

Московский государственный университет имени М. В. Ломоносова
Факультет вычислительной математики и кибернетики
Кафедра суперкомпьютеров и квантовой информатики

ТОЙГИЛЬДИН Владислав Петрович

**Разработка и исследование параллельного
алгоритма поиска неточных повторов в геноме.**

ДИПЛОМНАЯ РАБОТА

Научный руководитель
к.ф.-м.н., доцент
Н.Н. Попова

Москва, 2015

Работа посвящена использованию суперкомпьютерных технологий для обработки биологических последовательностей. Рассмотрен спектрально-аналитический метод поиска протяженных неточных повторов. Предложен и реализован параллельный алгоритм поиска протяженных неточных повторов. Применение разработанного алгоритма продемонстрировано на решении задачи сравнения биологических последовательностей двух организмов (*Mus musculus* и *Rattus norvegicus*). Объем входных последовательностей составлял 165Мб и 140Мб. Проведено исследование масштабируемости и эффективности разработанного алгоритма на высокопроизводительных системах Ломоносов и BlueGene/P.

Содержание

| | |
|---|-----------|
| Введение | 2 |
| 1 Исследование и построение решения | 4 |
| 1.1 Математическая модель поиска повторов в биологических последовательностях | 4 |
| 1.2 Спектрально-аналитический метод поиска повторов | 5 |
| 1.2.1 Профили биологических последовательностей | 5 |
| 1.2.2 Спектральное индексирование профилей | 6 |
| 1.2.3 Спектральное сравнение профилей | 6 |
| 1.2.4 Анализ гомологической матрицы | 7 |
| 1.3 Параллельный метод поиска повторов | 9 |
| 1.3.1 Профилирование и спектральное индексирование | 9 |
| 1.3.2 Спектральное сравнение и анализ матрицы гомологии. Матричный метод | 10 |
| 1.3.3 Спектральное сравнение и анализ матрицы гомологии. Блочный метод | 11 |
| 2 Структура программной реализации | 14 |
| 3 Результаты вычислительных экспериментов | 18 |
| 3.1 Сравнение блочной программы с матричной | 19 |
| 3.2 Использование графических процессоров | 20 |
| 3.3 Масштабируемость параллельного алгоритма | 22 |
| 3.4 Верификация полученных данных | 24 |
| Заключение | 25 |
| Список литературы | 26 |

Введение

С появлением в начале XXI века новых методов секвенирования (расшифровки биологических последовательностей) резко увеличился объем генетической информации. Рост банков данных носит экспоненциальный характер и есть все основания предполагать, что в ближайшее время этот процесс только усилится [11]. Увеличивающийся объем данных открывает возможности для проведения полномасштабных исследований на уровне целых геномов, однако это требует новых подходов. Дело в том, что почти все существующие алгоритмы обработки биологических последовательностей - это адаптированные алгоритмы обработки текстовых строк, т.к. биологические данные в расшифрованном виде представляют собой последовательности символов. Однако данные алгоритмы изначально не учитывают мутационных процессов, таких как вставка, замена, делеция. Учет же таких точечных мутаций является дорогой с вычислительной точки зрения операцией, вносящей существенную нелинейность в подобные алгоритмы, что значительно увеличивает время их выполнения и делает непригодными для практических целей.

Все это привело к тому, что существующие вычислительные мощности не удовлетворяют потребностям биологов. Становится актуальной разработка программных средств, позволяющих быстро обрабатывать большие биологические данные. Данная дипломная работа посвящена параллельным методам анализа и обработки биологических последовательностей на суперкомпьютерах с целью сокращения времени обработки и увеличения объема обрабатываемых данных.

Одной из частных задач молекулярной генетики является поиск повторяющихся элементов, изучение их структуры и распределения в биологических последовательностях. Повторы играют важную роль в функционировании организма, т.к. составляют значительную часть генома, возможно участвуют в его реорганизации и при попадании в кодирующие области повторы могут быть причиной нарушения функций этих геномов, что ведет к развитию заболеваний [5]. Таким образом повторы могут использоваться для диагностики генетических заболеваний и определения родства организмов.

Существует масса алгоритмов, ориентированных на поиск относительно точных или коротких (до 500 нуклеотидных пар) повторов. Однако именно протяженные (от 1000 н.п.) повторы могут служить источником для фундаментальных исследований при решении эволюционных и филогенетических задач - определении родства групп организмов на геномном уровне. Для полногеномного сравнения на практике часто применяется ДНК-ДНК гибридизация - биохимическая реакция, протекающая с участием ДНК двух организмов, проводимая для получения количественной оценки их схожести. Однако данный метод очень дорогостоящий, долгий и недостаточно точный [6].

В данной работе рассматривается метод поиска протяженных неточных повторов, разработанный коллективом кафедры математических методов прогнозирования и института математических проблем в биологии Российской академии наук (Пушкино) [3]. Данный метод позволяет эффективно решать задачу поиска неточных протяженных повторяющихся структур в генетических текстах. Основное отличие данного метода состоит в переходе от дискретного анализа к непрерывному анализу. Таким образом для задачи дискретной по своей природе, применяется континуальный подход, основанный на приближении непрерывных функций с

помощью ортогональных многочленов. Благодаря этому метод решает недостатки дискретного подхода и обладает следующими свойствами:

- Линейная сложность алгоритма
- Высокая степень устойчивости к мутациям
- Возможность распараллеливания алгоритма на кластерных системах

Для выполнения дипломной работы необходимо было решить следующие задачи:

- Разработать и реализовать параллельный алгоритм поиска повторов, ориентированного на суперкомпьютерную реализацию: большой объем обрабатываемых данных (размер данных порядка 1 ГБ), время обработки в пределах 1 часа.
- Исследовать эффективность использования графических процессоров для решения поставленной задачи.
- Анализировать масштабируемость разработанного алгоритма на примере задачи сравнения конкретных геномов
- Разработать графический интерфейс для работы пользователя на локальной и удаленных системах.

Дипломная работа состоит из трех глав.

Первая глава посвящена обзору математической модели повторов, разобрана схема работы спектрально-аналитического метода и предлагается параллельный алгоритм.

Во второй главе показана структура программной реализации, рассмотрены основные возможности программной реализации.

В третьей главе приведены результаты тестирования разработанного алгоритма.

В заключении приведены основные результаты работы и предложены перспективы ее дальнейшего развития.

Глава 1

Исследование и построение решения

1.1 Математическая модель поиска повторов в биологических последовательностях

Формально биологическая последовательность:

$$X = (x_n)_{n=1}^N, \quad N \in \mathbb{N}, \quad x_n \in \{A, T, G, C\},$$

где $N = |X|$ - длина последовательности, A, T, G, C - обозначения нуклеотидов.

Введем обозначение. **Подпоследовательностью** $X|_i^k$ последовательности X называется часть последовательности X с элемента с номером i длиной k при условии $i + k \leq |X|$:

$$X|_i^k = \{x_i, x_{i+1}, \dots, x_{i+k-1}\}$$

Под **повтором** будем понимать пару последовательностей (X_1, X_2) для которых справедливо неравенство:

$$\rho(X_1, X_2) \leq \varepsilon, \quad |X_1| = |X_2| = K$$

где:

- $\rho(X_1, X_2)$ - расстояние редактирования, оценка близости последовательностей, Конкретный вид функции расстояния редактирования $\rho(X_1, X_2)$ **определяется алгоритмом**
- ε - задаваемая точность поиска, значение которой будет **зависеть от задаваемой функции расстояния редактирования**
- K - длина последовательностей

Пусть есть две последовательности $X = (x_n)_{n=1}^{N_x}$ и $Y = (y_n)_{n=1}^{N_y}$.

Под **задачей поиска повторов** будем понимать нахождение всех троек $\{i_x, i_y, k\}$, $i_x, i_y, k \in \mathbb{N}$, таких что:

$$\begin{aligned} i_x + k &\leq N_x \\ i_y + k &\leq N_y \\ (X|_{i_x}^k, Y|_{i_y}^k) &- \text{повтор длины } k \end{aligned}$$

То есть задача найти все такие подпоследовательности последовательностей X и Y , что эти подпоследовательности будут повторами.

1.2 Спектрально-аналитический метод поиска повторов

В работе исследуется спектральный метод поиска повторов в биологических последовательностях предложенный и разработанный коллективом кафедры математических методов прогнозирования ВМК МГУ и института математических проблем в биологии Российской академии наук (Пушино).

Метод разбивается на четыре основных этапа:

1. Получение профилей биологических последовательностей
2. Спектральное индексирование полученных профилей
3. Сравнение коэффициентов спектрального индексирования и построение гомологической матрицы
4. Анализ гомологической матрицы

1.2.1 Профили биологических последовательностей

Одной из главных особенностей рассматриваемого метода поиска повторов является то, что на первом этапе последовательность преобразуется из дискретной в непрерывную область. Это достигается построением т.н. профилей.

Под GC-профилем последовательности $X = (x_n)_{n=1}^{N_x}$ с окном w в дальнейшем будем понимать такую последовательность $P_{GC}(X, w) = (p_n^{GC})_{n=1}^{N_p}$, $N_p = N_x - w + 1$, что

$$p_i^{GC} = \sum_{k=i}^{i+w} I^{GC}(x_k), i = \overline{1, N_p}$$

где

$$I^{GC} = \begin{cases} 1, & x_n \in \{G, C\} \\ 0, & \text{иначе} \end{cases}, n = \overline{1, N_x}$$

Понятие GA-профиля определяется аналогично:

$$P_{GA}(X, w) = (p_n^{GA})_{n=1}^{N_p} : p_i^{GA} = \sum_{k=i}^{i+w} I^{GA}(x_k), i = \overline{1, N_p},$$

где

$$I^{GA} = \begin{cases} 1, & x_n \in \{G, C\} \\ 0, & \text{иначе} \end{cases}, n = \overline{1, N_x}$$

1.2.2 Спектральное индексирование профилей

На этом этапе профили переводятся в спектральное представление с использованием в качестве базиса полиномов Чебышева дискретного аргумента.

Под спектральным представлением сигнала $P = (p_n)_{n=1}^{N_p}$ будем понимать вектор $\overline{C} = C_m(P) = (c_0, \dots, c_{m-1})$, где c_0, \dots, c_{m-1} - первые m коэффициентов разложения сигнала P по некоторой системе ортогональных функций $u_0(x), \dots, u_{m-1}(x), \dots$

Полиномы Чебышева определяются при помощи рекуррентного соотношения:

$$\begin{aligned}u_0(x) &= 1 \\u_1(x) &= x \\&\dots \\u_{n+1}(x) &= 2xu_n(x) - u_{n-1}(x)\end{aligned}$$

Вычисление коэффициентов разложения выполняется по рекуррентным соотношениям.

1.2.3 Спектральное сравнение профилей

На этой стадии спектральное представление профилей используется для производства сравнения на основе некоторого специально разработанного критерия.

Теперь можно ввести понятие расстояния редактирования для рассматриваемого метода поиска повторов.

Пусть X_1, X_2 - две биологические последовательности, такие что $|X_1| = |X_2| = K \geq w$. Будем понимать под GC- и GA-расстоянием редактирования для X_1 и X_2 :

$$\begin{aligned}\rho^{GC}(X_1, X_2) &= \left\| \overline{C}_1^{GC} - \overline{C}_2^{GC} \right\|, \\ \rho^{GA}(X_1, X_2) &= \left\| \overline{C}_1^{GA} - \overline{C}_2^{GA} \right\|, \\ \|\overline{C}\| &= \sum_{i=0}^{m-1} c_i^2\end{aligned}$$

Под повтором будем понимать пару последовательностей X_1, X_2 , удовлетворяющих системе:

$$\begin{cases} \rho^{GC}(X_1, X_2) < \varepsilon \\ \rho^{GA}(X_1, X_2) < \varepsilon \end{cases}$$

Под матрицей гомологии с окном профилирования w , окном аппроксимации a и шагом аппроксимации s для последовательностей X_1, X_2 будем понимать матрицу

$$M(X, Y, w, s, a) = (m_{ij})^{L_x \times L_y},$$

$$L_x = \lceil \frac{N_x - a - w + 1}{s} \rceil, L_y = \lceil \frac{N_y - a - w + 1}{s} \rceil$$

$$m_{ij} = \begin{cases} 1, & x_n \in \{G, C\} \\ 0, & \text{иначе} \end{cases}$$

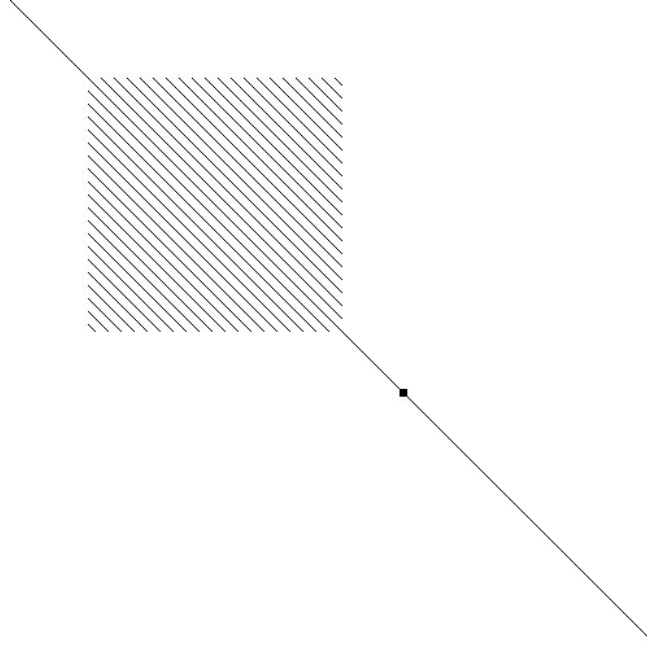


Рисунок 1.1: Пример гомологической матрицы

На рисунке 1.1 приведена гомологическая матрица, образованная в результате сравнения искусственной последовательности с самой собой. Матрица имеет симметричный вид.

1.2.4 Анализ гомологической матрицы

Рассмотрим все диагонали гомологической матрицы параллельные главной диагонали, включая ее саму. Фиксируем диагональ D_l . Назовем диагональным элементом $R(x, y, k)$ такое множество элементов m_{ij} гомологической матрицы $M^{L_x \times L_y}$, что:

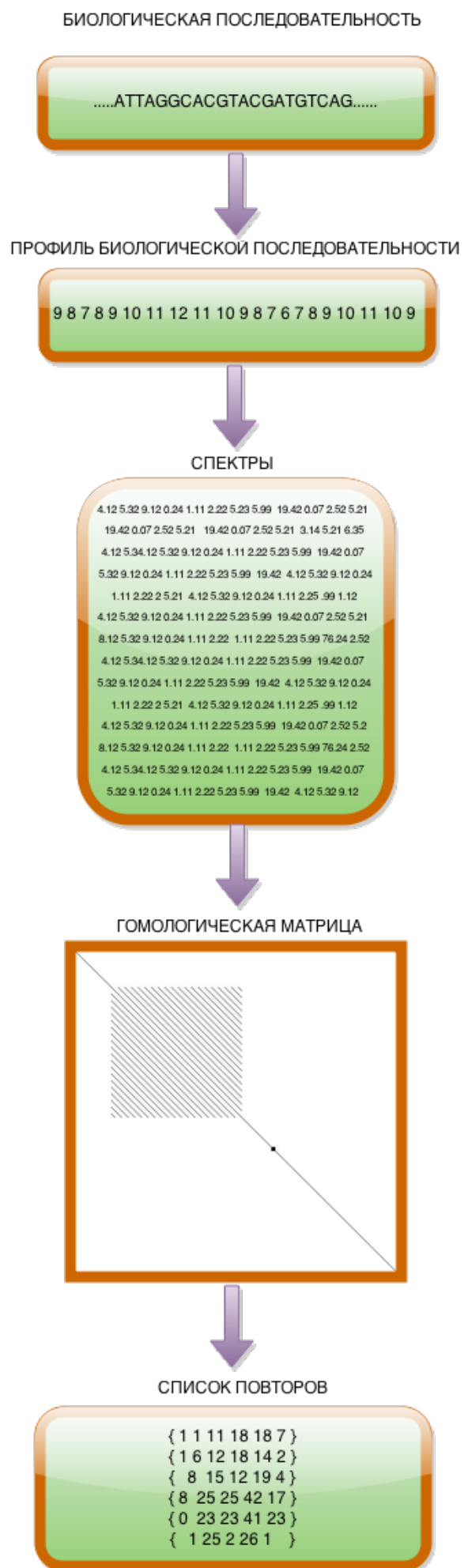
$$m_{ij} \in D_l$$

$$m_{ij} = 1, \quad i = \overline{y, y+k-1}, \quad j = \overline{x, x+k-1}, k \in \mathbb{N}$$

$$m_{x-1, y-1} = 0, \text{ если } x > 0, y > 0$$

$$m_{x+k, y+k} = 0, \text{ если } x+k < L_x, y+k < L_y$$

На последнем этапе анализа гомологической матрицы нужно найти все диагональные элементы этой матрицы. Зная параметры алгоритма по диагональным элементам можно восстановить искомые повторы.



1.3 Параллельный метод поиска повторов

Основная идея параллельного алгоритма поиска повторов заключается в равномерном распределении входных данных по процессам и их независимой обработке, запрашивая недостающие элементы у соседних процессов.

При разработке параллельного алгоритма во главу угла была поставлена автономность каждого этапа вычислений. Такой подход позволяет достигнуть следующих результатов:

- Логическое следование алгоритму
- Прозрачность системы
- Возможность отладки каждого этапа

1.3.1 Профилирование и спектральное индексирование

На рисунке 1.3 представлена схема выполнения первых двух этапов: получения профиля биологической последовательности и спектрального индексирования.

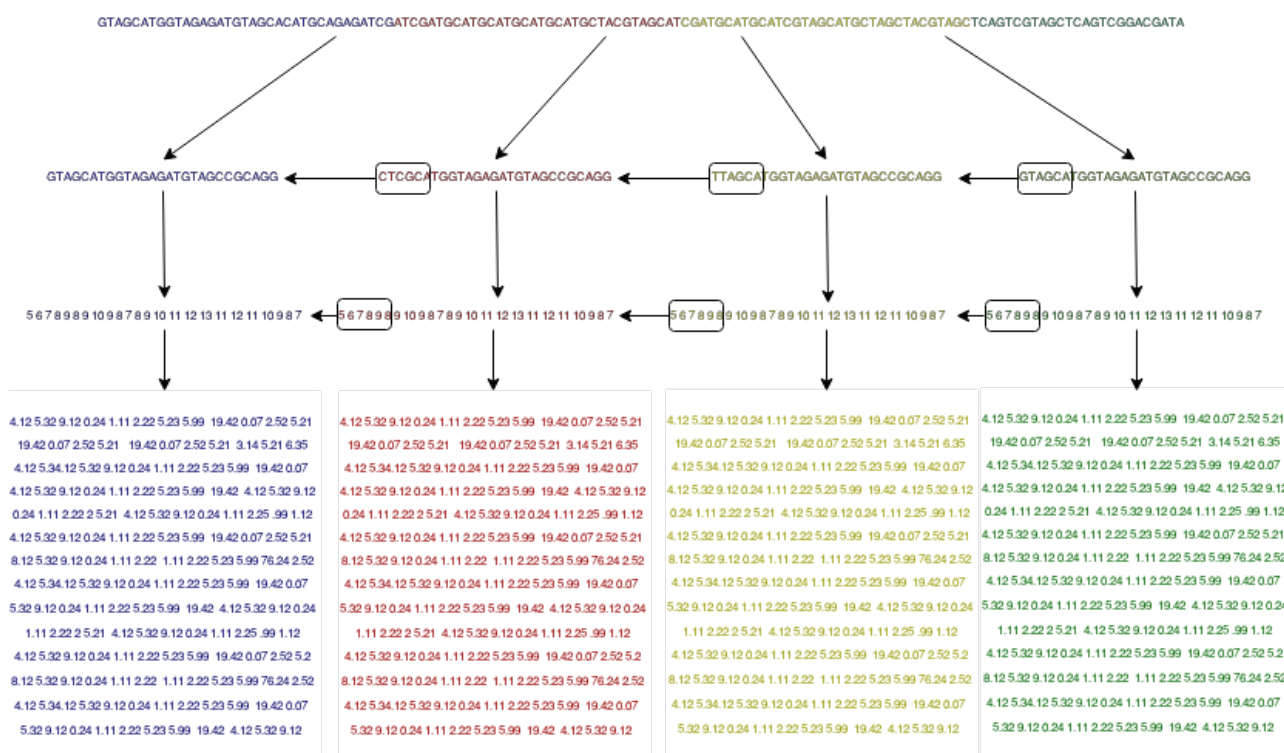


Рисунок 1.3: Параллельная схема работы этапа профилирования и спектрального индексирования

Параллельная схема двух этапов очень похожа, поэтому есть смысл рассматривать их вместе. Входная последовательность равномерно разбивается между всеми процессами и на каждом этапе процесс получает недостающие элементы у соседнего процесса справа и выполняет обработку данных. Данная схема основывается на принципе независимого выполнения этапов.

Естественно, что такой алгоритм не единственно возможный. Например, можно было бы использовать такую схему, когда профилирование и спектральное индексирование объединялись бы в один этап и по параметрам алгоритма каждый процесс рассчитывал сколько ему нужно взять элементов таким образом, чтобы обойтись без пересылок между процессами.

Но такой подход имеет и ряд недостатков: дублирование вычислений, сложность отладки, невозможность повторного использования данных для других параметров алгоритма.

К недостаткам предложенного алгоритма можно отнести обмены между процессами. Однако каждый процесс может начинать работу не дожидаясь завершения приема данных, т.к. они понадобятся только в конце работы. Таким образом можно скрыть передачу данных на фоне выполнения полезной работы и добиться ситуации, когда процессор не будет простаивать в ожидании данных.

Ниже рассматриваются два способа организации этапа спектрального сравнения и анализа гомологической матрицы.

1.3.2 Спектральное сравнение и анализ матрицы гомологии. Матричный метод

Во время спектрального сравнения каждый процесс будет отвечать за сравнение своей части спектров первой последовательности со всеми спектрами второй последовательности. Таким образом процесс будет хранить часть строк гомологической матрицы, соответствующих спектрам из первого набора.

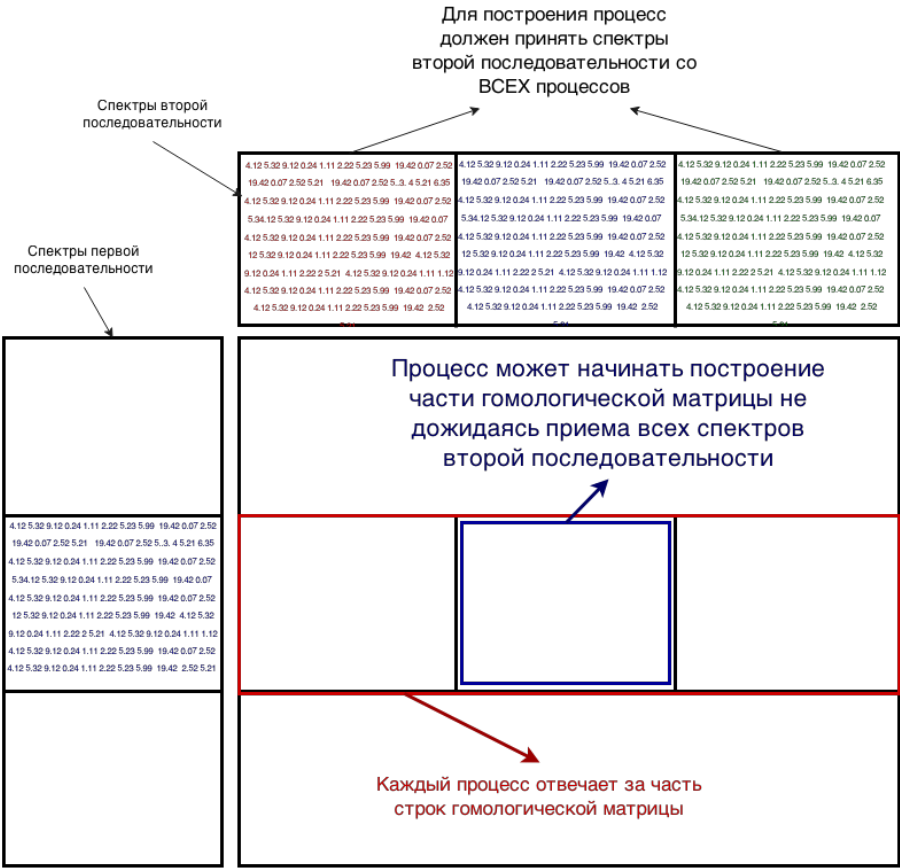


Рисунок 1.4: Сравнение спектров. Матричный метод

Для выполнения данного этапа процессу требуются собрать спектры второй последовательности со всех процессов. Однако и здесь можно добиться отсутствия простоя вычислительных платформ, если не дожидаться получения всех данных, а начинать работу по возможности. То есть как только придут данные хотя бы с одного процесса - сразу строить часть матрицы. Таким образом можно скрыть большую часть пересылок данных за полезными вычислениями и добиться хорошей масштабируемости алгоритма.

На последнем этапе каждый процесс анализирует имеющуюся часть гомологической матрицы на наличие диагональных элементов. Однако следует учитывать, что найденный диагональный элемент может иметь свое начало или окончание на другом процессе и таким образом быть разбитым по процессам, поэтому нужно “склеить” диагональные элементы с разных процессов.

Здесь будем действовать по следующему принципу: процесс “склеивает” диагональный элемент (т.е. ищет его целиком), только если диагональный элемент принадлежит нашему процессу. Диагональный элемент $R(x, y, k)$ принадлежит процессу i , если начало этого повторения находится на процессе i , т.е.:

$$R(x, y, k) \in i, \text{ если } m_{xy} \in M_i^{L_x \times L_y},$$

где $M_i^{L_x \times L_y}$ — гомологическая матрица, вычисленная на процессе i

Таким образом алгоритм анализа гомологической матрицы будет работать следующим образом:

1. Найти все диагональные элементы
2. Удалить диагональные элементы, не принадлежащие нашему процессу
3. Найти конец всех диагональных элементов

Такой алгоритм нахождения диагональных элементов, при котором последовательно идут этапы спектрального сравнения с полным построением матрицы гомологии и ее последующим анализом будем называть матричным методом.

1.3.3 Спектральное сравнение и анализ матрицы гомологии. Блочный метод

Явным недостатком матричного метода является объем памяти, необходимый для гомологической матрицы. Размер этой матрицы $\sim N/s$, где N - длина входной последовательности, s - сдвиг окна индексирования. Тогда, например, объем гомологической матрицы, необходимый для анализа человеческого генома (3.1 млрд. н.п.) со сдвигом s равным 100, будет составлять $(3.1 * 10^7 * 3.1 * 10^7 \sim 9 * 10^{14})$ 900 ТБ. Такой размер гомологической матрицы существенно сокращает количество возможных вычислительных платформ на которых можно было бы запускать программу.

Чтобы уйти от этого недостатка нам придется пренебречь принципом автономности этапов. Основная идея блочного метода заключается в чередовании этапов спектрального индексирования и анализа гомологической матрицы для обработки небольшого блока.

Итак, пусть пользователь задает новый параметр алгоритма - лимит оперативной памяти для гомологической матрицы. Также как и в матричном методе процесс будет отвечать за сравнение спектров первой последовательности со всеми спектрами второй последовательности. Единственное отличие блочного метода от матричного в том, что результат сравнения не будет сохраняться в матрице гомологии, а будет сразу же подвергаться анализу на наличие диагональных элементов. Таким образом гомологическая матрица становится чисто виртуальной структурой - на деле ничего в памяти не хранится.

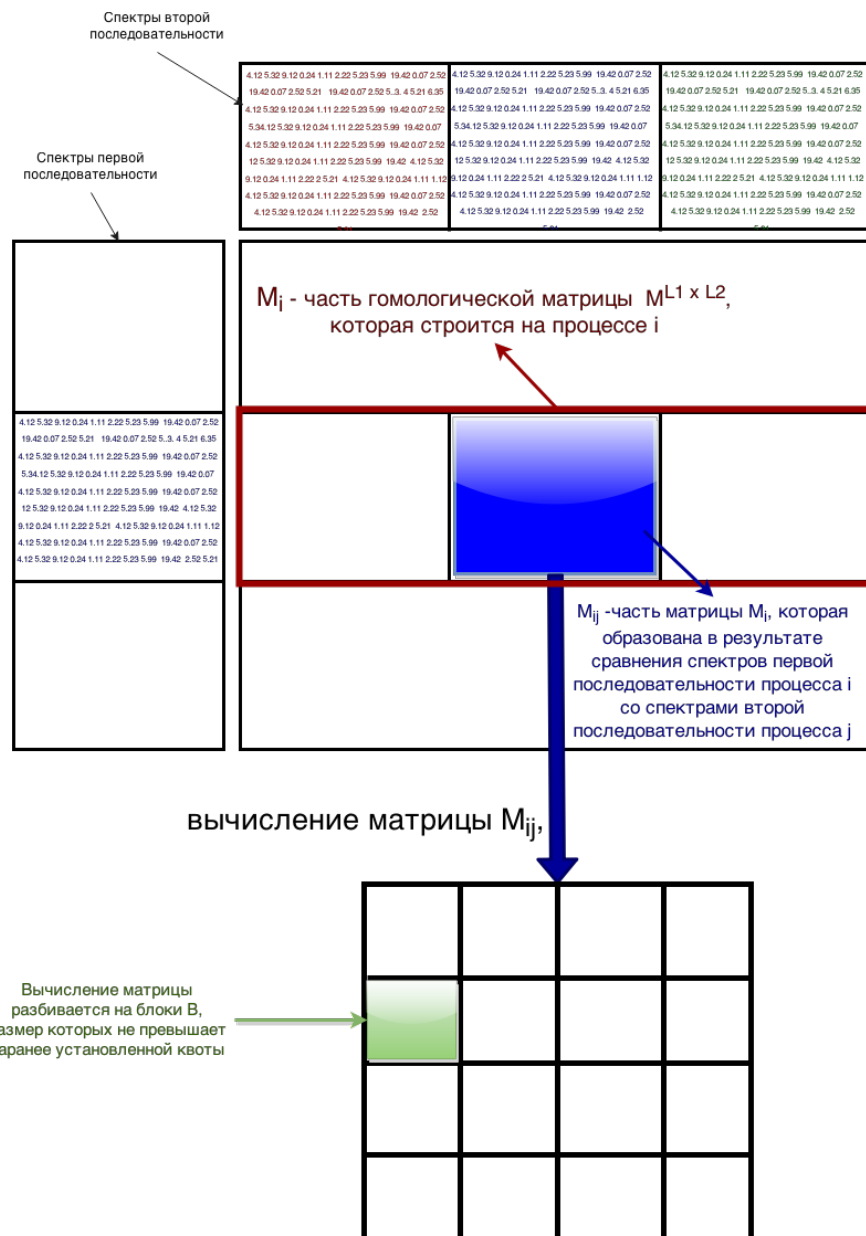


Рисунок 1.5: Сравнение спектров. Блочный метод

На рис. 1.5 разобрана схема блочного метода. Как только процесс принял часть спектров второй последовательности начинается их сравнение со спектрами первой последовательности. Выделяется блок памяти, объем которого не превышает заранее определенной квоты пользователя. Строится часть виртуальной гомологической матрицы равная размеру выделенного блока. Блок сразу же подвергается анализу и результат анализа сохраняется в списке диагональных элементов.

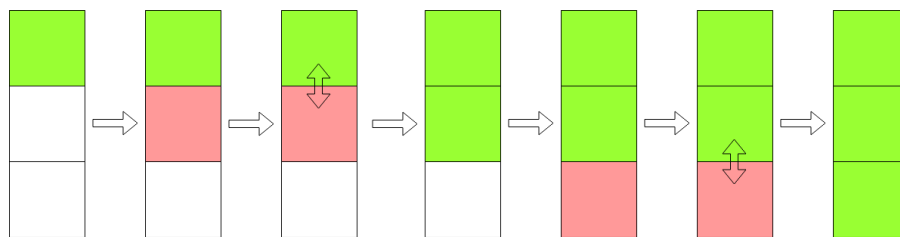


Рисунок 1.6: Склейка по вертикали

Блок сдвигается ниже на его высоту, идет построение и анализ. Результат анализа “склеивается” по вертикали с предыдущими результатами (рис. 1.6). Операция “склейки” диагональных элементов по вертикали не имеет ничего общего со “склежкой” между процессами, т.к. здесь процесс знает всю информацию о склеиваемых структурах.



Рисунок 1.7: Склейка по горизонтали

После построения столбца виртуальной гомологической матрицы результат сохраняется и происходит построение следующего столбца. После его постройки происходит “склейка” диагональных элементов по горизонтали. И так далее, пока не получим результат анализа с одним из процессов (см. рис 1.7).

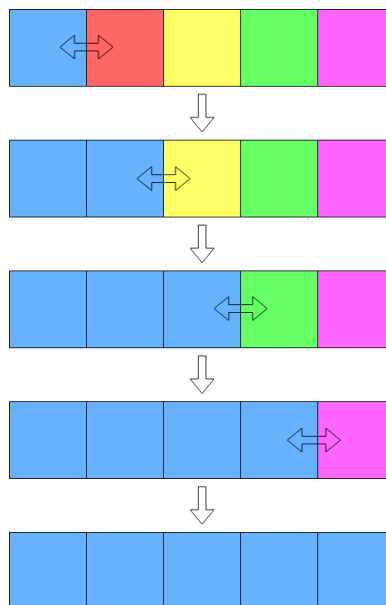


Рисунок 1.8: Склейка по горизонтали процессов

На последнем этапе будем иметь результаты анализа сравнения со всеми процессами. Остается применить ко всем операцию “склейка” горизонтали, и получим список диагональных элементов (см. рис 1.8).

То есть будем иметь такой же список, как и при матричном способе. Остается применить операцию “склейка” процессам и перевести диагональные элементы в искомые повторы.

Глава 2

Структура программной реализации

Программная реализация выполнена на C++ с использованием технологии параллельного программирования MPI и технологии CUDA-C для использования графических процессоров Nvidia. Общий объем кода (программа + GUI) составляет 6 тысяч строк.

Параллельный алгоритм разрабатывался таким способом, чтобы выполнялись следующие условия:

- Автономность каждого этапа вычислений
- Возможность загрузки/сохранения данных на каждом этапе
- Унифицированный доступ к ресурсам системы

В результате была разработана следующая структура классов:

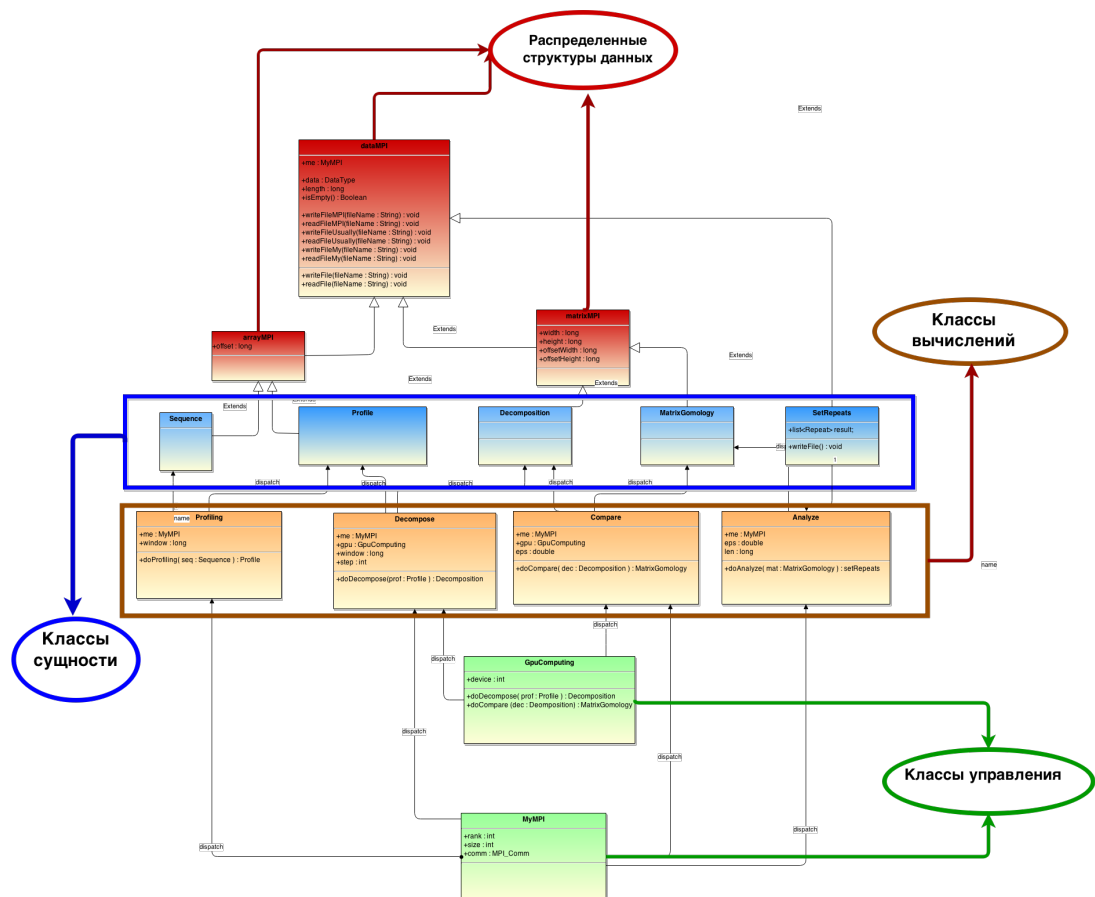


Рисунок 2.1: Диаграмма классов

Структура классов разбивается на следующие части:

1. Шаблонные классы, описывающие распределенные структуры данных.
2. Классы-сущности, специализирующие шаблонные классы.
3. Классы-вычислители, преобразующие одни классы-сущности в другие классы-сущности.
4. Управляющие классы, через которые осуществляется взаимодействие с MPI и GPU устройством.

Общая схема работы программы выглядит следующим образом:

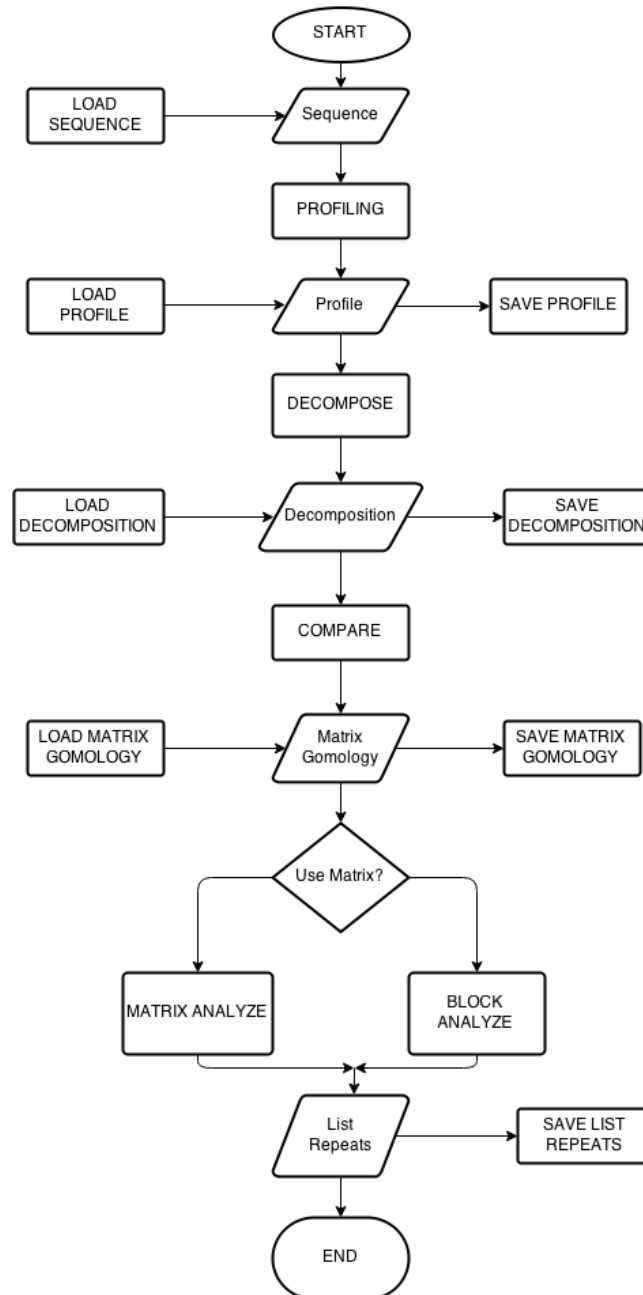


Рисунок 2.2: Блок-схема программы

Данная схема работы позволяет начать и завершить работу на любом этапе, ограничившись выполнением только нескольких этапов. Такая гибкость программы может быть очень

полезна если пользователь захочет посмотреть на результат работы программы при разных параметрах. Например, чтобы посмотреть на поведение программы при разном значении ε , то достаточно один раз сохранить спектры и в дальнейшем запускать программу сразу с этапа построения гомологической матрицы.

В целом программа выполнена в виде набора библиотечных функций, поэтому если пользователю потребуется специфичная версия программы, то относительно просто и в кратчайшие сроки такая версия может быть реализована.

Для работы с вводом/выводом файлов использовался архитектурно-независимый интерфейс стандарта MPI-2. Архитектура приложения позволяет использовать множество форматов ввода/вывода. В частности, для загрузки/сохранения гомологической матрицы предусмотрено два формата: бинарный файл и bmp изображение.

Для повышения масштабируемости алгоритма использовались только асинхронные операции передачи данных. Это позволило значительно снизить коммуникационные издержки, особенно на этапе построения гомологической матрицы где использовалась схема взаимодействия "all-to-all".

Для реализации этапа "склейки" по процессам использовались односторонние коммуникации. Данная технология MPI позволяет задавать все параметры, относящиеся к пересылке данных, только на одной стороне, что значительно упростило его реализацию этапа и повысило его эффективность.

Выделение в отдельные классы взаимодействия с MPI и GPU устройством позволило при компиляции отключить возможности этих классов и иметь следующие варианты сборки:

- последовательная программа
- последовательная программа, использующая графические процессоры
- параллельная программа без использования графических процессоров
- параллельная программа с использованием графических процессоров

Данная возможность позволяет существенно расширить список вычислительных платформ на которых может работать программа.

Тяжелые в вычислительном плане этапы спектрального индексирования и спектрального сравнения были перенесены на графические процессоры. Для эффективной работы графических ускорителей использовались механизм общей памяти и "набивка" данных. С помощью профилировщика было достигнуто такое разбиение задачи на блоки, что было достигнуто оптимальное соотношение между occupancy (загруженность устройства) и размером используемой общей памяти. Все эти меры позволили значительно ускорить этап построения гомологической матрицы.

Для удобства работы с программой был разработан ряд графических интерфейсов. Стоит отметить, что так как работа программы не предполагает диалога с пользователем, то данные интерфейсы работают в пакетном режиме, т.е. задают параметры алгоритма, запускают выполнение основной программы и ждут результатов. Поэтому все графические интерфейсы базируются на мощном Command Line Interface, реализованном в основной программе. CLI следует стандарту Posix, поддерживает разбор коротких и длинных опций. Таким образом графические интерфейсы правильно формируют строку аргументов и запускают основную программу.

Реализованы следующие виды интерфейсов:

- графический интерфейс на основе кроссплатформенной библиотеки Qt. Данный вариант очень если программа будет работать на локальной машине. (рисунок)

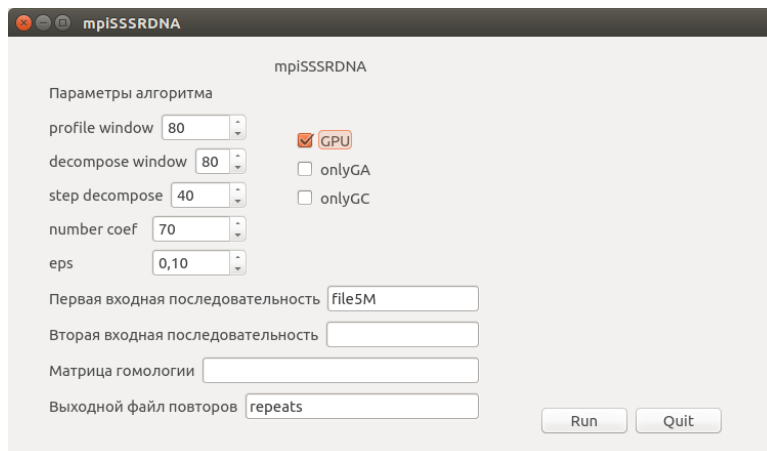


Рисунок 2.3: Графический интерфейс. Локальная машина

- графический интерфейс на основе кроссплатформенной библиотеки Qt и кроссплатформенной библиотеки libssh2. Данный вариант позволяет запускать программу графического интерфейса на локальном компьютере пользователя и запускать основную программу на удаленном сервере. (рисунок)

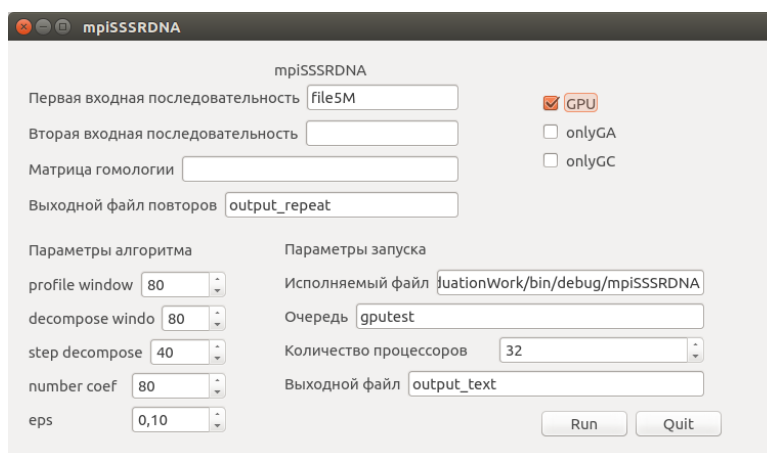


Рисунок 2.4: Графический интерфейс. Удаленная машина

Глава 3

Результаты вычислительных экспериментов

Все последующие эксперименты, за редким исключением, выполнялись на суперкомпьютере Ломоносов на базе Intel® Xeon 5570 и графических процессоров Tesla X2070 (поколение Fermi). Если запуск производился с использованием графических процессоров, то на каждое устройство приходилось по одному mpi-процессу, если не оговорено обратного.

В дальнейшем программа с использованием графических процессоров будет называться GPU-программой, а без использования - HOST-программой. Программа с построением матрицы гомологии и ее последующим анализом - матричной программой, а программу, использующую блочный метод (одновременное построение блока матрицы и его анализ), будем называть блочной программой.

В качестве входных данных использовалась искусственно сгенерированная тестовая биологическая последовательность длиной 5 млн. символов, содержащая в себе 2 протяженных повтора. Данная биологическая последовательность сравнивалась сама с собой.

В качестве параметров алгоритма использовались следующие значения:

- Длина окна профилирования: 250
- Длина окна спектрального индексирования: 250
- Шаг окна спектрального индексирования: 100
- Количество коэффициентов разложения: 75
- Точность поиска ε : 0.01

3.1 Сравнение блочной программы с матричной

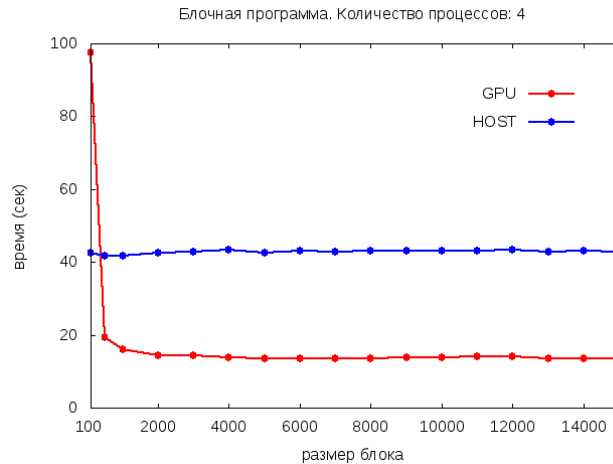


Рисунок 3.1: Зависимость времени работы блочной программы от размера блока

Как видно из графика 3.1 для блочной HOST-программы не имеет значения размер вычисляемого блока. Однако для GPU-программы размер блока имеет принципиальное значение - именно размер блока определяет загруженность устройства работой. Для эффективной работы GPU-программы нужен достаточно большой размер блока - такой, чтобы устройство заработало в полную силу. В тоже время не стоит увлекаться с увеличением размера блока, т.к. на устройстве может попросту не хватить памяти.

Во всех последующих экспериментах мы будем использовать блоки размером 10000x10000.

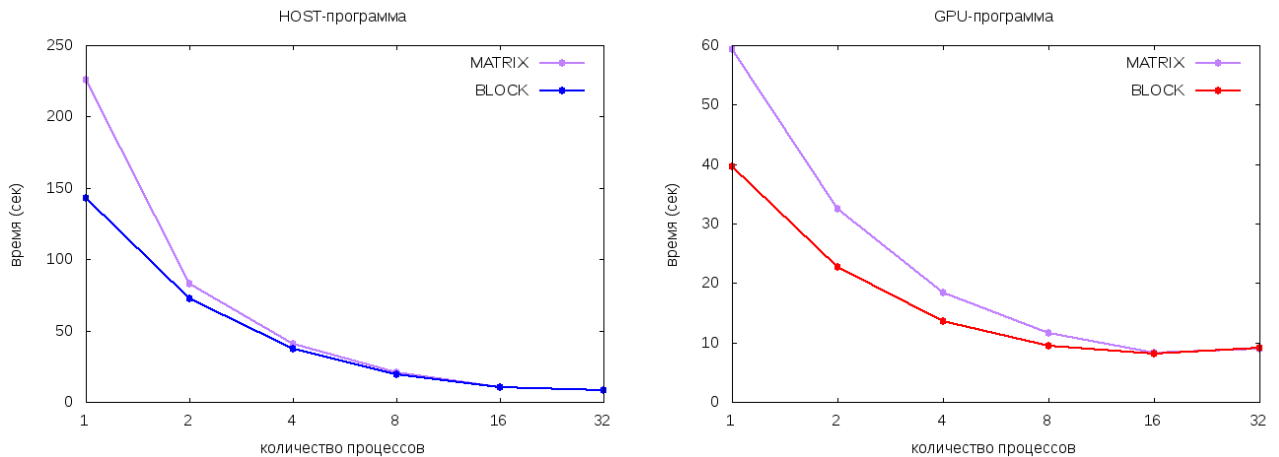


Рисунок 3.2: Сравнение времени работы блочного и матричного метода

Изначально предполагалось, что блочная программа будет дополнением к матричной, чтобы можно было использовать программу на системах с небольшим объемом оперативной памяти. Однако по результатам работы двух программ стало ясно, что и для GPU-программы, и для HOST-программы во всех случаях предпочтительнее использовать блочный вариант.

Стоит заранее оговорить, что использование блочного метода в GPU-программе дает лучшие результаты по сравнению с матричным методом, если использовать конфигурацию при которой на одно GPU устройство приходится больше одного `mpi`-процесса (см. рис. 3.5 б).

Также теоретически блочная программа будет лучше масштабироваться на системах со слабой коммуникационной средой, т.к. имеет больше времени на асинхронную передачу спектров (такое увеличение времени стало возможным за счет объединения этапа построения гомологической матрицы и её анализа).

Во всех последующих экспериментах мы будем использовать блочную программу.

3.2 Использование графических процессоров

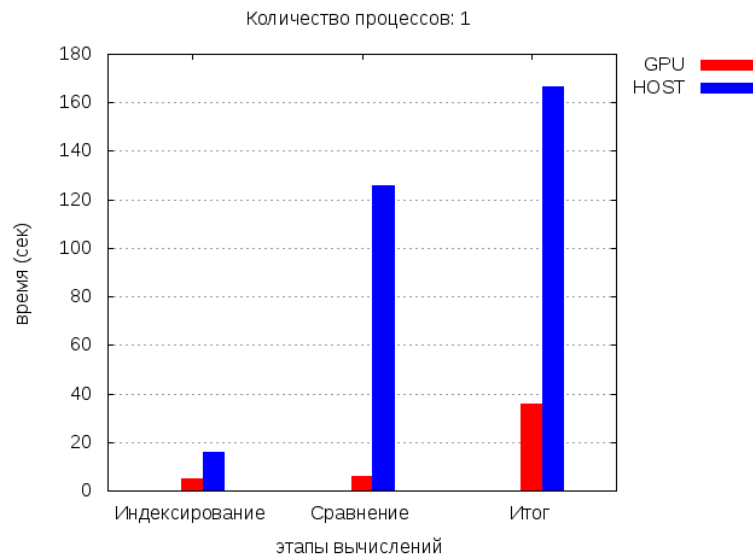


Рисунок 3.3: Сравнение HOST-программы и GPU-программы

На диаграмме 3.3 мы видим результат сравнения GPU-программы и HOST-программы, запущенных на одном процессе. Этап спектрального индексирования ускорился в 3 раза, этап спектрального сравнения в 22 раза, программа в целом ускорилась в 5 раз. Такие результаты ускорений значительно снизили долю этапов спектрального индексирования и спектрального сравнения в программе.

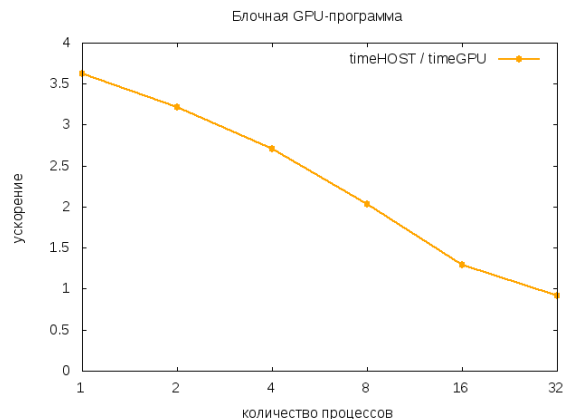


Рисунок 3.4: Масштабируемость ускорения GPU-программы

На графике 3.4 мы видим изменение ускорения GPU-программы по сравнению с HOST-программой в зависимости от числа процессов. Как и ожидалось - с ростом числа процессов ускорение падает. Это объясняется уменьшением в вычислениях доли этапа спектрального сравнения, а в последующем и недостаточной загруженностью графических процессоров.

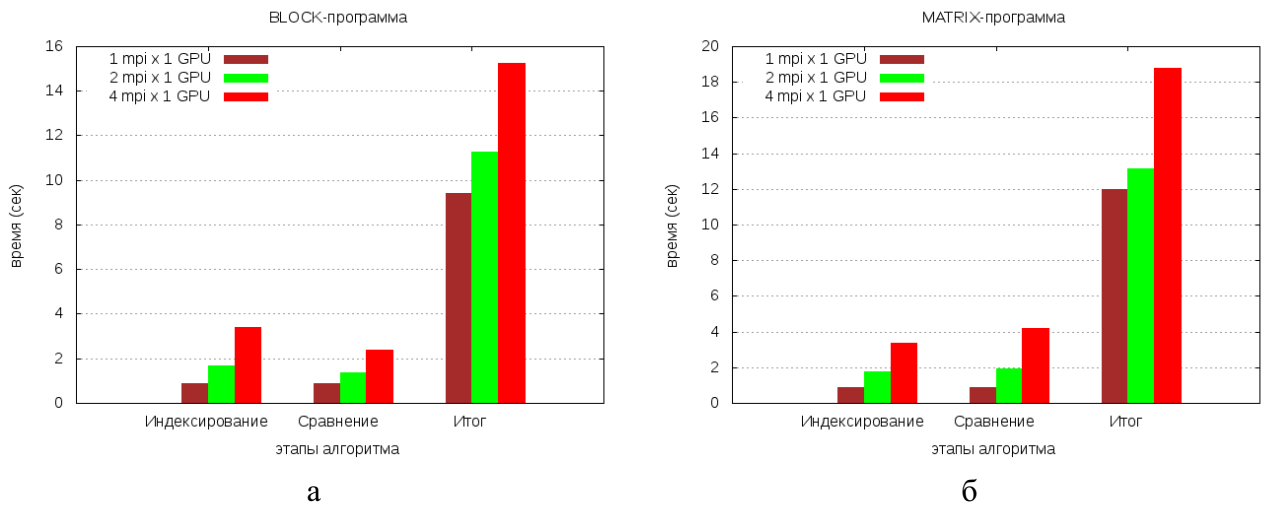


Рисунок 3.5: Изменение времени выполнения этапов программы от количества процессов на устройство GPU

Рассмотрим поведение GPU-программы, если на каждое GPU устройство будет приходиться больше одного mpi-процесса. Так как каждый процесс занимает устройство на короткое время, а не блокирует его на время выполнения всей программы, то эффективность использования GPU падает совсем незначительно. Это даст хорошие результаты на практике, т.к. большинство кластеров имеют в своем распоряжении большое число обычных процессоров и незначительное количество графических плат. Даже на таких системах можно будет запускать программу в конфигурации сильного разделения устройства (8 mpi x 1 GPU, 16 mpi x 1 GPU) и все равно время выполнения GPU-программы будет меньше времени выполнения HOST-программы. Результаты матричной программы хоть и не намного, но оказались хуже результатов блочной программы.

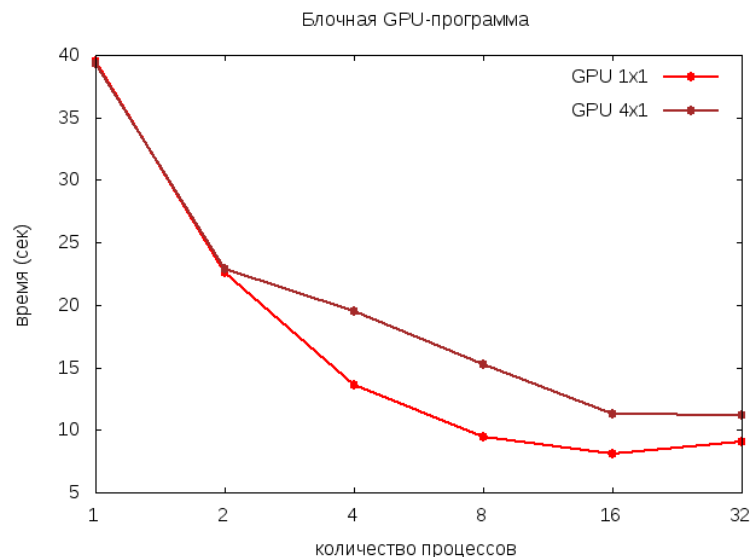


Рисунок 3.6: Масштабируемость алгоритма при использовании нескольких процессов на одно устройство

Разделение устройства между несколькими процессами не оказывает влияния на масштабируемость.

3.3 Масштабируемость параллельного алгоритма

Для следующих запусков использовалась искусственно сгенерированная биологическая последовательность длиной 20 млн.

Кроме Ломоносова вычисления производились на суперкомпьютере Blue Gene/P (4xPowerPC 450).

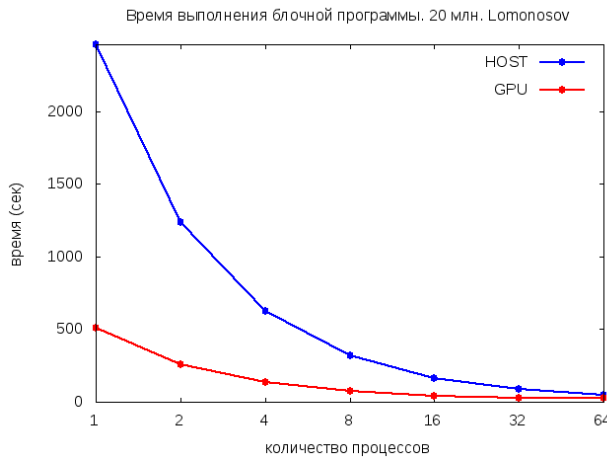


Рисунок 3.7: Сильная масштабируемость GPU- и HOST-программы

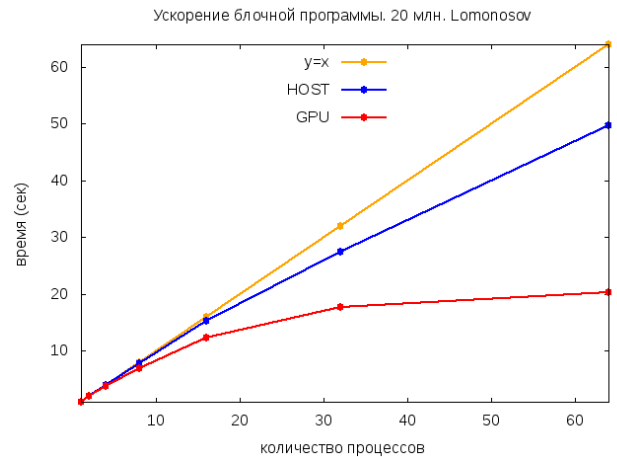


Рисунок 3.8: Ускорение GPU- и HOST-программы

GPU-программа показала чуть меньшую масштабируемость. Это объясняется уменьшением доли GPU-вычислений с уменьшением входных данных.

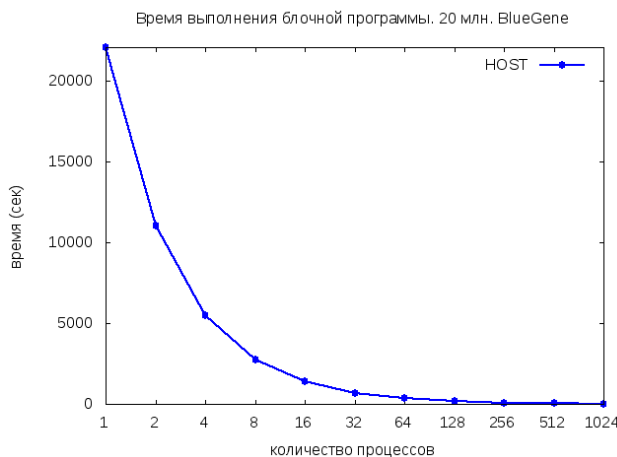


Рисунок 3.9: Сильная масштабируемость HOST-программы

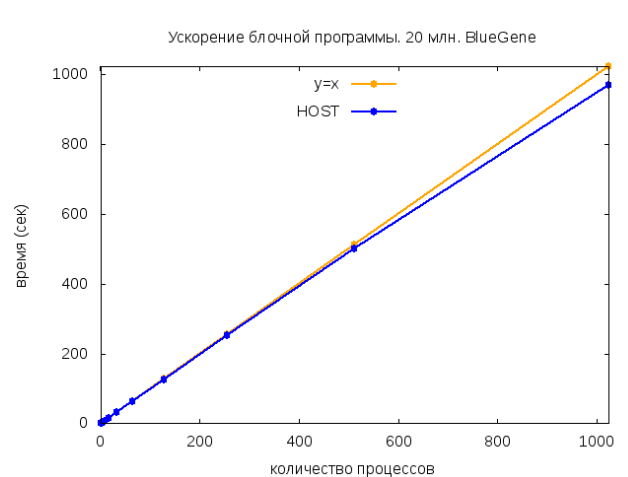


Рисунок 3.10: Ускорение HOST-программы

Программа показала очень хорошую масштабируемость на системе BlueGene/P. Во многом это произошло благодаря архитектуре системы, а так же отсутствию в программе блокирующих операций.

На рисунках ниже представлены результаты работы системы для анализа поведения параллельных программ, разрабатываемой Матвеевым Владимиром в рамках спецсеминара “Суперкомпьютерная обработка данных с использованием нейросетей и эволюционных вычислений” под руководством Л.Н.Королева и Н.Н.Поповой. Система позволяет визуализировать трассу параллельной программы, на которой отображается появление коммуникационных событий исследуемой программы во времени. Также система позволяет просматривать различные коммуникационные матрицы, в данном случае, по суммарному размеру сообщений и суммарному времени передачи сообщений между процессами.

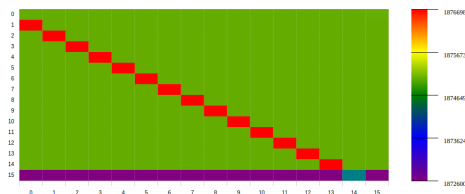


Рисунок 3.11:
Коммуникационная матрица взаимодействия
по размеру передаваемых данных

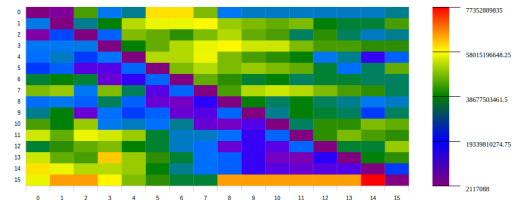


Рисунок 3.12:
Коммуникационная матрица взаимодействия
по времени

Из рисунка 3.11 видно, что все процессы обмениваются почти одинаковым количеством данных.

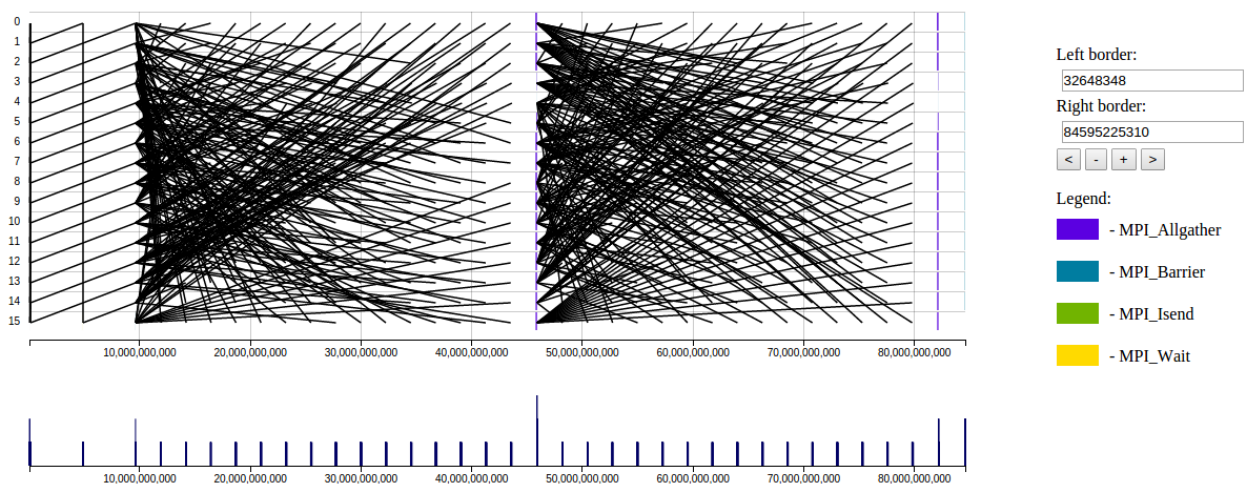


Рисунок 3.13: Трасса программы

Трасса из рисунка 3.13 очень хорошо отображает общую структуру программы:

- Обмен по топологии кольца (два этапа профилирования можно разглядеть в самом начале)
- Асинхронный обмен “каждый-с-каждым” который выполняется при анализе два раза для GC и GA профилей.

3.4 Верификация полученных данных

Ниже на рис. 3.14 и 3.15 изображены гомологические матрицы, полученные при помощи последовательной реализации спектрально-аналитического метода и параллельной программы. Эксперименты проводились на системе Ломоносов. Использовалась матричная HOST программа.

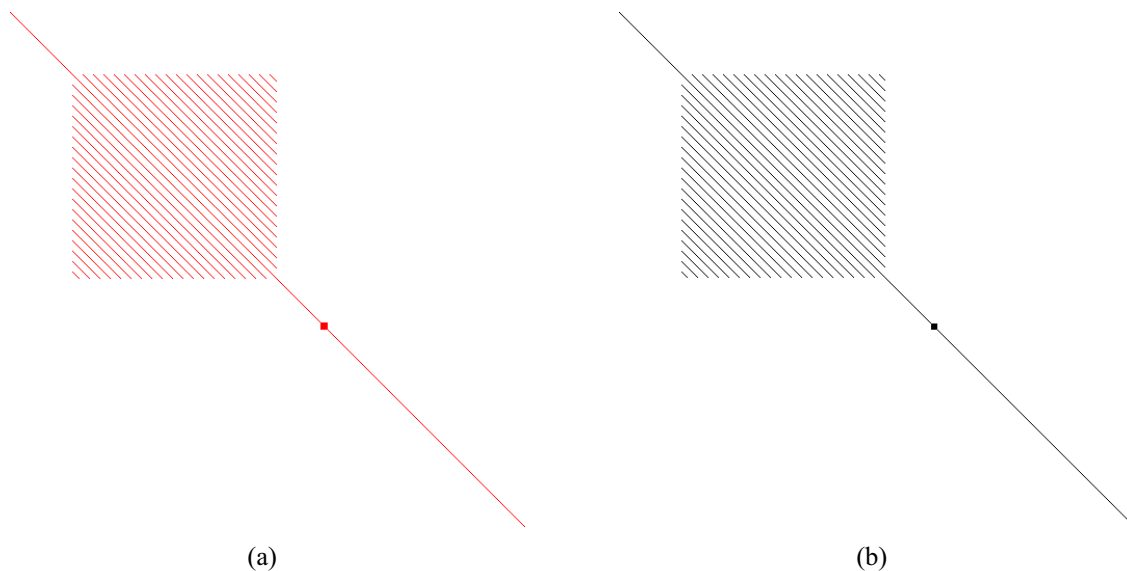


Рисунок 3.14: Гомологическая матрица сравнения искусственной последовательности с самой собой. Длина последовательности 50 тыс.; (a) последовательная программа; (b) параллельная программа;

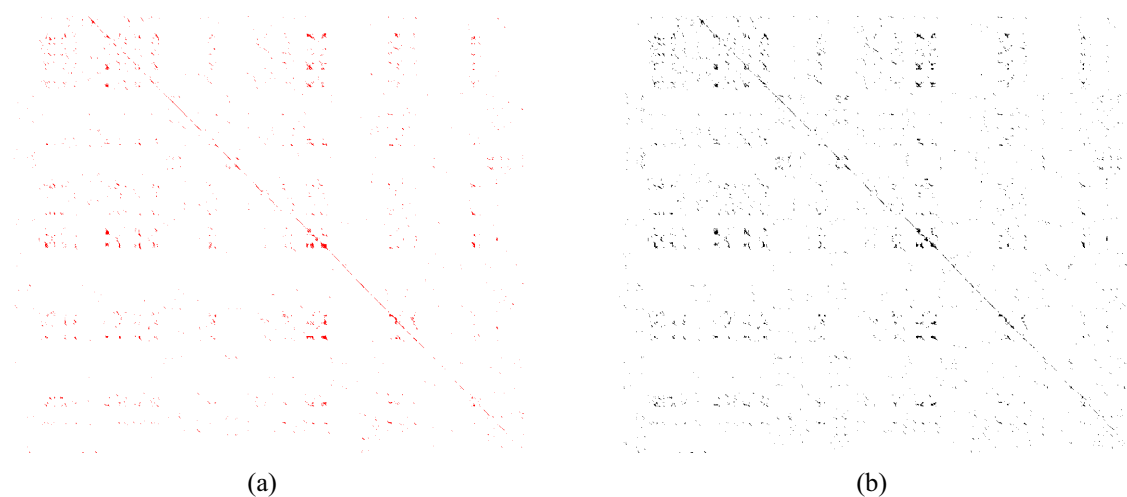


Рисунок 3.15: Гомологическая матрица сравнения *Mus musculus* и *Rattus norvegicus*. Длина последовательностей 172.5 Мб и 143.6 Мб соответственно. Параметры алгоритма: $W1 : 300000, d1 : 64, W2 : 800000, d2 : 160000, coef : 5, \varepsilon 4 * 10^{-5}$; (a) последовательная программа; (b) параллельная программа;

Заключение

В ходе выполнения дипломной работы были получены следующие основные результаты:

- Разработан и реализован на суперкомпьютерах Ломоносов и BlueGene/P параллельный алгоритм поиска повторов в биологических последовательностях. Алгоритм реализует парные сравнения последовательностей большого размера. Параллельная реализация алгоритма выполнена с использованием технологии MPI с возможностью подключения cuda-модулей для использования графических ускорителей.
- Проведено тестирование разработанного алгоритма на искусственных данных, а так же на решении реальной задачи сравнения биологических последовательностей крысы и мыши.
- Проведено исследование эффективности и масштабируемости разработанной параллельной программы.
- Показано, что использование графических процессоров позволяет ускорить алгоритм в 5 раз.
- Разработан графический интерфейс для параллельной программы, позволяющий пользователю запускать программу как на локальной, так и на удаленной системе. Интерфейс реализован с использованием кроссплатформенной библиотекой Qt, что позволяет использовать программу в различных операционных средах.

Часть работы с использованием графических процессоров была представлена на конкурсе студенческих работ CUDA Center of Excellence МГУ и была поддержана компанией Nvidia. Результаты работы опубликованы в CUDA АЛЬМАНАХе [9].

В качестве перспектив развития данной работы стоит рассмотреть возможность асинхронной работы между CPU и GPU в блочной программе на этапе построения гомологической матрицы и анализа результатов. Построение блока гомологической матрицы на GPU устройстве и асинхронный анализ этой матрицы на центральном процессоре могли бы снизить время выполнения этапа анализа (последнего весомого этапа программы, который не перенесен на GPU) и показать еще большее ускорение по сравнению с HOST-программой.

Список литературы

- [1] Антонов А. С. Технологии параллельного программирования MPI и OpenMP. — Издательство Московского университета, 2012.
- [2] Колесин М. С. — Исследование эффективности реализации спектрального метода поиска неточных повторов в последовательностях ДНК на графических процессорах. — Master's thesis, МГУ ВМК, 2011.
- [3] Поиск протяженных повторов в геномах на основе спектрально-аналитического метода / А. Н. Панкратов, М. И. Пятков, Р. К. Тетуев et al. // *Математическая биология и биоинформатика*. — 2012. — Vol. 7, no. 2. — Pp. 476–492.
- [4] Пятков М. И. Распараллеливание спектрального алгоритма поиска повторов в геномных последовательностях // Научный сервис в сети Интернет: суперкомпьютерные центры и задачи: Труды Международной суперкомпьютерной конференции (20-25 сентября 2010 г., г. Новороссийск). — Издательство Московского университета, 2010. — Pp. 663–666.
- [5] Пятков М. И. Спектрально-аналитический метод поиска протяженных повторяющихся нуклеотидных последовательностей в геномах: Ph.D. thesis / Институт математических проблем биологии Российской академии наук. — 2013.
- [6] Пятков М. И., Панкратов А. Н. Разномасштабное сравнение геномов с применением различных технологий параллельных вычислений. — конкурс «Эффективное использование GPU-ускорителей при решении больших задач».
- [7] Распараллеливание спектрального алгоритма поиска повторов в геномах / А. Н. Панкратов, М. И. Пятков, С. А. Комаров et al. // Научный сервис в сети Интернет: экзафлопсное будущее: Труды Международной суперкомпьютерной конференции (19-24 сентября 2011 г., г. Новороссийск). — Издательство Московского университета, 2011. — Pp. 304–306.
- [8] Сандерс Д., Кэндрот Э. Технология CUDA в примерах: введение в программирование графических процессоров / Под ред. А. В. Боресков. — ДМК Пресс, 2013.
- [9] Cuda альманах. — <http://www.nvidia.ru/docs/IO/141194/CUDA>— 2015. — февраль.
- [10] Pyatkov M. I., Pankratov A. N. Sbars: fast creation of dotplots for dna sequences on different scales using ga-,gc-content // *Bioinformatics*. — 2014. — Vol. 30, no. 12. — Pp. 1765–1766.
- [11] Statistic ncbi gene bank. — <http://www.ncbi.nlm.nih.gov/genbank/statistics>.