицы, каждая из которых хранится в некотором разреженном формате. Разбиение матрицы на подматрицы можно выполнить разными способами, например, на полоски (по срокам или по столбцам) или на прямоугольные блоки. Так как рассматривается работа с симметричными матрицами, то при разбиении на полоски способ разбиения по строкам или по столбцам не принципиален.

2.3.1.1. Разбиение на вертикальные полоски

От симметричной разреженной матрицы достаточно хранить только верхний треугольник. Диагональ, как и внедиагональную часть, разобьем на K фрагментов. Внедиагональные фрагменты будем хранить в разреженном столбцовом формате (изображено на рисунке 3).

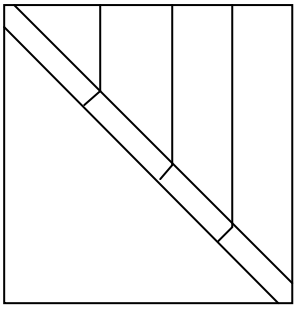

Рисунок 3 – Схема фрагментации матрицы на вертикальные полоски

Недостатком такого представления является нарушение 2 и 3 рекомендации к фрагментированному алгоритму, а именно фрагменты данных значительно отличаются по размеру, и размер фрагментов данных зависит от размера матрицы.

Фрагментированный алгоритм умножения матрицы на вектор при фрагментации на вертикальные полоски будет выглядеть на языке LuNA следующим образом:

*df di[i] :=block(SizeofDouble*BI) | i = 1..n;*

*df xk[i] :=block(SizeofDouble*BI) | i = 1..n;*

*df rk[i] :=block(SizeofDouble*BI) | i = 1..n;*

df ia[i] :=block(SizeofInt(NI+1)) | i = 1..k;*

df ja[i] :=block(SizeofInt NI*NI*i) | i = 1..k;*

*df gg[i] :=block(SizeofDouble*NI*NI*i) | i = 1..k;*

cf init[i]:= init <k,i>(out: xk[i], rk[i], gg[i], ia[i], ja[i]) | i = 1..k;

cf mult_di[i]:= mult_di<BI>(in: di[i], xk[i], rk[i]; out: rk[i]) | i = 1..k;

cf mult_gg[i,j,k]:= mult_gg<BI,NI,i,j,k>(in: xk[i], ia[j], ja[j], gg[j], rk[k]; out: rk[k]) | j = 1..k, i = 1..j, k=1..j;

init[i] < mult_di[i] | i = 1..k;

init[j] < mult_gg[i,j,k] | j = 1..k, i = 1..j, k=1..j;

Здесь *n* – размерность матрицы, *k* - задаваемое количество фрагментов, BI – количество элементов во фрагментах *di[i]*, *xk[i]*, *rk[i]*, *NI* – количество строк во фрагменте *ia[i]*. Фрагменты данных: *xk[i]* и *rk[i]* части умножаемого вектора и вектора результата соответственно, в *ia[i]*, *ja[i]*, *gg[i]* содержится информация о частях матрицы. Фрагменты вычислений *init[i]* инициализирует фрагменты данных вектора и фрагменты данных матрицы соответственно. Фрагменты вычислений *mult_di[i]* выполняют умножение части диагонали на соответствующую часть вектора. Фрагменты вычислений *mult_gg[i,j,k]* выполняют умножение *i*-фрагмента матрицы на *j*-фрагмент вектора *xk* и записывают его в *k*-фрагмент вектора *rk*.

Схема, приведенная на рисунке 4, иллюстрирует использование фрагментов данных фрагментами вычисления в работе умножение матрицы на вектор. Для простоты изложения и наглядности было взято *k=2*. Также

введено следующее обозначение фрагмент данных *mat[i]* есть объединение элементов фрагментов *gg[i], ia[i], ja[i]*.

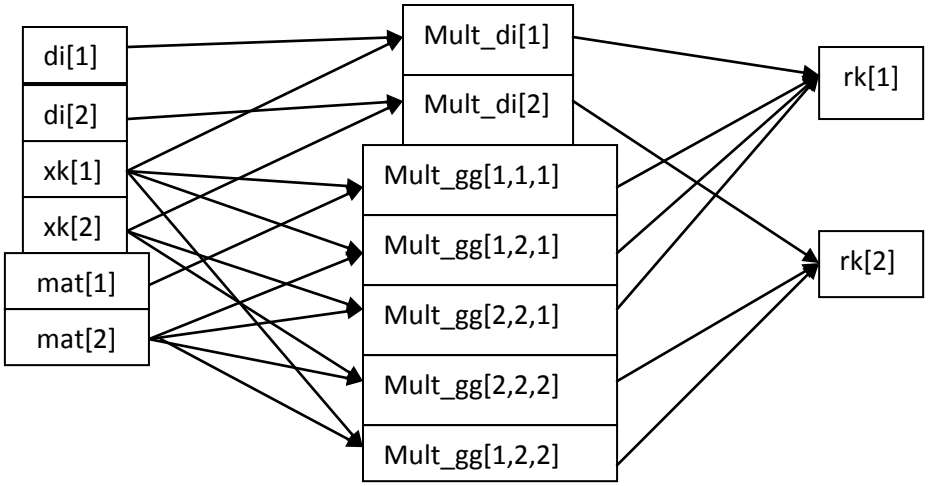

Рисунок 4 – Схема использования фрагментов данных фрагментами вычислений в алгоритме умножения матрицы на вектор с фрагментацией на полоски по способу 2

В случае фрагментации на вертикальные полоски также есть фрагменты вычислений, которые могут и не производить вычисления, но их максимально возможное количество значительно меньше, чем при фрагментации симметричной матрицы, представленной разреженным строчным форматом. Общее количество фрагментов вычислений имеет порядок $\frac{k(k+1)}{2}$.

Матрица в столбцовом разреженном формате занимает $\left(\left(\frac{n}{k}+1\right)k + n1\right) * c + (n+n1) * f$ единиц памяти, где *n1* – число ненулевых внедиагональных элементов матрицы, *c* – количество памяти, требуемая для хранения переменной типа *int, f* – количество памяти, требуемая для хранения переменной типа *double, n* – число элементов на диагонали, k – параметр, задающий степень фрагментации.

2.3.1.2. Разбиение на блоки

Верхний треугольник симметричной матрицы разобьется на K фрагментов диагонали и на $\frac{k(k+1)}{2}$ внедиагональных фрагментов. Внедиагональные фрагменты хранятся в разреженном столбцовом формате. Также внедиагональные фрагменты можно подразделить на два вида: треугольные и в общем случае прямоугольные фрагменты данных (изображено на рисунке 5).

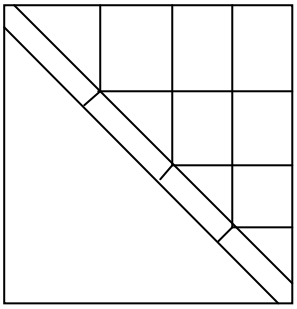

Рисунок 5 – Схема фрагментации матрицы на блоки

Так же, как и при фрагментации на вертикальные полосы, этот метод не удовлетворяет строго рекомендации 2 относительно одинакового размера фрагментов данных. Но он удовлетворяет рекомендации 3 относительно независимости размера данных от размера задачи.

Фрагментированный алгоритм умножения матрицы на вектор при фрагментации на блоки будет выглядеть на языке LuNA следующим образом:

```
df  di[i] :=block(SizeofDouble*BI) | i = 1..n;
df  xk[i] :=block(SizeofDouble*BI) | i = 1..n;
df  rk[i] :=block(SizeofDouble*BI) | i = 1..n;
df  ia[i,j] :=block(SizeofInt*(NI+1)) | i = 1..k, j = 1..i;
df  ja[i,j] :=block(SizeofInt* NI*NI) | i = 1..k, j = 1..i;
df  gg[i,j] :=block(SizeofDouble*NI*NI*i) i = 1..k, j = 1..i;
cf init[i]:= init <k,i>(out: xk[i], rk[i],) | i = 1..k;
cf init2[i,j]:=init_gg<k,i>(out: gg[i,j], ia[i,j], ja[i,j]) |i=1..k,j=1..i;
```

cf mult_di[i]:= mult_di<BI>(in: di[i], xk[i], rk[i]; out: rk[i]) | i = 1..k;

cf mult_gg1[i]:=mult_gg1<BI,NI,i>(in: xk[i], ia[i,i], ja[i,i], gg[i,i], rk[i]; out: rk[i]) | i =1..k;

cf mult_gg2[i,j]:= mult_gg2<BI,NI,i,j>(in: xk[i], xk[j], ia[i,j], ja[i,j], gg[i,j], rk[i], rk[j]; out: rk[i], rk[j]) | i=2..k, j =1..i-1;

init[i] < mult_di[i] | i = 1..k;

init[i] < mult_gg1[i] | i =1..k;

init2[i,i] < mult_gg1[i] | i =1..k;

init[i] < mult_gg2[i,j] | i=2..k, j =1..i-1;

init[j] < mult_gg2[i,j] | i=2..k, j =1..i-1;

init2[i,j] < mult_gg2[i,j] | i=2..k, j =1..i-1;

init2[i,j] < mult_gg2[i,j] | i=2..k, j =1..i-1;

Здесь *n* – размерность матрицы, *k* – задаваемое количество фрагментов, BI – количество элементов во фрагментах *di[i], xk[i], rk[i], NI* – количество строк во фрагменте *ia[i]*. Фрагменты данных: *xk[i]* и *rk[i]* части умножаемого вектора и вектора результата соответственно, в *ia[i,j], ja[i,j], gg[i,j]* содержится информация о частях матрицы, индексы i и j характеризуют положение фрагмента в матрице, строку и столбец соответственно. Фрагменты вычислений *init[i]* и *init2[i]* инициализируют фрагменты данных вектора и фрагменты данных матрицы соответственно. Фрагменты вычислений *mult_di[i]* выполняют умножение части диагонали на соответствующую часть вектора. Фрагменты вычислений *mult_gg1[i]* выполняют умножение треугольных фрагментов матрицы. Фрагменты вычислений *mult_gg2[i,j]* выполняют умножение *(i,j)*-фрагмента матрицы на *j*-фрагмент вектора *xk* и записывают его в *i*-фрагмент вектора *rk.*

Схема, приведенная на рисунке 6, иллюстрирует использование фрагментов данных фрагментами вычисления в работе умножение матрицы на вектор. Для простоты изложения и наглядности было взято *k=2*. Также

введено следующее обозначение фрагмент данных *mat[i,j]* есть объединение элементов фрагментов *gg[i,j], ia[i,j], ja[i,j].*

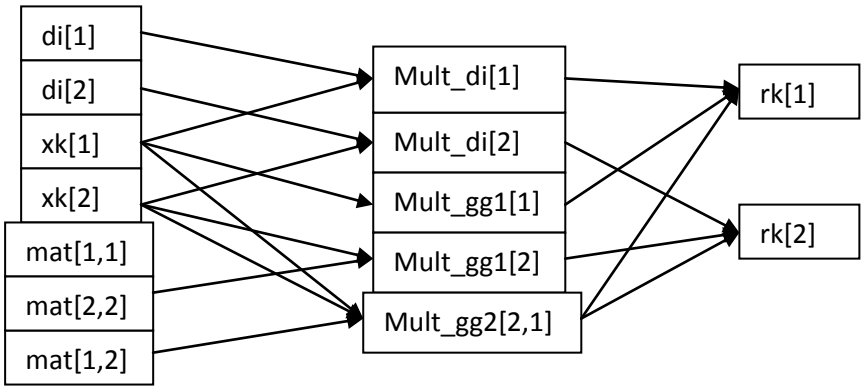

Рисунок 6 – Схема использования фрагментов данных фрагментами вычислений в алгоритме умножения матрицы на вектор с фрагментацией на блоки по способу 2

Общее количество фрагментов вычислений имеет порядок $\frac{k(k+1)}{2}$, что совпадает с вариантом разбиения на вертикальные полоски. Но в тоже время вариант с разбиением на прямоугольные блоки обладает большей степенью параллелизма.

На хранение информации о матрице затрачивается $\left(\left(\frac{n}{k}+1\right)*\frac{k(k+1)}{2}+n1\right)*c+(n+n1)*f$ единиц памяти, где *n1* –ненулевых внедиагональных элементов матрицы, *c* – количество памяти, требуемая для хранения переменной типа *int, f* количество памяти, требуемая для хранения переменной типа *double, n* – число элементов на диагонали, *k* – параметр, задающий степень фрагментации.

2.3.1.3. Сравнительный анализ форматов хранения

Вариант с разбиением матрицы на блоки при таком же количестве фрагментов вычислений, как и в варианте с разбиением матрицы на вертикальные полосы, позволяет реализовать фрагментированный алгоритм с

большей степенью параллелизма. В тоже время этот вариант требует большего количества памяти на хранение матрицы по сравнению с разбиением на вертикальные полоски.

2.3.2. Фрагментация векторных операций

2.3.2.1. Вычисление скалярного произведения

Фрагментированный алгоритм скалярного произведения представлен ниже:

*df a[i]:=block(SizeOfDouble*BI) |i=1..K;*

*df b[i]:=block(SizeOfDouble*BI) |i=1..K;*

df scal:=block(SizeOfDouble);

df norm1:=block(SizeOfDouble);

cf init[i]:= init<BI> (out: a[i], b[i]) | i=1..K;

cf init_pr:=init_param(out:scal,norm1);

cf sc[i]: = scalar<BI> (in: a[i], b[i], scal; out: scal) |i =1..K;

init[i] < sc[i] | i=1..K;

init_param < sc[i] | i = 1..K;

Здесь два вектора фрагментированы одинаково, *BI* – размер фрагмента вектора, *K* – количество фрагментов. Фрагменты вычислений *init[i]* инициализирует фрагменты векторов *a* и *b*, *init_pr* инициализирует фрагмент данных *norm1* и *scal*, *sc[i]* выполняют функцию скалярного произведения для *i*-фрагментов векторов.

2.3.2.2. Вычисление квадрата нормы вектора

Приведем фрагментированный алгоритм расчета квадрата нормы вектора. Инициализация фрагментов данных приведена в предыдущем примере.

cf norm[i]:=norma<BI>(in: a[i], norm1; out: norm1) |i =1..K;

init_param < norm[i] | i = 1..K;

init[i] < norm[i] |i=1..K;

Здесь фрагменты вычислений *norm[i]* выполняют функцию вычисления квадрата нормы для *i*–фрагмента вектора и добавляют результат к значению фрагмента данных *norm1*.

Полностью фрагментированная программа на языке LuNA, реализующая метод сопряженных градиентов с разбиением матрицы на полоски, приведена в приложении А, а фрагментированная программа с разбиением матрицы на блоки – в приложении Б.

3. Сравнительное тестирование фрагментированных программ

Было проведено сравнительное тестирование фрагментированных программ на языке LuNA, реализующих метод сопряженных градиентов, при использовании различных форматов хранения разреженных матриц.

Тестирование проводилось на кластере МВС-100К (МСЦ РАН). Основные параметры кластера приведены в таблице 1.

Таблица 1 – Характеристики кластера МВС-100К

Количество ядер	8576
Тип процессоров	Intel(R) Xeon(R) CPU E5450 3GHz, Intel(R) Xeon(R) CPU 5365 3GHz
Число ядер в узле	8
Объем оперативной памяти узла, Гб	8
Коммуникационная сеть	InfiniBand 4x DDR

Для тестирования использовалась матрица размерностью 4545 на 4545 с 46640 ненулевыми внедиагональными элементами. Сравнивалось количество времени на исполнение (секунды).

3.1. Дублирование фрагментов данных

В алгоритме МСГ на каждой итерации вычисляются значения переменных α^k, β^k . При фрагментации алгоритма эти переменные необходимо использовать одновременно на нескольких узлах. Т.к. в текущей версии системы LuNA автоматическая рассылка фрагментов данных по всем узлам не выполняется, одно значение будет последовательно передаваться с узла на узел, исключая возможность параллельного исполнения. Эту проблему может решить дублирование этих переменных для всех вычислительных узлов.

3.1.1. Разбиение матрицы на блоки

Выполнено тестирование на 4 процессах, у каждого процесса 8 потоков. Результаты приведены в таблице 2 и на рисунке 7. Проделано 100 итераций МСГ.

Таблица 2 – Сравнение времени исполнения для разбиения на блоки

Разбиение	Время исполнения (секунды)	
	Без дублирования	С дублированием
1	17,78552	26,94332
3	59,42875	53,87879
5	163,2923	127,0674
9	487,3324	333,9781
15	1322,079	926,6276

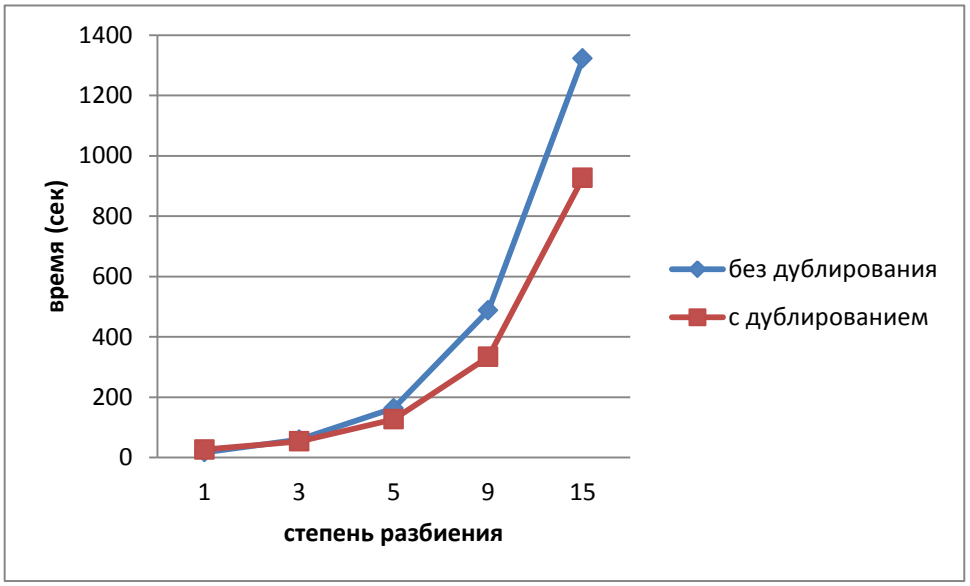

Рисунок 7 – Сравнение времени исполнения для разбиения на блоки

Из приведенных результатов видно, что реализация с дублированием требует меньше количества времени на исполнение. Увеличение времени исполнения при увеличении степени фрагментации говорит о возрастающих накладных расходах на управление фрагментами в системе LuNA.

3.1.2. Разбиение матрицы на вертикальные полоски

Тестирование на 4 процессах у каждого процесса 8 потоков. Результаты приведены в таблице 3 и на рисунке 8. Проделано 10 итераций МСГ.

Таблица 3 – Сравнение времени исполнения для разбиения на блоки

Разбиение	Время выполнения(секунды)	
	Без дублирования	С дублированием
1	21,30031	30,07006
3	146,9459	129,9074
5	784,4721	781,8932

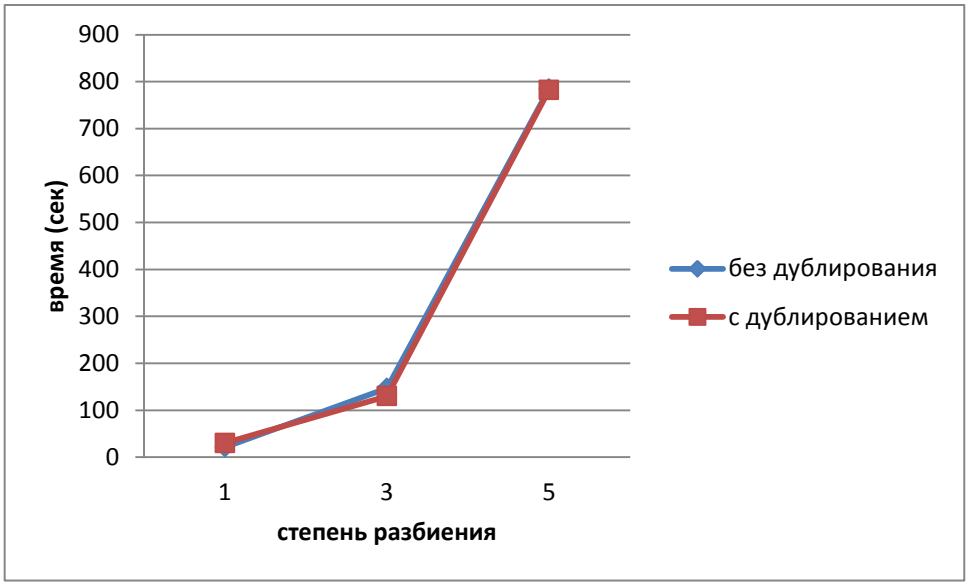

Рисунок 8 – Сравнение времени исполнения для разбиения на полоски

Из приведенных результатов видно, что реализация с дублированием требует меньше количества времени на исполнение. Правда, разница для разбиения на полоски не значительна.

Дальнейшие исследования проведены на реализации с дублированием.

3.2. Зависимость времени исполнения от количества процессов

3.2.1. Разбиение матрицы на блоки

Результаты тестирования фрагментированной программы с разбиением матрицы на блоки приведены в таблице 4 и на рисунке 9. Здесь у каждого процесса 1 поток. Проделано 100 итераций МСГ. Разбиение на $k=9$.

Таблица 4 – Зависимость времени исполнения от количества процессов для разбиения матрицы на блоки

Количество процессов	Время выполнения
2	518,058
3	439,296
4	344,406
5	297,711
6	284,6308
7	285,5768
8	301,2794
9	309,566
10	243,954
12	283,255209
16	341,103505

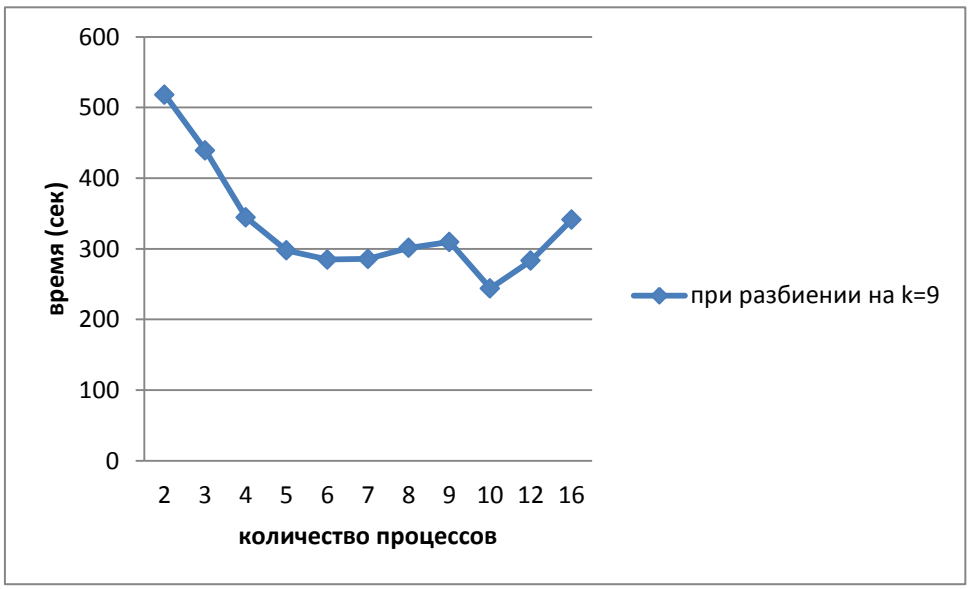

Рисунок 9 – Зависимость времени исполнения от увеличения количества процессов при разбиении на блоки

Из приведенных результатов видно, что при увеличении количества процессов, количество требуемого времени на исполнения программы уменьшается. Но доходя до некоторой величины процессов, уменьшение прекращается, и время начинает возрастать. Это связанно с ростом накладных расходов на коммуникации.

3.2.2. Разбиение матрицы на вертикальные полоски

Результаты тестирования фрагментированной программы с разбиением матрицы на вертикальные полоски приведены в таблице 5 и на рисунке 10. Здесь у каждого процесса 1 поток. Проделано 10 итераций МСГ. Разбиение на $k=5$.

Таблица 5 – Зависимость времени исполнения от количества процессов для разбиения матрицы на вертикальные полосы

Количество процессов	Время выполнения
2	813,35671
4	749,42742
8	727,25467
10	733,00592
12	727,73273
16	796,24862

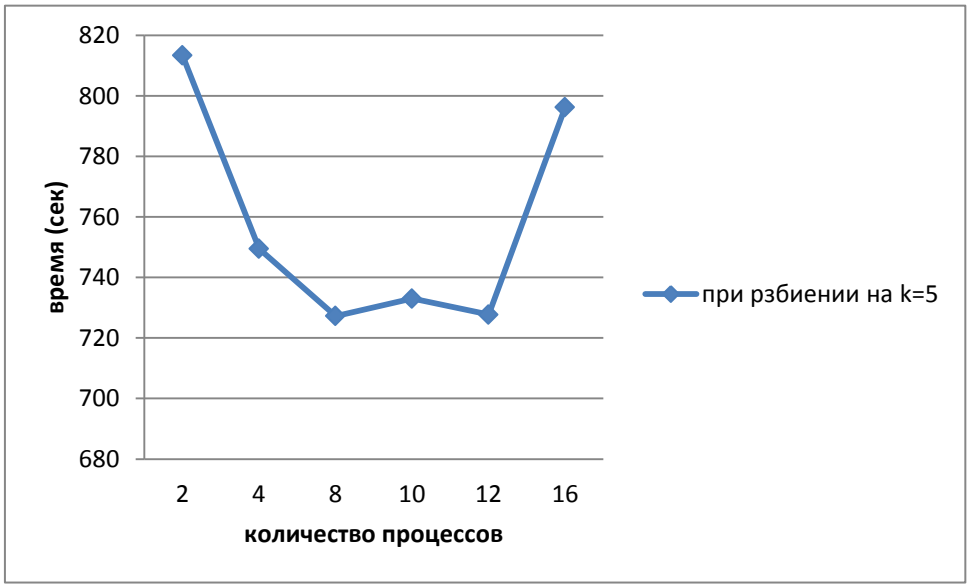

Рисунок 10 – Зависимость времени исполнения от увеличения количества процессов при разбиении на вертикальные полоски

В приведенных результатах прослеживается такая же тенденция, как и для разбиения на блоки.

3.3. Зависимость времени исполнения от количества потоков

3.3.1. Разбиение матрицы на блоки

Результаты тестирования фрагментированной программы с разбиением матрицы на блоки приведены в таблице 6 и на рисунке 11. Здесь у каждого процесса 8 потоков. Проделано 100 итераций МСГ. Разбиение на $k=9$.

Таблица 6 – Зависимость времени исполнения от количества потоков для разбиения матрицы на блоки

Количество процессов	Время выполнения
2	551,386466
3	446,778348
4	439,322416
5	302,399861
8	300,817841
10	252,372966
12	278,010797
16	342,748011

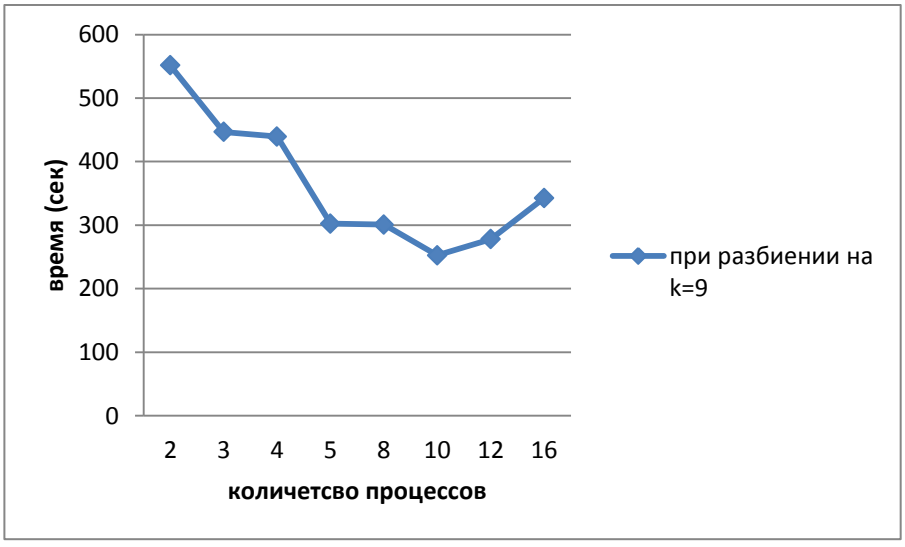

Рисунок 11 – Зависимость времени исполнения от увеличения количества потоков при разбиении на блоки

Из приведенных результатов видно, что при росте количества потоков, время исполнения уменьшается. Но, доходя до некоторой величины потоков, уменьшение прекращается, и время начинает возрастать. Это связанно, с

ростом накладных расходов на коммуникации и с тем, что в вычислительном узле доступно всего 8 ядер.

3.3.2. Разбиение матрицы на вертикальные полоски

Результаты тестирования фрагментированной программы с разбиением матрицы на вертикальные полоски приведены в таблице 7 и на рисунке 12. Здесь у каждого процесса 8 поток. Проделано 10 итераций МСГ. Разбиение на $k=5$.

Таблица 7 – Зависимость времени исполнения от количества процессов для разбиения матрицы на вертикальные полосы

Количество процессов	Время выполнения
2	912,90158
4	771,83563
8	711,62757
12	695,78635
16	785,28407

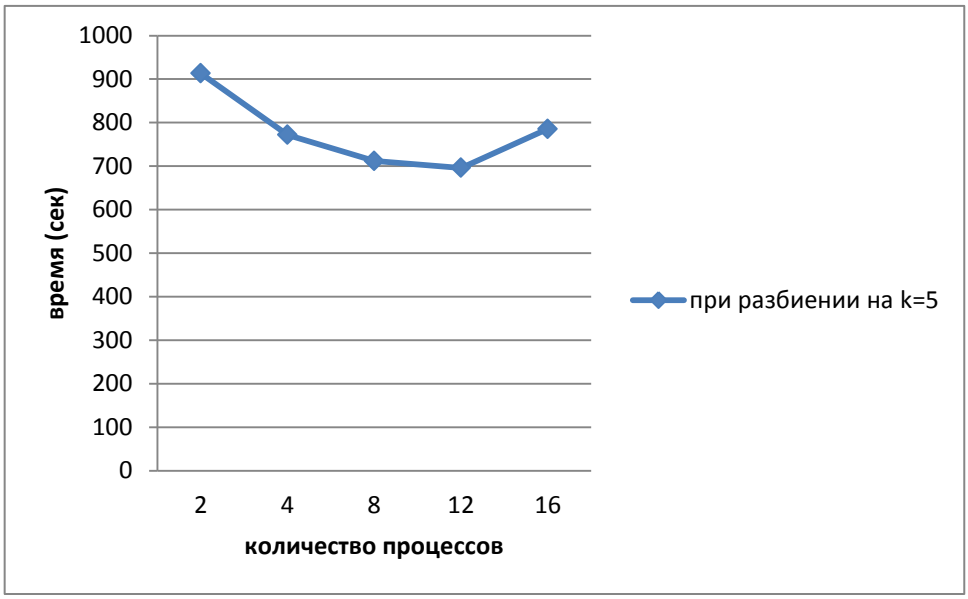

Рисунок 12 – Зависимость времени исполнения от увеличения количества потоков при разбиении на вертикальные полоски

В приведенных результатах прослеживается такая же тенденция, как и для разбиения на блоки.

3.4. Сравнение результатов для вариантов программы с разбиением на блоки и на вертикальные полосы

Из результатов тестирования можно сделать вывод о том, что при реализации программы с фрагментацией матрицы блоками времени затрачивается меньше. А так как этот формат представления позволяет добиться большей степени параллелизма, то можно сказать, что, несмотря на большее количество требуемой памяти, разбиение на блоки наиболее подходящий способ фрагментации для разреженной матрицы.

Заключение

В работе рассмотрена проблема эффективного представления параллельных алгоритмов с разреженными матрицами. Выполнен обзор и анализ параллельных библиотек для работы с разреженными матрицами. Разработаны форматы хранения фрагментированных симметричных разреженных матриц. Разработаны фрагментированные алгоритмы, реализующий метод сопряженных градиентов для разных форматов хранения разреженной матрицы. Алгоритмы реализованы в системе фрагментированного программирования LuNA. Выполнено сравнительное тестирование реализаций программ в системе LuNA для различных форматов хранения матрицы. По результатам тестирования сделаны выводы.

В дальнейшем планируется продолжить работу по следующим направлениям:

- исследование способов эффективного распределения ресурсов в разработанных алгоритмах (путем задания рекомендаций и прямого управления);
- разработка фрагментированных алгоритмов для других операций с разреженными матрицами с целью наполнения библиотеки фрагментированных численных подпрограмм.

Список литературы

1. AZTEC [Электронный ресурс]. – Режим доступа: http://www.cs.sandia.gov/CRF/aztec1.html.
2. Blocksolve95 [Электронный ресурс]. – Режим доступа: http://www.mcs.anl.gov/sumaa3d/BlockSolve95/.
3. Optimization of Parallel Execution of Numerical Programs in LuNA Fragmented Programming System / V.E. Malyshkin, V.A. Perepelkin // MTPP-2010 revised selected papers. – 2010. – LNCS 6083. – p. 1-10.
4. PAP [Электронный ресурс]. – Режим доступа: http://www.cerfacs.fr/algor/Setup/JSEnabledFrameset.html.
5. PETSc [Электронный ресурс]. – Режим доступа: http://www.mcs.anl.gov/petsc/petsc-as/.
6. PIM [Электронный ресурс]. – Режим доступа: http://chasqueweb.ufrgs.br/~rudnei.cunha/pim.html.
7. PSparselib [Электронный ресурс]. – Режим доступа: http://www.cs.umn.edu/Research/arpa/p_sparslib/psp-abs.html.
8. PSPASES [Электронный ресурс]. – Режим доступа: http://www-users.cs.umn.edu/~mjoshi/pspases/.
9. Charm++ [Электронный ресурс]. – Режим доступа: http://charm.cs.uiuc.edu/
10. SMP Superscalar [Электронный ресурс]. – Режим доступа: http://www.bsc.es/smpsuperscalar
11. Distibuted Dense Numerical Linear Algebra Algorithms on Massively Parallel Architectures: DPLASMA / G. Bosilca [и др.] // University of Tennessee Computer Science Technical Report. – 2010. – UT-CS-10-660, Sept. 15.

Приложение А. Текст программы для разбиения на блоки

Фрагментированная программа в системе LuNA (без дублирования)

```
//@CODE CodeFrags_generate_kubik.cpp
const P = PE_COUNT;
const  N = 4545;                    //size matrix
const  K = 9;                       //number block
const  KI = K*(K+1)/2;   //number block ia
const  BI = N/K;            //size block di
const  NI = N/K;  //number row of block ia
 const maxiter = 100;
const SizeOfFloat = 8;
const SizeOfInt = 4;
//-----------------------------------------------------
df di[i]  :=block(SizeOfFloat*BI) | i=1..K;
df ia[i,j] :=block(SizeOfInt*(NI+1))   | i=1..K,j=1..i;
df ja[i,j] :=block(SizeOfInt*NI*NI)   | i=1..K, j=1..i;
df gg[i,j] :=block(SizeOfFloat*NI*NI) | i=1..K, j=1..i;
df f[i]   :=block(SizeOfFloat*BI)   | i=1..K;
df xk[i]  :=block(SizeOfFloat*BI)   | i=1..K;
df rk[i] :=block(SizeOfFloat*BI) | i=1..K;
df zk[i]  :=block(SizeOfFloat*BI)   | i=1..K;
df Azk[i] :=block(SizeOfFloat*BI)   | i=1..K;
df norm1  :=block(SizeOfFloat);
df norm2  :=block(SizeOfFloat);
df beta  := block(SizeOfFloat);
df scal :=block(SizeOfFloat);
df scal2:=block(SizeOfFloat);
df one := block(SizeOfFloat);
 ////////------initilize data----------///////
 cf init[i] := init_di <K,BI,i>(out: di[i],xk[i],rk[i],f[i],Azk[i]) | i=1..K;
 cf init2[i,j] :=init_gg<KI,N,i,j,NI,K> (out: ia[i,j], ja[i,j], gg[i,j]) |i=1..K,j=1..i;
cf init_pr:= init_scalar(out:norm1,norm2,beta,scal,scal2,one);
```

```
cf time  := time_start();
//////--------norma f----------/////
cf norm_f[i]:=norma<BI>(in:f[i],norm1; out:norm1) |i=1..K;
////////-------rk = multiply: A*x ----------/////
cf mult_di[i] :=mult_di<BI>(in: di[i],xk[i],rk[i];out:rk[i]) | i=1..K;
cf mult_gg1[i]:= mult_gg1<BI, NI, i>(in: xk[i], ia[i,i], ja[i,i], gg[i,i], rk[i]; out:rk[i])  |i=1..K;
cf mult_gg2[i,j] :=mult_gg2<BI,NI,i,j>(in: xk[i], ia[i,j], ja[i,j], gg[i,j], rk[i], rk[j], xk[j]; out:rk[i], rk[j])| i=2..K,j=1..i-1;
////////-------rk = f - rk -----zk=rk----------/////
cf c_rk[i] := calculate<BI,1>(in:f[i], one, one, rk[i]; out:rk[i]) |i=1..K;
cf c_zk[i] := copy<BI>(in: rk[i]; out:zk[i])| i=1..K;
//-----------------i=1..maxiter-----------------------------------------////
cf rk_rk[i,j]:=scalar<BI,j>(in:rk[j],rk[j],scal;out:scal) | i=1..maxiter,j=1..K;
cf c_Az[i,j]:=mult_di_v1<BI>(in: di[j], zk[j], Azk[j]; out: Azk[j]) |i=1..maxiter,j=1..K;
cf  c_Az1[i,j]:=mult_gg1<BI,  NI,  j>(in:  zk[j],  ia[j,j],  ja[j,j],  gg[j,j],  Azk[j];  out:  Azk[j]) |i=1..maxiter,j=1..K;
cf  c_Az2[i,j,m]:=mult_gg2<BI,NI,j,m>(in:  zk[j],  ia[j,m],  ja[j,m],  gg[j,m],  Azk[j],  Azk[m], zk[m]; out: Azk[j], Azk[m]) |i=1..maxiter,j=2..K, m=1..j-1;
cf Azk_zk[i,j]:=scalar<BI,j>(in: Azk[j], zk[j] ,scal2; out: scal2) | i=1..maxiter,j=1..K;
cf c_xk1[i,j]:=calculate<BI,0>(in: xk[j], scal, scal2, zk[j]; out: xk[j]) |i=1..maxiter,j=1..K;
cf c_rk1[i,j]:=calculate<BI,1>(in: rk[j], scal, scal2, Azk[j]; out:rk[j]) |i=1..maxiter,j=1..K;
cf rk1_rk1[i,j]:=scalar<BI>(in: rk[j], rk[j], beta;out:beta) | i=1..maxiter,j=1..K;
cf c_zk1[i,j]:=calculate<BI,0>(in: rk[j], beta, scal, zk[j]; out:zk[j]) |i=1..maxiter,j=1..K;
cf norm_rk1[i,j]:=norma<BI>(in: rk[j], norm2; out:norm2) | i = 1..maxiter, j=1..K;
cf  nul[i]:=proverka<i>(in:scal,  scal2,  beta,  norm1,  norm2;  out:scal,  scal2,  beta,  norm2) |i=1..maxiter;
cf fin := _abort();
//-------------------------------------------------------------
init[i] < init[i+1] | i = 1..K-1;
init2[i,j] < init2[i,j+1] | i = 1..K, j=1..i-1;
init2[i,j] < init2[i+1,j] | i = 1..K-1, j = 1..i;
init2[i,j] < init2[i+1,j+1] | i = 1..K-1, j = 1..i-1;
```

init[i] < mult_di[i] |i = 1..K;

init2[i,i] < mult_gg1[i] | i = 1..K;

init2[i,j] < mult_gg2[i,j] | i = 2..K,j = 1..i-1;

init[i] < mult_gg1[i] | i = 1..K;

init[i] < mult_gg2[i,j] |i = 2..K,j = 1..i-1;

init[j] < mult_gg2[i,j] |i = 2..K,j = 1..i-1;

init[i] < time | i =1..K;

init2[i,j] < time | i = 1..K,j = 1..i;

init_pr < time;

time < norm_f[i] | i =1..K;

time < mult_di[i] | i =1..K;

time < mult_gg1[i] | i =1..K;

time < mult_gg2[i,j] | i =2..K,j=1..i-1;

init[i] < norm_f[i] | i = 1..K;

init_pr < norm_f[i] | i = 1..K;

init_pr < c_rk[i] | i = 1..K;

mult_di[i] < c_rk[i] | i = 1..K;

mult_gg1[i] < c_rk[i] |i = 1..K;

mult_gg2[i,j] < c_rk[i] | i = 2..K,j=1..i-1;

mult_gg2[i,j] < c_rk[j] | i = 2..K,j=1..i-1;

c_rk[i] < c_zk[i] | i =1..K;

c_rk[i] < rk_rk[1,i] | i =1..K;

nul[i-1] < rk_rk[i,j] | i = 2..maxiter, j = 1..K;

rk_rk[j,i] < rk_rk[j+1,i] |j=1..maxiter-1, i =1..K;

c_zk[j] < c_Az[1,j] | j=1..K;

nul[i-1] < c_Az[i,j] | i = 2..maxiter,j = 1..K;

c_Az[i,j] < c_Az[i+1,j] | i=1..maxiter-1, j =1..K;

c_Az2[m,i,j] < c_Az2[m+1,i,j] |m = 1..maxiter-1,i=2..K,j=1..i-1;

c_Az[i,j] < c_Az1[i,j] | i = 1..maxiter,j=1..K;

c_Az[m,i] < c_Az2[m,i,j] |m = 1..maxiter,i=2..K,j=1..i-1;

c_Az[m,j] < c_Az2[m,i,j] |m = 1..maxiter,i=2..K,j=1..i-1;

c_Az[i,j] < Azk_zk[i,j] |i = 1..maxiter,j=1..K;

```
c_Az1[i,j] < Azk_zk[i,j] |i = 1..maxiter,j=1..K;
c_Az2[m,i,j] < Azk_zk[m,i] | m = 1..maxiter,i=2..K,j=1..i-1;
c_Az2[m,i,j] < Azk_zk[m,j] | m = 1..maxiter,i=2..K,j=1..i-1;
Azk_zk[i,j] < Azk_zk[i+1,j] |i = 1..maxiter-1,j=1..K;
Azk_zk[i,j] < c_xk1[i,k] |i = 1..maxiter,j=1..K, k = 1..K;
rk_rk[i,j]  < c_xk1[i,k] | i = 1..maxiter,j=1..K, k = 1..K;
rk_rk[i,j] < c_rk1[i,k] |i = 1..maxiter,j=1..K, k = 1..K;
Azk_zk[i,j] < c_rk1[i,k] | i = 1..maxiter,j=1..K, k = 1..K;
c_xk1[i,j] < c_xk1[i+1,j] | i = 1..maxiter-1,j=1..K;
c_rk1[i,j] < c_rk1[i+1,j] | i = 1..maxiter-1,j =1..K;
c_rk1[i,j] <  rk1_rk1[i,j] | i = 1..maxiter, j=1..K;
rk1_rk1[i,j] < rk1_rk1[i+1,j] |  i = 1..maxiter-1, j=1..K;
rk1_rk1[i,j] < c_zk1[i,k] | i = 1..maxiter,j=1..K,k=1..K;
c_zk1[i,j] < c_zk1[i+1,j] | i = 1..maxiter-1, j=1..K;
c_rk1[i,j] < norm_rk1[i,j] | i = 1..maxiter,j=1..K;
norm_rk1[i,j] < norm_rk1[i+1,j] |i = 1..maxiter-1,j=1..K;
norm_rk1[i,j] < nul[i] | i = 1..maxiter, j = 1..K;
norm_f[j] < nul[i] | i = 1..maxiter,j = 1..K;
c_zk1[i,j] < nul[i] | i =1..maxiter,j=1..K;
nul[i] < nul[i+1] | i = 1..maxiter-1;
nul[maxiter] < fin;
//------------------------------------------------------
init_pr.location = 0;
norm_f[i].location = 0 | i = 1..K;
c_rk[j].location = j%P  | j = 1..K;
c_zk[j].location = j%P | j = 1..K;
rk_rk[i,j].location = 0 | i = 1..maxiter,j=1..K;
c_Az[i,j].location = j%P| i = 1..maxiter,j=1..K;
c_Az1[i,j].location = j%P| i = 1..maxiter,j=1..K;
c_Az2[m,i,j].location = i%P |m = 1..maxiter,i=2..K,j=1..i-1;
Azk_zk[i,j].location = 0 | i = 1..maxiter,j=1..K;
c_xk1[i,j].location = j%P| i = 1..maxiter,j=1..K;
```

```
c_rk1[i,j].location = j%P| i = 1..maxiter,j=1..K;
rk1_rk1[i,j].location = 0 | i = 1..maxiter,j=1..K;
c_zk1[i,j].location = j%P | i = 1..maxiter,j=1..K;
norm_rk1[i,j].location = 0| i = 1..maxiter,j=1..K;
nul[i].location = 0 | i =1..maxiter;
fin.location = 0;
time.location = 0;
init[i].location = i%P | i = 1..K;
init2[i,j].location = i%P | i = 1..K,j = 1..i;
mult_di[i].location = i%P | i =1..K;
mult_gg1[i].location = i%P |i = 1..K;
mult_gg2[i,j].location = i%P |i = 2..K,j=1..i-1;
```

Фрагментированная программа в системе LuNA (с дублированием)

```
//@CODE CodeFrags_generate_kubik.cpp
const P = PE_COUNT;
const  N = 4545;                    //size matrix
const  K = 9;                       //number  block
const  KI = K*(K+1)/2;   //number  block ia
const  BI = N/K;            //size block di
const  NI = N/K;  //number row of block ia
 const maxiter = 100;
const SizeOfFloat = 8;
const SizeOfInt = 4;
//-----------------------------------------------------
df di[i]  :=block(SizeOfFloat*BI) | i=1..K;
df ia[i,j] :=block(SizeOfInt*(NI+1))   | i=1..K,j=1..i;
df ja[i,j] :=block(SizeOfInt*NI*NI)   | i=1..K, j=1..i;
df gg[i,j] :=block(SizeOfFloat*NI*NI) | i=1..K, j=1..i;
df f[i]   :=block(SizeOfFloat*BI)   | i=1..K;
df xk[i]  :=block(SizeOfFloat*BI)   | i=1..K;
df rk[i] :=block(SizeOfFloat*BI) | i=1..K;
df zk[i]  :=block(SizeOfFloat*BI)   | i=1..K;
```

```
df Azk[i] :=block(SizeOfFloat*BI)   | i=1..K;
df norm1  :=block(SizeOfFloat);
df norm2  :=block(SizeOfFloat);
df beta  := block(SizeOfFloat);
df beta_copy[j,i]:= block(SizeOfFloat) | j=1..maxiter, i=0..P-1;
df scal :=block(SizeOfFloat);
df scal_copy[j,i] := block(SizeOfFloat) | j=1..maxiter ,i=0..P-1;
df scal2:=block(SizeOfFloat);
df scal2_copy[j,i] := block(SizeOfFloat) | j=1..maxiter,i=0..P-1;
df one := block(SizeOfFloat);
df one_copy[i]:= block(SizeOfFloat) | i=0..P-1;
 ////////------initilize data----------///////
 cf init[i] := init_di <K,BI,i>(out: di[i],xk[i],rk[i],f[i],Azk[i]) | i=1..K;
 cf init2[i,j] :=init_gg<KI,N,i,j,NI,K> (out: ia[i,j], ja[i,j], gg[i,j]) |i=1..K,j=1..i;
cf init_pr:= init_scalar(out:norm1,norm2,beta,scal,scal2,one);
cf one_cp[i]:= cp (in:one;out:one_copy[i]) | i=0..P-1;
cf time  := time_start();
//////--------norma f---------------/////
cf norm_f[i]:=norma<BI>(in:f[i],norm1; out:norm1) |i=1..K;
////////-------rk = multiply: A*x -----------------/////
cf mult_di[i] :=mult_di<BI>(in: di[i],xk[i],rk[i];out:rk[i]) | i=1..K;
cf mult_gg1[i]:= mult_gg1<BI, NI, i>(in: xk[i], ia[i,i], ja[i,i], gg[i,i], rk[i]; out:rk[i])  |i=1..K;
cf mult_gg2[i,j] :=mult_gg2<BI,NI,i,j>(in: xk[i], ia[i,j], ja[i,j], gg[i,j], rk[i], rk[j], xk[j]; out:rk[i],
rk[j])| i=2..K,j=1..i-1;
////////-------rk = f - rk -----zk=rk------------/////
cf c_rk[i] := calculate<BI,1>(in:f[i], one_copy[i%P], one_copy[i%P], rk[i]; out:rk[i]) |i=1..K;
cf c_zk[i] := copy<BI>(in: rk[i]; out:zk[i])| i=1..K;
//-----------------i=1..maxiter-----------------------------------------////
cf rk_rk[i,j]:=scalar<BI,j>(in:rk[j],rk[j],scal;out:scal) | i=1..maxiter,j=1..K;
cf scal_cp[i,j]:= cp(in:scal;out:scal_copy[i,j]) | i=1..maxiter,j=0..P-1;
cf c_Az[i,j]:=mult_di_v1<BI>(in: di[j], zk[j], Azk[j]; out: Azk[j]) |i=1..maxiter,j=1..K;
```

```
cf c_Az1[i,j]:=mult_gg1<BI, NI, j>(in: zk[j], ia[j,j], ja[j,j], gg[j,j], Azk[j]; out: Azk[j])
|i=1..maxiter,j=1..K;
cf c_Az2[i,j,m]:=mult_gg2<BI,NI,j,m>(in: zk[j], ia[j,m], ja[j,m], gg[j,m], Azk[j], Azk[m],
zk[m]; out: Azk[j], Azk[m]) |i=1..maxiter,j=2..K, m=1..j-1;
cf Azk_zk[i,j]:=scalar<BI,j>(in: Azk[j], zk[j] ,scal2; out: scal2) | i=1..maxiter,j=1..K;
cf scal2_cp[i,j]:= cp (in:scal2;out:scal2_copy[i,j]) | i=1..maxiter,j=0..P-1;
cf c_xk1[i,j]:=calculate<BI,0>(in: xk[j], scal_copy[i,j%P], scal2_copy[i,j%P], zk[j]; out: xk[j])
|i=1..maxiter,j=1..K;
cf c_rk1[i,j]:=calculate<BI,1>(in: rk[j], scal_copy[i,j%P], scal2_copy[i,j%P], Azk[j]; out:rk[j])
|i=1..maxiter,j=1..K;
cf rk1_rk1[i,j]:=scalar<BI>(in: rk[j], rk[j], beta;out:beta) | i=1..maxiter,j=1..K;
cf beta_cp[i,j]:= cp (in:beta;out:beta_copy[i,j]) | i=1..maxiter,j=0..P-1;
cf c_zk1[i,j]:=calculate<BI,0>(in: rk[j], beta_copy[i,j%P], scal_copy[i,j%P], zk[j]; out:zk[j])
|i=1..maxiter,j=1..K;
cf norm_rk1[i,j]:=norma<BI>(in: rk[j], norm2; out:norm2) | i = 1..maxiter, j=1..K;
cf nul[i]:=proverka<i>(in:scal, scal2, beta, norm1, norm2; out:scal, scal2, beta, norm2)
|i=1..maxiter;
cf fin := _abort();
//-------------------------------------------------------------
init[i] < init[i+1] | i = 1..K-1;
init2[i,j] < init2[i,j+1] | i = 1..K, j=1..i-1;
init2[i,j] < init2[i+1,j] | i = 1..K-1, j = 1..i;
init2[i,j] < init2[i+1,j+1] | i = 1..K-1, j = 1..i-1;
init[i] < mult_di[i] |i = 1..K;
init2[i,i] < mult_gg1[i] | i = 1..K;
init2[i,j] < mult_gg2[i,j] | i = 2..K,j = 1..i-1;
init[i] < mult_gg1[i] | i = 1..K;
init[i] < mult_gg2[i,j] |i = 2..K,j = 1..i-1;
init[j] < mult_gg2[i,j] |i = 2..K,j = 1..i-1;
init[i] < time | i =1..K;
init2[i,j] < time | i = 1..K,j = 1..i;
init_pr < time;
```

time < norm_f[i] | i =1..K;

time < mult_di[i] | i =1..K;

time < mult_gg1[i] | i =1..K;

time < mult_gg2[i,j] | i =2..K,j=1..i-1;

init[i] < norm_f[i] | i = 1..K;

init_pr < norm_f[i] | i = 1..K;

init_pr < one_cp[i] | i = 0..P-1;

one_cp[i%P] < c_rk[i] | i = 1..K;

mult_di[i] < c_rk[i] | i = 1..K;

mult_gg1[i] < c_rk[i] |i = 1..K;

mult_gg2[i,j] < c_rk[i] | i = 2..K,j=1..i-1;

mult_gg2[i,j] < c_rk[j] | i = 2..K,j=1..i-1;

c_rk[i] < c_zk[i] | i =1..K;

c_rk[i] < rk_rk[1,i] | i =1..K;

nul[i-1] < rk_rk[i,j] | i = 2..maxiter, j = 1..K;

rk_rk[j,i] < rk_rk[j+1,i] |j=1..maxiter-1, i =1..K;

rk_rk[i,j] < scal_cp[i,m] | i=1..maxiter, j = 1..K, m=0..P-1;

c_zk[j] < c_Az[1,j] | j=1..K;

nul[i-1] < c_Az[i,j] | i = 2..maxiter,j = 1..K;

c_Az[i,j] < c_Az[i+1,j] | i=1..maxiter-1, j =1..K;

c_Az2[m,i,j] < c_Az2[m+1,i,j] |m = 1..maxiter-1,i=2..K,j=1..i-1;

c_Az[i,j] < c_Az1[i,j] | i = 1..maxiter,j=1..K;

c_Az[m,i] < c_Az2[m,i,j] |m = 1..maxiter,i=2..K,j=1..i-1;

c_Az[m,j] < c_Az2[m,i,j] |m = 1..maxiter,i=2..K,j=1..i-1;

c_Az[i,j] < Azk_zk[i,j] |i = 1..maxiter,j=1..K;

c_Az1[i,j] < Azk_zk[i,j] |i = 1..maxiter,j=1..K;

c_Az2[m,i,j] < Azk_zk[m,i] | m = 1..maxiter,i=2..K,j=1..i-1;

c_Az2[m,i,j] < Azk_zk[m,j] | m = 1..maxiter,i=2..K,j=1..i-1;

Azk_zk[i,j] < Azk_zk[i+1,j] |i = 1..maxiter-1,j=1..K;

Azk_zk[i,j] < scal2_cp[i,m] |i=1..maxiter,j=1..K,m=0..P-1;

scal2_cp[i,k%P] < c_xk1[i,k] |i = 1..maxiter, k = 1..K;

scal_cp[i,k%P] < c_xk1[i,k] | i = 1..maxiter, k = 1..K;

```
scal_cp[i,k%P]< c_rk1[i,k] |i = 1..maxiter, k = 1..K;
scal2_cp[i,k%P] < c_rk1[i,k] | i = 1..maxiter, k = 1..K;
c_xk1[i,j] < c_xk1[i+1,j] | i = 1..maxiter-1,j=1..K;
c_rk1[i,j] < c_rk1[i+1,j] | i = 1..maxiter-1,j =1..K;
c_rk1[i,j] <  rk1_rk1[i,j] | i = 1..maxiter, j=1..K;
rk1_rk1[i,j] < rk1_rk1[i+1,j] |  i = 1..maxiter-1, j=1..K;
rk1_rk1[i,j] < beta_cp[i,m] | i=1..maxiter,j=1..K,m=0..P-1;
beta_cp[i,k%P] < c_zk1[i,k] | i = 1..maxiter,k=1..K;
c_zk1[i,j] < c_zk1[i+1,j] | i = 1..maxiter-1, j=1..K;
c_rk1[i,j] < norm_rk1[i,j] | i = 1..maxiter,j=1..K;
norm_rk1[i,j] < norm_rk1[i+1,j] |i = 1..maxiter-1,j=1..K;
norm_rk1[i,j] < nul[i] | i = 1..maxiter, j = 1..K;
norm_f[j] < nul[i] | i = 1..maxiter,j = 1..K;
c_zk1[i,j] < nul[i] | i =1..maxiter,j=1..K;
nul[i] < nul[i+1] | i = 1..maxiter-1;
nul[maxiter] < fin;
//-------------------------------------------------------
init_pr.location = 0;
norm_f[i].location = 0 | i = 1..K;
c_rk[j].location = j%P  | j = 1..K;
c_zk[j].location = j%P | j = 1..K;
rk_rk[i,j].location = 0 | i = 1..maxiter,j=1..K;
c_Az[i,j].location = j%P| i = 1..maxiter,j=1..K;
c_Az1[i,j].location = j%P| i = 1..maxiter,j=1..K;
c_Az2[m,i,j].location = i%P |m = 1..maxiter,i=2..K,j=1..i-1;
Azk_zk[i,j].location = 0 | i = 1..maxiter,j=1..K;
c_xk1[i,j].location = j%P| i = 1..maxiter,j=1..K;
c_rk1[i,j].location = j%P| i = 1..maxiter,j=1..K;
rk1_rk1[i,j].location = 0 | i = 1..maxiter,j=1..K;
c_zk1[i,j].location = j%P | i = 1..maxiter,j=1..K;
norm_rk1[i,j].location = 0| i = 1..maxiter,j=1..K;
nul[i].location = 0 | i =1..maxiter;
```

```
fin.location = 0;
time.location = 0;
init[i].location = i%P | i = 1..K;
init2[i,j].location = i%P | i = 1..K,j = 1..i;
mult_di[i].location = i%P | i =1..K;
mult_gg1[i].location = i%P |i = 1..K;
mult_gg2[i,j].location = i%P |i = 2..K,j=1..i-1;
one_cp[i].location = 0 | i=0..P-1;
scal_cp[i,j].location = 0 | i=1..maxiter,j=0..P-1;
scal2_cp[i,j].location = 0 | i=1..maxiter,j=0..P-1;
beta_cp[i,j].location = 0 | i=1..maxiter,j=0..P-1;
```

Файл Code_generate_kubik. cpp

```
#include <iostream>
#include <stdlib.h>
#include <math.h>
#include <stdio.h>
#include "iCodeFrag.h"
#include "iLuna.h"
#include <sys/time.h>
using namespace std;
using namespace luna;
struct timeval tv1;
#define BBI 4545/1
#define KKK 1
#define EPS 1e-6
void time_start(CodeArgs& in, CodeArgs& out, vector<int>& params)
{
   gettimeofday(&tv1,NULL);
}
void cp(CodeArgs& in, CodeArgs& out, vector<int>& params)
{
   double* a =(double*)in[0].Data;
   double* b = (double*)out[0].Data;
   b[0] = a[0];
}
void PostProcessDFs()
{
```

```
        int rank;
        MPI_Comm_rank(MPI_COMM_WORLD,&rank);
        if(rank == 0)
        {
            struct timeval tv2,dtv;
            gettimeofday(&tv2,NULL);
            dtv.tv_sec = tv2.tv_sec - tv1.tv_sec;
            dtv.tv_usec = tv2.tv_usec - tv1.tv_usec;
            if(dtv.tv_usec < 0) {dtv.tv_sec--; dtv.tv_usec+=1000000;}
            printf("Time: %lf\n",dtv.tv_sec+dtv.tv_usec*1e-6);
        }
        FILE*fp;
        fp = fopen("rez_kubik.txt","w");
        for(int i = 1; i <= KKK;i++)
        {
           fInstance df("xk");
           df.index.push_back(i); // 1st index is '2'
           double*p = (double*) luna::getDF(df);
           if(p!=NULL)
           {
               for(int j = 0;j < BBI; j++)
               fprintf(fp,"% .15lf \n", p[j]);
           }
        }
fclose(fp);
}
///cf init[i] := init_di <K,BI,i>(out: di[i],xk[i],rk[i],f[i],fk[i]) | i=1..K;
void init_di(CodeArgs& in, CodeArgs& out, vector<int>& params)
{
   int K = params[0];
   int BI = params[1];
   int ki = params[2];
   double* di = (double*)out[0].Data;
   double* xk = (double*)out[1].Data;
   double* rk = (double*)out[2].Data;
   double* f = (double*)out[3].Data;
   double* fk = (double*)out[4].Data;
   ki--;
   int i;
   for(i = 0;i < BI; i++)
   {
        rk[i] = 0.0;
```

```
        fk[i] = 0.0;
        xk[i] = 2.0;
    }
    FILE*hp;
    double* diag = new double [K*BI];
    double* pr = new double [K*BI];
    hp = fopen("di.txt","r");
    for(i = 0; i < K*BI; i++)
        fscanf(hp,"%lf",&diag[i]);
    fclose(hp);
    hp = fopen("f_1.txt","r");
    for(i = 0; i < K*BI; i++)
        fscanf(hp,"%lf",&pr[i]);
    fclose(hp);
    for(i = 0; i < BI; i++)
    {
        di[i] = diag[i+ki*BI];
        f[i] = pr[i+ki*BI];
    }
    delete diag;
    delete pr;
}
//cf init2[i,j]:=init_gg<KI,N,i,j,NI,K> (out: ia[i,j], ja[i,j], gg[i,j]) |i=1..K,j=1..i;
void init_gg(CodeArgs& in, CodeArgs& out, vector<int>& params)
{
    int KI = params[0];
    int N = params[1];
    int ki = params[2];
    int kj = params[3];
    int NI = params[4];
    int K = params[5];
    int* ia = (int*)out[0].Data;
    int* ja = (int*)out[1].Data;
    double* gg = (double*)out[2].Data;
    int row_beg = (ki-1)*NI;
    int row_end = ki*NI;
    int col_beg = (kj-1)*NI;
    int col_end = kj*NI;
    int i,j,nado = 0;
    int* row = new int [N+1];
    FILE*hp,*fp;
    hp = fopen("ig.txt","r");
```

```
for(i = 0; i < N+1; i++)
       fscanf(hp,"%d",&row[i]);
fclose(hp);
if(row[0] == 1)
{
     nado = 1;
     for(i = 0; i < N+1; i++)
        row[i]--;
}
j = row[N];
int* col = new int[j];
double* ggg = new double [j];
fp = fopen("ggl.txt","r");
hp = fopen("jg.txt","r");
for(i = 0; i < j; i++)
{
       fscanf(hp,"%d",&col[i]);
       fscanf(fp,"%lf",&ggg[i]);
}
fclose(hp);
fclose(fp);
if (nado == 1)
    for(i=0;i < j;i++)   col[i]--;
for(i=0; i<NI+1;i++) ia[i]=0;
int m=0,i1,i2,i3,k,S = row[row_beg];
for(i = 0; i < NI; i++)
{
     i1 = row[row_beg+i];
     i2 = row[row_beg+i+1];
     i3 = i2 - i1;
     for(k = 0; k < i3;k++)
     {
        j = col[k+S];
        if(j < col_end && j >=col_beg)
        {
                 for(int z =i+1; z <= NI; z++)  ia[z]++;
                 gg[m] = ggg[k+S];
                 ja[m] = j;
                 m++;
        }
     }
     S+=i3;
```

```
    }
    ia[0] = 0;
    delete col;
    delete row;
    delete ggg;
}
void init(CodeArgs& in, CodeArgs& out, vector<int>& params)
{
    double* norm1 = (double*)out[0].Data;
    double* norm2 = (double*)out[1].Data;
    double* beta = (double*)out[2].Data;
    double* scal = (double*)out[3].Data;
    double* scal2 = (double*)out[4].Data;
    double* one = (double*)out[4].Data;
    norm1[0] = 0.0;
    norm2[0] = 0.0;
    scal[0] = 0.0;
    scal2[0] = 0.0;
    beta[0] = 0.0;
    one[0] = 1.0;
}
void norma(CodeArgs& in, CodeArgs& out, vector<int>& params)
{
    int bi = params[0];
    double* f = (double*)in[0].Data;
    double* a = (double*)in[1].Data;
    double* nr = (double*)out[0].Data;
    double b = 0.0;
    int i;
    for(i = 0; i < bi; i++)     b+=f[i]*f[i];
    nr[0] = a[0] + b;
}
void mult_di(CodeArgs& in, CodeArgs& out, vector<int>& params)
{
    int n = params[0];
    double* di =(double*)in[0].Data;
    double* x = (double*)in[1].Data;
    double* y = (double*)in[2].Data;
    double* r = (double*)out[0].Data;
    for(int i=0;i<n;i++) r[i]=y[i]+di[i]*x[i];
}
void mult_di_v1(CodeArgs& in, CodeArgs& out, vector<int>& params)
```

```
{
   int n = params[0];
   double* di =(double*)in[0].Data;
   double* x = (double*)in[1].Data;
   double* y = (double*)in[2].Data;
   double* r = (double*)out[0].Data;
   for(int i=0;i<n;i++) r[i]=di[i]*x[i];
}
void mult_gg1(CodeArgs& in, CodeArgs& out, vector<int>& params)
{
   int BI = params[0];  //size block x
   int NI = params[1]; // number row of matrix
   int ki = params[2];//number of block x
   double* x =(double*)in[0].Data;
   int* ia = (int*)in[1].Data;
   int* ja = (int*)in[2].Data;
   double* gg = (double*)in[3].Data;
   double* rk = (double*)in[4].Data;
   double* r =  (double*)out[0].Data;
   int row = (ki-1)*NI;
   int sd = ia[0];
   int n = ia[NI]-ia[0];
   int x_row = (ki-1)*BI;
   int i1,i2,i3,k1=0,j1,j2;
   double h=0.0;
   for(int i=0;i<NI;i++)
   {
          i1=  ia[i];
          i2 = ia[i+1];
          i3 = i2 - i1;
          for(int k=0;k<i3;k++)
          {
               j1= ja[k1+k];
               h = gg[k+k1] * x[j1-x_row];
               r[i]=rk[i]+h;
               h = gg[k+k1] * x[i];
               r[j1-x_row]=rk[j1-x_row] + h;
          }
          k1=k1+i3;
   }
}
void mult_gg2(CodeArgs& in, CodeArgs& out, vector<int>& params)
```

```
{
   int BI = params[0];
   int NI = params[1];
   int ki = params[2];
   int kj=params[3];
   double* x1 =(double*)in[0].Data;
   int* ia = (int*)in[1].Data;
   int* ja = (int*)in[2].Data;
   double* gg =(double*)in[3].Data;
   double* rk1 =(double*)in[4].Data;
   double* rk2 =(double*)in[5].Data;
   double* r1 =(double*)out[0].Data;
   double* r2 =(double*)out[1].Data;
   double* x2 =(double*)in[6].Data;
   int row = (ki-1)*NI;
   int sd = ia[0];
   int n = ia[NI]-ia[0];
    int x_row = (kj-1)*BI;
   int i1,i2,i3,k1=0,j1,j2;
   double h=0.0;
   for(int i=0;i<NI;i++)
   {
          i1=  ia[i];
           i2 = ia[i+1];
           i3 = i2 - i1;
           for(int k=0;k<i3;k++)
           {
                j1= ja[k1+k];
                h = gg[k+k1] * x2[j1-x_row];
                r1[i]=rk1[i]+h;
                h = gg[k+k1] * x1[i];
                r2[j1-x_row]=rk2[j1-x_row] + h;
          }
   k1=k1+i3;
   }
}
void calculate(CodeArgs& in, CodeArgs& out, vector<int>& params)
{
   int n = params[0];
   int num = params[1];
   double * f =(double*)in[0].Data;
   double * b =(double*)in[1].Data;
```

```
   double * c =(double*)in[2].Data;
   double * Ax =(double*)in[3].Data;
   double * r =(double*)out[0].Data;
   int i;
   double alfa;
   alfa = b[0]/c[0];
   if(num==1)
   {
        for(i=0;i<n;i++)
             r[i] = f[i] - alfa*Ax[i];
   } else
   {
        for(i = 0;i < n;i++)
          r[i] = f[i] + alfa*Ax[i];
   }
}
void scalar(CodeArgs& in, CodeArgs& out, vector<int>& params)
{
   int n = params[0];
   int j = params[1];
   double * a =(double*)in[0].Data;
   double * b =(double*)in[1].Data;
   double * c =(double*)in[2].Data;
   double * f =(double*)out[0].Data;
   f[0]=c[0];
   for(int i=0;i<n;i++)    f[0]+=a[i]*b[i];
}
void copy (CodeArgs& in, CodeArgs& out, vector<int>& params)
{
   int n = params[0];
   double * a =(double*)in[0].Data;
   double * b =(double*)out[0].Data;
   for(int i=0;i<n;i++)     b[i]=a[i];
}
void proverka(CodeArgs& in, CodeArgs& out, vector<int>& params).....
{
   int iter = params[0];
   double* norm1 = (double*)in[3].Data;
   double* norm2 = (double*)in[4].Data;
   double* a1 =(double*)out[0].Data;
   double* a2 =(double*)out[1].Data;
   double* a3 =(double*)out[2].Data;
```

```
   double* a4 =(double*)out[3].Data;
   double otn = sqrt(norm2[0]/norm1[0]);
   int code =0;
   if (otn < EPS)   luna::exit(code);
   a1[0] = 0.0;
   a2[0] = 0.0;
   a3[0] = 0.0;
   a4[0] = 0.0;
}
void Init_CodeFrags(map<string, CodeFragment>& cdf)
{
        cdf["init_di"] = init_di;
        cdf["init_gg"] = init_gg;
        cdf["cp"] = cp;
        cdf["mult_di"] = mult_di;
        cdf["mult_gg1"] = mult_gg1;
        cdf["mult_gg2"] = mult_gg2;
        cdf["init_scalar"] = init;
        cdf["norma"] = norma;
        cdf["calculate"] = calculate;
        cdf["scalar"] = scalar;
        cdf["copy"] = copy;
        cdf["proverka"] = proverka;
        cdf["mult_di_v1"] = mult_di_v1;
        cdf["time_start"] = time_start;
}
```

Приложение Б. Текст программы для разбиения на вертикальные полоски

Фрагментированная программа в системе LuNA (без дублирования)

```
//@CODE CodeFrags_no_generate.cpp
const  N = 4545;                    //size matrix
const  K = 5;               //number  block
const  KI = 5;              //number  block di
const  BI = N/KI;           //size block di
const  NI = N/K;                    //number row of block ia
const maxiter = 10;
const SizeOfFloat = 8;
const SizeOfInt = 4;
const P = PE_COUNT;
//-----------------------------------------------------
df di[i]  :=block(SizeOfFloat*BI) | i=1..KI;
df ia[i]  :=block(SizeOfInt*(NI+1))   | i=1..KI;
df ja[i]  :=block(SizeOfInt*(NI*NI*i))   | i=1..KI;
df gg[i]  :=block(SizeOfFloat*(NI*NI*i)) | i=1..KI;
df f[i]   :=block(SizeOfFloat*BI)   | i=1..KI;
df xk[i]  :=block(SizeOfFloat*BI)   | i=1..KI;
df rk[i]  :=block(SizeOfFloat*BI)   | i=1..KI;
df zk[i]  :=block(SizeOfFloat*BI)   | i=1..KI;
df Azk[i] :=block(SizeOfFloat*BI)   | i=1..KI;
df norm1  :=block(SizeOfFloat);
df norm2  :=block(SizeOfFloat);
df beta  := block(SizeOfFloat);
df scal :=block(SizeOfFloat);
df scal2:=block(SizeOfFloat);
df one := block(SizeOfFloat);
//-----------------------------------------------------
cf init[i]:= init_param <N,KI,BI,i>(out: xk[i], di[i], ia[i], ja[i], gg[i], f[i], rk[i], Azk[i]) | i = 1..KI;
cf init_pr:= init_scalar(out:norm1,norm2,beta,scal,scal2,one);
```

```
cf time  := time_start();
cf norm_f[i]:=norma<BI>(in:f[i],norm1; out:norm1) |i=1..KI;
cf mult_di[i] :=mult_di<BI>(in: di[i],xk[i],rk[i];out:rk[i]) | i=1..KI;
cf  mult_gg1[i,j,k]:=mult_gg<j,BI,NI,i,k>(in:  xk[i],  ia[j],  ja[j],  gg[j],  rk[k];  out:rk[k])|
j=1..KI,i=1..j,k=1..j;
cf c_rk[i] := calculate<BI,1>(in: f[i], one_copy[i%P], one_copy[i%P], rk[i]; out:rk[i]) |i=1..KI;
cf c_zk[i] := copy<BI>(in: rk[i]; out:zk[i])| i=1..KI;
//---------------------------------------------------------------------------------
cf rk_rk[i,j]:=scalar<BI,j>(in:rk[j],rk[j],scal;out:scal) | i=1..maxiter,j=1..KI;
cf c_Az[i,j]:=mult_di_v1<BI>(in: di[j], zk[j], Azk[j]; out: Azk[j]) |i=1..maxiter,j=1..KI;
cf  c_Az2[m,i,j,k]:=mult_gg<j,BI,NI,i,k>(in:  zk[i],  ia[j],  ja[j],  gg[j],  Azk[k];  out:  Azk[k])|m  =
1..maxiter,i=1..j,j=1..KI,k=1..j;
cf Azk_zk[i,j]:=scalar<BI,j>(in: Azk[j], zk[j], scal2; out:scal2) | i=1..maxiter,j=1..KI;
cf  c_xk1[i,j]:=calculate<BI,0>(in:  xk[j],  scal_copy[i,j%P],  scal2_copy[i,j%P],  zk[j];  out:xk[j])
|i=1..maxiter,j=1..KI;
cf  c_rk1[i,j]:=calculate<BI,1>(in:  rk[j],  scal_copy[i,j%P],  scal2_copy[i,j%P],  Azk[j];  out:rk[j])
|i=1..maxiter,j=1..KI;
cf rk1_rk1[i,j]:=scalar<BI>(in:rk[j],rk[j],beta;out:beta) | i=1..maxiter,j=1..KI;
cf  c_zk1[i,j]:=calculate<BI,0>(in:  rk[j],  beta_copy[i,j%P],  scal_copy[i,j%P],  zk[j];  out:  zk[j])
|i=1..maxiter,j=1..KI;
cf norm_rk1[i,j] := norma<BI>(in: rk[j], norm2; out:norm2) | i=1..maxiter,j=1..KI;
cf  nul[i]:=proverka<i>  (in:scal,  scal2,  beta,  norm1,  norm2;  out:  scal,  scal2,  beta,  norm2)
|i=1..maxiter;
cf fin := _abort();
//**************************
init[i] < init[i+1] | i = 1..KI-1;
init[i] < mult_di[i] | i = 1..KI;
init[j] < mult_gg1[i,j,k]|  j=1..KI,i=1..j,k=1..j;
init[i] < time | i =1..KI;
init_pr < time;
time < norm_f[i] | i = 1..KI;
time < mult_di[i] | i = 1..KI;
```

time < mult_gg1[i,j,k] |j=1..KI,i=1..j,k=1..j;

init[i] < norm_f[i] | i = 1..KI;

init_pr < norm_f[i] | i = 1..KI;

init_pr < c_rk[i] | i = 1..KI;

mult_di[i] < c_rk[i] | i = 1..KI;

mult_gg1[j,i,k] < c_rk[k] | i = 1..KI,j=1..i,k=1..i;

c_rk[i] < c_zk[i] | i =1..KI;

c_rk[i] < rk_rk[1,i] | i =1..KI;

nul[i-1] < rk_rk[i,j] | i = 2..maxiter, j = 1..KI;

rk_rk[j,i] < rk_rk[j+1,i] |j=1..maxiter-1, i =1..KI;

c_zk[j] < c_Az[1,j] | j=1..KI;

nul[i-1] < c_Az[i,j] | i = 2..maxiter,j = 1..KI;

c_zk[i] < c_Az2[1,i,j,k]| i=1..j,j=1..KI,k=1..j;

nul[m-1] < c_Az2[m,i,j,k]| m = 2..maxiter-1,i = 1..j,j=1..KI, k =1..j;

c_Az[i,j] < c_Az[i+1,j] | i=1..maxiter-1, j =1..KI;

c_Az2[m,i,j,k] < c_Az2[m+1,i,j,k] |m = 1..maxiter-1,i=1..j,j=1..KI,k=1..j;

c_Az[m,k] < c_Az2[m,i,j,k] |m = 1..maxiter,j=1..KI,i=1..j,k=1..j;

c_Az[i,j] < Azk_zk[i,j] |i = 1..maxiter,j=1..KI;

c_Az2[m,i,j,k] < Azk_zk[m,k] | m = 1..maxiter,i=1..j,j=1..KI,k=1..j;

Azk_zk[i,j] < Azk_zk[i+1,j] |i = 1..maxiter-1,j=1..KI;

Azk_zk[i,j] < c_xk1[i,k] |i = 1..maxiter, j=1..KI, k = 1..KI;

rk_rk[i,j] < c_xk1[i,k] | i = 1..maxiter, j=1..KI, k = 1..KI;

Azk_zk[i,j] < c_rk1[i,k] |i = 1..maxiter, j=1..KI, k = 1..KI;

rk_rk[i,j] < c_rk1[i,k] | i = 1..maxiter, j=1..KI, k = 1..KI;

c_xk1[i,j] < c_xk1[i+1,j] | i = 1..maxiter-1,j=1..KI;

c_rk1[i,j] < c_rk1[i+1,j] | i = 1..maxiter-1,j =1..KI;

c_rk1[i,j] < rk1_rk1[i,j] | i = 1..maxiter, j=1..KI;

rk1_rk1[i,j] < rk1_rk1[i+1,j] | i = 1..maxiter-1, j=1..KI;

rk1_rk1[i,j] < c_zk1[i,k] | i = 1..maxiter,j=1..KI,k =1..KI;

c_zk1[i,j] < c_zk1[i+1,j] | i = 1..maxiter-1, j=1..KI;

c_rk1[i,j] < norm_rk1[i,j] | i = 1..maxiter,j=1..KI;

norm_rk1[i,j] < norm_rk1[i+1,j] |i = 1..maxiter-1,j=1..KI;

```
norm_rk1[i,j] < nul[i] | i = 1..maxiter, j = 1..KI;
norm_f[j] < nul[i] | i = 1..maxiter,j = 1..KI;
c_zk1[i,j] < nul[i] | i =1..maxiter,j=1..KI;
nul[i] < nul[i+1] | i = 1..maxiter-1;
nul[maxiter] < fin;
//-----------------------------------------------------
init[i].location = 0 | i = 1..KI;
mult_di[i].location = i%PE_COUNT | i = 1..KI;
mult_gg1[i,j,k].location = j%PE_COUNT| j=1..KI,i=1..j,k=1..j;
init_pr.location = 0;
norm_f[i].location = 0 | i = 1..KI;
c_rk[j].location = j%PE_COUNT | j = 1..KI;
c_zk[j].location = j%PE_COUNT | j = 1..KI;
rk_rk[i,j].location = 0| i = 1..maxiter,j=1..KI;
c_Az[i,j].location = j%PE_COUNT| i = 1..maxiter,j=1..KI;
c_Az2[m,i,j,k].location = j%PE_COUNT |m = 1..maxiter,j=1..KI,i=1..j,k=1..j;
Azk_zk[i,j].location = 0 | i = 1..maxiter,j=1..KI;
c_xk1[i,j].location = j%PE_COUNT| i = 1..maxiter,j=1..KI;
c_rk1[i,j].location = j%PE_COUNT| i = 1..maxiter,j=1..KI;
rk1_rk1[i,j].location = 0 | i = 1..maxiter,j=1..KI;
c_zk1[i,j].location = j%PE_COUNT| i = 1..maxiter,j=1..KI;
norm_rk1[i,j].location = 0| i = 1..maxiter,j=1..KI;
nul[i].location = 0 | i =1..maxiter;
fin.location = 0;
time.location = 0;
```

Фрагментированная программа в системе LuNA (с дублированием)

```
//@CODE CodeFrags_no_generate.cpp
const  N = 4545;                    //size matrix
const  K = 5;               //number  block
const  KI = 5;              //number  block di
const  BI = N/KI;           //size block di
const  NI = N/K;                    //number row of block ia
```

```
const maxiter = 10;
const SizeOfFloat = 8;
const SizeOfInt = 4;
const P = PE_COUNT;
//-----------------------------------------------------
df di[i]  :=block(SizeOfFloat*BI) | i=1..KI;
df ia[i]  :=block(SizeOfInt*(NI+1))   | i=1..KI;
df ja[i]  :=block(SizeOfInt*(NI*NI*i))   | i=1..KI;
df gg[i]  :=block(SizeOfFloat*(NI*NI*i)) | i=1..KI;
df f[i]   :=block(SizeOfFloat*BI)   | i=1..KI;
df xk[i]  :=block(SizeOfFloat*BI)   | i=1..KI;
df rk[i]  :=block(SizeOfFloat*BI)   | i=1..KI;
df zk[i]  :=block(SizeOfFloat*BI)   | i=1..KI;
df Azk[i] :=block(SizeOfFloat*BI)   | i=1..KI;
df norm1  :=block(SizeOfFloat);
df norm2  :=block(SizeOfFloat);
df beta  := block(SizeOfFloat);
df beta_copy[j,i]:= block(SizeOfFloat) | j=1..maxiter,i=0..P-1; .....
df scal :=block(SizeOfFloat);
df scal_copy[j,i] := block(SizeOfFloat) |j=1..maxiter, i=0..P-1;.....
df scal2:=block(SizeOfFloat);
df scal2_copy[j,i] := block(SizeOfFloat) | j=1..maxiter,i=0..P-1;.....
df one := block(SizeOfFloat);
df one_copy[i]:= block(SizeOfFloat) | i=0..P-1;
//-----------------------------------------------------
cf init[i]:= init_param <N,KI,BI,i>(out: xk[i], di[i], ia[i], ja[i], gg[i], f[i], rk[i], Azk[i]) | i = 1..KI;
cf init_pr:= init_scalar(out:norm1,norm2,beta,scal,scal2,one);
cf one_cp[i]:= cp (in:one;out:one_copy[i]) | i=0..P-1;
cf time  := time_start();
cf norm_f[i]:=norma<BI>(in:f[i],norm1; out:norm1) |i=1..KI;
cf mult_di[i] :=mult_di<BI>(in: di[i],xk[i],rk[i];out:rk[i]) | i=1..KI;
```

```
cf mult_gg1[i,j,k]:=mult_gg<j,BI,NI,i,k>(in: xk[i], ia[j], ja[j], gg[j], rk[k]; out:rk[k])|
j=1..KI,i=1..j,k=1..j;
cf c_rk[i] := calculate<BI,1>(in: f[i], one_copy[i%P], one_copy[i%P], rk[i]; out:rk[i]) |i=1..KI;
cf c_zk[i] := copy<BI>(in: rk[i]; out:zk[i])| i=1..KI;
//---------------------------------------------------------------------------------
cf rk_rk[i,j]:=scalar<BI,j>(in:rk[j],rk[j],scal;out:scal) | i=1..maxiter,j=1..KI;
cf scal_cp[i,j]:= cp(in:scal;out:scal_copy[i,j]) | i=1..maxiter,j=0..P-1;
cf c_Az[i,j]:=mult_di_v1<BI>(in: di[j], zk[j], Azk[j]; out: Azk[j]) |i=1..maxiter,j=1..KI;
cf c_Az2[m,i,j,k]:=mult_gg<j,BI,NI,i,k>(in: zk[i], ia[j], ja[j], gg[j], Azk[k]; out: Azk[k])|m =
1..maxiter,i=1..j,j=1..KI,k=1..j;
cf Azk_zk[i,j]:=scalar<BI,j>(in: Azk[j], zk[j], scal2; out:scal2) | i=1..maxiter,j=1..KI;
cf scal2_cp[i,j]:= cp(in:scal2;out:scal2_copy[i,j]) | i=1..maxiter,j=0..P-1;
cf c_xk1[i,j]:=calculate<BI,0>(in: xk[j], scal_copy[i,j%P], scal2_copy[i,j%P], zk[j]; out:xk[j])
|i=1..maxiter,j=1..KI;
cf c_rk1[i,j]:=calculate<BI,1>(in: rk[j], scal_copy[i,j%P], scal2_copy[i,j%P], Azk[j]; out:rk[j])
|i=1..maxiter,j=1..KI;
cf rk1_rk1[i,j]:=scalar<BI>(in:rk[j],rk[j],beta;out:beta) | i=1..maxiter,j=1..KI;
cf beta_cp[i,j]:= cp(in:beta;out:beta_copy[i,j]) | i=1..maxiter,j=0..P-1;
cf c_zk1[i,j]:=calculate<BI,0>(in: rk[j], beta_copy[i,j%P], scal_copy[i,j%P], zk[j]; out: zk[j])
|i=1..maxiter,j=1..KI;
cf norm_rk1[i,j] := norma<BI>(in: rk[j], norm2; out:norm2) | i=1..maxiter,j=1..KI;
cf nul[i]:=proverka<i> (in:scal, scal2, beta, norm1, norm2; out: scal, scal2, beta, norm2)
|i=1..maxiter;
cf fin := _abort();
//**************************
init[i] < init[i+1] | i = 1..KI-1;
init[i] < mult_di[i] | i = 1..KI;
init[j] < mult_gg1[i,j,k]|  j=1..KI,i=1..j,k=1..j;
init[i] < time | i =1..KI;
init_pr < time;
time < norm_f[i] | i = 1..KI;
time < mult_di[i] | i = 1..KI;
```

time < mult_gg1[i,j,k] |j=1..KI,i=1..j,k=1..j;

init[i] < norm_f[i] | i = 1..KI;

init_pr < norm_f[i] | i = 1..KI;

init_pr < one_cp[i] | i = 0..P-1;

one_cp[i%P] < c_rk[i] | i = 1..KI;

mult_di[i] < c_rk[i] | i = 1..KI;

mult_gg1[j,i,k] < c_rk[k] | i = 1..KI,j=1..i,k=1..i;

c_rk[i] < c_zk[i] | i =1..KI;

c_rk[i] < rk_rk[1,i] | i =1..KI;

nul[i-1] < rk_rk[i,j] | i = 2..maxiter, j = 1..KI;

rk_rk[j,i] < rk_rk[j+1,i] |j=1..maxiter-1, i =1..KI;

rk_rk[i,j] < scal_cp[i,m] | i=1..maxiter,j=1..KI,m=0..P-1;

c_zk[j] < c_Az[1,j] | j=1..KI;

nul[i-1] < c_Az[i,j] | i = 2..maxiter,j = 1..KI;

c_zk[i] < c_Az2[1,i,j,k]| i=1..j,j=1..KI,k=1..j;

nul[m-1] < c_Az2[m,i,j,k]| m = 2..maxiter-1,i = 1..j,j=1..KI, k =1..j;

c_Az[i,j] < c_Az[i+1,j] | i=1..maxiter-1, j =1..KI;

c_Az2[m,i,j,k] < c_Az2[m+1,i,j,k] |m = 1..maxiter-1,i=1..j,j=1..KI,k=1..j;

c_Az[m,k] < c_Az2[m,i,j,k] |m = 1..maxiter,j=1..KI,i=1..j,k=1..j;

c_Az[i,j] < Azk_zk[i,j] |i = 1..maxiter,j=1..KI;

c_Az2[m,i,j,k] < Azk_zk[m,k] | m = 1..maxiter,i=1..j,j=1..KI,k=1..j;

Azk_zk[i,j] < Azk_zk[i+1,j] |i = 1..maxiter-1,j=1..KI;

Azk_zk[i,j] < scal2_cp[i,m] |i=1..maxiter,j=1..KI,m=0..P-1;

scal2_cp[i,k%P] < c_xk1[i,k] |i = 1..maxiter, k = 1..KI;

scal_cp[i,k%P] < c_xk1[i,k] | i = 1..maxiter, k = 1..KI;

scal2_cp[i,k%P] < c_rk1[i,k] |i = 1..maxiter, k = 1..KI;

scal_cp[i,k%P] < c_rk1[i,k] | i = 1..maxiter, k = 1..KI;

c_xk1[i,j] < c_xk1[i+1,j] | i = 1..maxiter-1,j=1..KI;

c_rk1[i,j] < c_rk1[i+1,j] | i = 1..maxiter-1,j =1..KI;

c_rk1[i,j] < rk1_rk1[i,j] | i = 1..maxiter, j=1..KI;

rk1_rk1[i,j] < rk1_rk1[i+1,j] | i = 1..maxiter-1, j=1..KI;

rk1_rk1[i,j] < beta_cp[i,m] | i=1..maxiter,j=1..KI,m=0..P-1;

```
beta_cp[i,k%P] < c_zk1[i,k] | i = 1..maxiter,k =1..KI;
c_zk1[i,j] < c_zk1[i+1,j] | i = 1..maxiter-1, j=1..KI;
c_rk1[i,j] < norm_rk1[i,j] | i = 1..maxiter,j=1..KI;
norm_rk1[i,j] < norm_rk1[i+1,j] |i = 1..maxiter-1,j=1..KI;
norm_rk1[i,j] < nul[i] | i = 1..maxiter, j = 1..KI;
norm_f[j] < nul[i] | i = 1..maxiter,j = 1..KI;
c_zk1[i,j] < nul[i] | i =1..maxiter,j=1..KI;
nul[i] < nul[i+1] | i = 1..maxiter-1;
nul[maxiter] < fin;
//-----------------------------------------------------
init[i].location = 0 | i = 1..KI;
mult_di[i].location = i%PE_COUNT | i = 1..KI;
mult_gg1[i,j,k].location = j%PE_COUNT| j=1..KI,i=1..j,k=1..j;
init_pr.location = 0;
norm_f[i].location = 0 | i = 1..KI;
c_rk[j].location = j%PE_COUNT | j = 1..KI;
c_zk[j].location = j%PE_COUNT | j = 1..KI;
rk_rk[i,j].location = 0| i = 1..maxiter,j=1..KI;
c_Az[i,j].location = j%PE_COUNT| i = 1..maxiter,j=1..KI;
c_Az2[m,i,j,k].location = j%PE_COUNT |m = 1..maxiter,j=1..KI,i=1..j,k=1..j;
Azk_zk[i,j].location = 0 | i = 1..maxiter,j=1..KI;
c_xk1[i,j].location = j%PE_COUNT| i = 1..maxiter,j=1..KI;
c_rk1[i,j].location = j%PE_COUNT| i = 1..maxiter,j=1..KI;
rk1_rk1[i,j].location = 0 | i = 1..maxiter,j=1..KI;
c_zk1[i,j].location = j%PE_COUNT| i = 1..maxiter,j=1..KI;
norm_rk1[i,j].location = 0| i = 1..maxiter,j=1..KI;
nul[i].location = 0 | i =1..maxiter;
fin.location = 0;
time.location = 0;
one_cp[i].location = 0 | i=0..P-1;
scal_cp[i,j].location = 0 | i=1..maxiter,j=0..P-1;
scal2_cp[i,j].location = 0 | i=1..maxiter,j=0..P-1;
```

```
beta_cp[i,j].location = 0 | i=1..maxiter,j=0..P-1;
```

Code_no_generate. cpp

```
#include <iostream>
#include <stdlib.h>
#include <math.h>
#include <stdio.h>
#include "iCodeFrag.h"
#include "iLuna.h"
#include <sys/time.h>
using namespace std;
using namespace luna;
struct timeval tv1;
#define BBI 4545/1
#define KKK 1
#define EPS 1e-6
void time_start(CodeArgs& in, CodeArgs& out, vector<int>& params)
{
   gettimeofday(&tv1,NULL);
}
void cp(CodeArgs& in, CodeArgs& out, vector<int>& params)
{
   double* a =(double*)in[0].Data;
   double* b = (double*)out[0].Data;
   b[0] = a[0];
}
void PostProcessDFs()
{
        int rank;
        MPI_Comm_rank(MPI_COMM_WORLD,&rank);
        if(rank == 0)
        {
            struct timeval tv2,dtv;
            gettimeofday(&tv2,NULL);
            dtv.tv_sec = tv2.tv_sec - tv1.tv_sec;
            dtv.tv_usec = tv2.tv_usec - tv1.tv_usec;
            if(dtv.tv_usec < 0) {dtv.tv_sec--; dtv.tv_usec+=1000000;}
            printf("Time: %lf\n",dtv.tv_sec+dtv.tv_usec*1e-6);
        }
        FILE*fp;
        fp = fopen("rez_kubik.txt","w");
        for(int i = 1; i <= KKK;i++)
```

```
        {
           fInstance df("xk");
           df.index.push_back(i); // 1st index is '2'
           double*p = (double*) luna::getDF(df);
           if(p!=NULL)
           {
               for(int j = 0;j < BBI; j++)
               fprintf(fp,"% .15lf \n", p[j]);
           }
        }
fclose(fp);
}
void init_param(CodeArgs& in, CodeArgs& out, vector<int>& params)
{
   int n = params[0];
   int ki = params[1];
   int bi = params[2];
   int k = params[3];
   double* xk = (double*)out[0].Data;
   double* di = (double*)out[1].Data;
   int* ia = (int*)out[2].Data;
   int* ja = (int*)out[3].Data;
   double* gg = (double*)out[4].Data;
   double* f = (double*)out[5].Data;
   double* temp = (double*)out[6].Data;
   double* Azk = (double*)out[7].Data;
   int i,j;
   for(i = 0; i < bi; i++)
   {
         temp[i] = 0.0;
         xk[i] =  2.0;
         Azk[i] = 0.0;
   }
   k = k-1;
   double* diag = new double [n];
   int* row = new int [n+1];
   double* pr = new double [n];
   FILE* hp;
   hp = fopen("di.txt","r");
   for(i = 0; i < n; i++)  fscanf(hp,"%lf",&diag[i]);
   fclose(hp);
   hp=fopen("f_1.txt","r");
```

```
   for(i = 0;i < n; i++)  fscanf(hp,"%lf",&pr[i]);
   fclose(hp);
  for(i = 0;i < bi; i++)
   {
         di[i] = diag[i+k*bi];
         f[i] = pr[i+k*bi];
   }
   delete diag;
   delete pr;
   int nado=0;
   hp=fopen("ig.txt","r");
   for(i = 0;i < n+1; i++) fscanf(hp,"%d",&row[i]);
   fclose(hp);
   if(row[0]==1)
   {
       nado = 1;
       for(i = 0; i < n+1; i++)    row[i]--;
   }
   ia[0] = row[k*bi];
   for(i = 0;i < bi;i++)  ia[i+1] = row[k*bi+i+1];
   j=row[n];
   int* col = new int [j];
   double* ggl = new double[j];
   hp=fopen("jg.txt","r");
   for (i=0;i<j;i++)
         fscanf(hp,"%d",&col[i]);
   fclose(hp);
   if(nado) for(i=0;i<j;i++) col[i]--;
   hp=fopen("ggl.txt","r");
   for (i=0;i<j;i++)    fscanf(hp,"%lf",&ggl[i]);
   fclose(hp);
   int Size = row[(k+1)*bi] - row[k*bi];
   for(i = 0; i < Size; i++)
   {
         ja[i] = col[i+ia[0]];
         gg[i] = ggl[i+ia[0]];
   }
   delete ggl;
   delete row;
   delete col;
}
void init(CodeArgs& in, CodeArgs& out, vector<int>& params)
```

```
{
   double* norm1 = (double*)out[0].Data;
   double* norm2 = (double*)out[1].Data;
   double* beta = (double*)out[2].Data;
   double* scal = (double*)out[3].Data;
   double* scal2 = (double*)out[4].Data;
   double* one = (double*)out[4].Data;
   norm1[0] = 0.0;
   norm2[0] = 0.0;
   scal[0] = 0.0;
   scal2[0] = 0.0;
   beta[0] = 0.0;
   one[0] = 1.0;
}
void norma(CodeArgs& in, CodeArgs& out, vector<int>& params)
{
   int bi = params[0];
   double* f = (double*)in[0].Data;
   double* a = (double*)in[1].Data;
   double* nr = (double*)out[0].Data;
   double b = 0.0;
   int i;
   for(i = 0; i < bi; i++)     b+=f[i]*f[i];
   nr[0] = a[0] + b;
}
void mult_di(CodeArgs& in, CodeArgs& out, vector<int>& params)
{
   int n = params[0];
   double* di =(double*)in[0].Data;
   double* x = (double*)in[1].Data;
   double* y = (double*)in[2].Data;
   double* r = (double*)out[0].Data;
   for(int i=0;i<n;i++) r[i]=y[i]+di[i]*x[i];
}
void mult_di_v1(CodeArgs& in, CodeArgs& out, vector<int>& params)
{
   int n = params[0];
   double* di =(double*)in[0].Data;
   double* x = (double*)in[1].Data;
   double* y = (double*)in[2].Data;
   double* r = (double*)out[0].Data;
   for(int i=0;i<n;i++) r[i]=di[i]*x[i];
```

```
}
void mult_gg(CodeArgs& inputs, CodeArgs& outputs, vInt& params)
{
   int j = params[0];                  //number of  block matrix
   int S = params[1];       //size block x
   int NI = params[2];    // number row of matrix
   int k_x=params[3];     //number of block x
   int k_r=params[4];    //number of block rez
  double* x =(double*)inputs[0].Data;
   int* ia = (int*)inputs[1].Data;
   int* ja = (int*)inputs[2].Data;
   double* gg =(double*)inputs[3].Data;
   double* rk =(double*)inputs[4].Data;
   double* r =(double*)outputs[0].Data;
  int sd = ia[0];
   int row = (j-1)*NI;
   int n = ia[NI]-ia[0];
   int x_row = (k_x-1)*S;
   int r_row = (k_r-1)*S;
   int i1,i2,i3,k1=0,j1,j2;
   double h=0.0;
   for(int i=0;i<NI;i++)
   {
          i1=  ia[i];
          i2 = ia[i+1];
          i3 = i2 - i1;
          for(int k=0;k<i3;k++)
          {
               j1= ja[k1+k];
               if(j1>=x_row && j1<(x_row+S))
               {
                     if((row+i)>=r_row && (row+i)<(r_row+S))
                    {
                          h = gg[k+k1] * x[j1-x_row];
                          r[i]=rk[i]+h;
                     }
              }
               if((row+i)>=x_row && (row+i)<(x_row+S))
              {
                    if(j1>=r_row && j1<(r_row+S))
                    {
                           h = gg[k+k1] * x[i];
```

```
                        r[j1-r_row]=rk[j1-r_row] + h;
                    }
                }
            }
    k1=k1+i3;
    }
}
void calculate(CodeArgs& in, CodeArgs& out, vector<int>& params)
{
    int n = params[0];
    int num = params[1];
    double * f =(double*)in[0].Data;
    double * b =(double*)in[1].Data;
    double * c =(double*)in[2].Data;
    double * Ax =(double*)in[3].Data;
    double * r =(double*)out[0].Data;
    int i;
    double alfa;
    alfa = b[0]/c[0];
    if(num==1)
    {
        for(i=0;i<n;i++)
            r[i] = f[i] - alfa*Ax[i];
    } else
    {
        for(i = 0;i < n;i++)
           r[i] = f[i] + alfa*Ax[i];
    }
}
void scalar(CodeArgs& in, CodeArgs& out, vector<int>& params)
{
    int n = params[0];
    int j = params[1];
    double * a =(double*)in[0].Data;
    double * b =(double*)in[1].Data;
    double * c =(double*)in[2].Data;
    double * f =(double*)out[0].Data;
    f[0]=c[0];
    for(int i=0;i<n;i++)    f[0]+=a[i]*b[i];
}
void copy (CodeArgs& in, CodeArgs& out, vector<int>& params)
{
```

```
   int n = params[0];
   double * a =(double*)in[0].Data;
   double * b =(double*)out[0].Data;
   for(int i=0;i<n;i++)     b[i]=a[i];
}
void proverka(CodeArgs& in, CodeArgs& out, vector<int>& params).....
{
   int iter = params[0];
   double* norm1 = (double*)in[3].Data;
   double* norm2 = (double*)in[4].Data;
   double* a1 =(double*)out[0].Data;
   double* a2 =(double*)out[1].Data;
   double* a3 =(double*)out[2].Data;
   double* a4 =(double*)out[3].Data;
   double otn = sqrt(norm2[0]/norm1[0]);
   int code =0;
   if (otn < EPS)   luna::exit(code);
   a1[0] = 0.0;
   a2[0] = 0.0;
   a3[0] = 0.0;
   a4[0] = 0.0;
}
void Init_CodeFrags(map<string, CodeFragment>& cdf)
{
        cdf["init_param"] = init_param;
        cdf["cp"] = cp;
        cdf["mult_di"] = mult_di;
        cdf["mult_gg"] = mult_gg;
        cdf["init_scalar"] = init;
        cdf["norma"] = norma;
        cdf["calculate"] = calculate;
        cdf["scalar"] = scalar;
        cdf["copy"] = copy;
        cdf["proverka"] = proverka;
        cdf["mult_di_v1"] = mult_di_v1;
        cdf["time_start"] = time_start;
}
```