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

РАБОТА ПРОВЕРЕНА

Рецензент
Заведующий кафедрой ТУиО ЧелГУ,
д.ф.-м.н., профессор

_____ В.И. Ухоботов

«____»_____ 2022 г.

ДОПУСТИТЬ К ЗАЩИТЕ

Заведующий кафедрой, д.ф.-м.н.,
профессор

_____ Л.Б. Соколинский

«____»_____ 2022 г.

Разработка параллельного алгоритма поиска типичных подпоследовательностей временного ряда для графического процессора

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
ЮУрГУ – 02.04.02.2022.308-186.ВКР

Научный руководитель,
профессор кафедры СП, д.ф.-м.н.,
доцент
_____ М.Л. Цымблер

Автор работы,
студент группы КЭ-220
_____ А.И. Гоглачев

Ученый секретарь
(нормоконтролер)
_____ И.Д. Володченко
«____»_____ 2022 г.

Челябинск, 2022 г.

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

УТВЕРЖДАЮ
Зав. кафедрой СП

_____ Л.Б. Соколинский

07.02.2022 г.

ЗАДАНИЕ
на выполнение выпускной квалификационной работы магистра
студенту группы КЭ-220
Гоглачеву Андрею Игоревичу,
обучающемуся по направлению
02.04.02 «Фундаментальная информатика и информационные технологии»
(магистерская программа «Интеллектуальный анализ данных цифровой индустрии»)

1. **Тема работы** (утверждена приказом ректора от 25.04.2022 № 697-13/12)
Разработка параллельного алгоритма поиска типичных подпоследовательностей временного ряда для графического процессора.
2. **Срок сдачи студентом законченной работы:** 23.05.2022 г.
3. **Исходные данные к работе**

3.1. Imani S., Madrid F., Ding W., Crouter S., Keogh E. Matrix Profile XIII: Time Series Snippets: A New Primitive for Time Series Data Mining // Proceedings of the 9th IEEE Int. Conf. on Big Knowledge (ICBK). Singapore, November 17–18, 2018. – P. 382–389.

3.2. Gharghabi S. An ultra-fast time series distance measure to allow data mining in more complex real-world deployments. // Data Mining and Knowledge Discovery – 2020. – Vol. 34. – No. 4. – P. 1104–1135.

3.3. Yeh C.M. Matrix Profile I: All Pairs Similarity Joins for Time Series: A Unifying View That Includes Motifs, Discords and Shapelets. // IEEE 16th International Conference on Data Mining, ICDM 2016, Barcelona, Spain, December 12-15, 2016. – 2016. – P. 1317–1322.

3.4. Cheng J., Grossman M., McKercher T. Professional CUDA C Programming. 1st Edition. – United Kingdom: Wrox, 2014. – 528 p.

4. Перечень подлежащих разработке вопросов

4.1. Провести обзор алгоритмов поиска типичных подпоследовательность временного ряда.

4.2. Изучить аппаратную архитектуру и программную модель графического процессора NVIDIA.

4.3. Спроектировать и реализовать параллельный алгоритм поиска типичных подпоследовательностей для графического процессора.

4.4. Провести вычислительные эксперименты по анализу производительности и точности разработанного алгоритма.

5. Дата выдачи задания: 07.02.2022 г.

Научный руководитель,
профессор кафедры СП, д.ф.-м.н., доцент М.Л. Цымблер

Задание принял к исполнению А.И. Гоглачев

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Актуальность темы

Интеллектуальный анализ данных (Data Mining) представляет собой совокупность методов и алгоритмов для обнаружения в данных ранее неизвестных и практически полезных знаний, используемых для принятия стратегически важных решений в различных сферах человеческой деятельности [8].

Временные ряды являются одним из наиболее важных классов данных, подвергаемых интеллектуальному анализу. Под временным рядом (time series) понимают последовательность хронологически упорядоченных вещественных значений. Временные ряды встречаются в широком спектре научных и практических задач: моделирование климата и погоды [15], компьютерная обработка речи и музыки [17], финансовое прогнозирование [6, 14], мониторинг показателей функциональной диагностики организма человека в медицине [5], исследования ДНК в биологии [4], классификация звезд в астрономии [12, 19] и др.

Поиск типичных подпоследовательностей временного ряда является одной из актуальных задач интеллектуального анализа временных рядов. Данная задача предполагает нахождение набора подпоследовательностей временного ряда, которые адекватно отражают течение процесса или явления, задаваемого этим рядом. Поиск типичных подпоследовательностей дает возможность резюмировать и визуализировать большие временные ряды в широком спектре приложений: мониторинг технического состояния сложных машин и механизмов, интеллектуальное управление системами жизнеобеспечения, мониторинг показателей функциональной диагностики организма человека и др.

Предложенная недавно концепция сниппета формализует типичную подпоследовательность временного ряда следующим образом. Сниппет

представляет собой подпоследовательность, на которую похожи многие другие подпоследовательности данного ряда в смысле специализированной меры схожести, основанной на евклидовом расстоянии. Поиск типичных подпоследовательностей с помощью сниппетов показывает адекватные результаты для временных рядов из широкого спектра предметных областей, однако соответствующий алгоритм имеет высокую вычислительную сложность $O(n^2 \cdot (n-m)/m)$, где n – длина временного ряда, m – длина сегмента [9], – что дает низкую производительность при обработке временных рядов, насчитывающих от сотни тысяч элементов.

В настоящее время графический процессор (Graphics Processing Unit, GPU) компании NVIDIA является одной из наиболее популярных платформ высокопроизводительных вычислений [13]. Графические процессоры обеспечивают от сотен до тысяч процессорных ядер и значительно опережают традиционные процессоры по производительности. В соответствии с этим актуальной является задача разработки параллельного алгоритма поиска типичных подпоследовательностей временного ряда на современных графических процессорах на архитектуре NVIDIA CUDA.

Цель и задачи исследования

Целью данной выпускной квалификационной работы является разработка параллельного алгоритма поиска типичных подпоследовательностей временного ряда для современных графических процессоров архитектуры NVIDIA CUDA.

Для достижения поставленной цели необходимо решить следующие задачи:

1) провести обзор последовательных алгоритмов поиска типичных подпоследовательностей временного ряда;

2) изучить аппаратную архитектуру и программную модель графического процессора NVIDIA;

3) выполнить проектирование и реализацию параллельного алго-

ритма поиска типичных подпоследовательностей для графического процессора;

4) провести вычислительные эксперименты, исследующие эффективность разработанного алгоритма.

Структура и объем работы

Работа состоит из введения, четырех глав, заключения и списка литературы. Объем работы составляет 40 страниц, объем списка литературы – 27 источников.

Содержание работы

Первая глава, «Обзор работ по тематике исследования», содержит обзор и сравнительный анализ существующих последовательных алгоритмов поиска типичных подпоследовательностей временного ряда.

Вторая глава, «Формальные обозначения и постановка задачи», содержит формальные определения и обозначения, а также краткое описание последовательного алгоритма поиска типичных подпоследовательностей Snippet-Finder [16], используемого в качестве основы разработки параллельного алгоритма.

В третьей главе, «Параллельный алгоритм поиска лейтмотивов временного ряда», приведено описание аппаратной архитектуры и программная модель графического процессора NVIDIA и дано детальное описание разработанного параллельного алгоритма.

В четвертой главе, «Вычислительные эксперименты», приведены результаты вычислительных экспериментов по исследованию эффективности предложенного алгоритма.

Заключение резюмирует результаты, полученные в рамках исследования.

1. ОБЗОР РАБОТ ПО ТЕМАТИКЕ ИССЛЕДОВАНИЯ

Теме поиска типичных подпоследовательностей временного ряда посвящены следующие основные работы. В работе [16] Муин (Mueen) и др. описали концепцию лейтмотивов, которая может использоваться для формализации задачи поиска типичных подпоследовательностей временного ряда. Лейтмотив (motif) представляет собой пару подпоследовательностей временного ряда, которые наименее удалены друг от друга в смысле некоторой функции расстояния. Авторами предложен эффективный алгоритм поиска лейтмотивов на основе евклидова расстояния, который использует отбрасывание бесперспективных кандидатов в лейтмотивы с помощью аксиомы треугольника. Позднее в работах [25] и [26] были предложены параллельные версии указанного алгоритма для многоядерного ускорителя и графического процессора соответственно. Использование концепции лейтмотива позволяет найти схожие подпоследовательности ряда, которые можно считать повторяющимися и типичными. Однако лейтмотивы не вполне подходят для резюмирования, поскольку отсутствует возможность указать долю временного ряда, которой соответствует лейтмотив.

В предметных областях, не связанных с временными рядами, задачи резюмирования и визуализации множеств объектов могут быть решены с помощью методов кластеризации. Кластеризация (clustering) предполагает разбиение заданного множества объектов на подмножества (кластеры) таким образом, чтобы объекты из одного кластера были более похожи друг на друга, чем на объекты из других кластеров, в смысле выбранной меры схожести. Однако кластеризация множества подпоследовательностей временного ряда, имеющих одну длину (которые в этом случае рассматриваются как точки многомерного метрического пространства), несостоятельна и приводит к лишенным смысла результатам, как показал Кеог (Keogh) и др. в работе [11].

В работе [21] Йе (Ye) и др. предложили концепцию шейплетов, которая может использоваться для формализации понятия типичных подпоследовательностей временного ряда. Данная концепция предполагает, что подпоследовательности ряда предварительно подвергнуты классификации, и шейплет (shapelet) представляет собой подпоследовательность ряда, которая минимально удалена от большинства подпоследовательностей заданного класса и максимально удалена от подпоследовательностей, принадлежащих другим классам, в смысле заданной функции расстояния. Применение шейплетов в качестве типичных подпоследовательностей затруднено, поскольку данный подход требует обучения с учителем на основе экспертных знаний в целевой предметной области.

В работе [10] Индык (Indyk) и др. предложили концепции ослабленного периода и средней тенденции ряда, определяемые следующим образом. Пусть для заданного временного ряда фиксированы длина подпоследовательности и мера схожести. Подпоследовательность временного ряда является ослабленным периодом (relaxed period), если синтетический ряд, который имеет ту же длину, что исходный ряд, и составлен посредством многократной конкатенации указанной подпоследовательности, имеет наибольшую схожесть с исходным рядом. Средней тенденцией (average trend) временного ряда называется подпоследовательность, для которой достигается наибольшее значение суммы квадратов значений ее схожести со всеми прочими подпоследовательностями ряда. Однако использование данных концепций предполагает, что временной ряд предварительно разделен на периоды с четко определенной длительностью и начальным индексом, что существенно сужает спектр предметных областей для применения данного подхода.

В работе [9] Кеогом (Keogh) и др. для формализации задачи поиска типичных подпоследовательностей временного ряда предложена концепция сниппетов. Сниппет (snippet) представляет собой подпоследователь-

ность заданной длины, на которую похожи многие другие подпоследовательности данного ряда в смысле специально определяемой меры схожести. Набор сниппетов имеет существенно меньшую мощность, чем множество подпоследовательностей ряда, имеющих заданную длину, и потому используется для резюмирования исходного временного ряда. По сравнению с вышеупомянутыми подходами к решению задачи поиска типичных подпоследовательностей временного ряда сниппеты поддерживают для результатов поиска количественно измеряемые свойства достоверности и покрытия. Достоверность (fidelity) сниппета означает, что между данным сниппетом и каждой из подпоследовательностей, которые он резюмирует, обеспечивается схожесть, наибольшая из возможных. Покрытие (coverage) сниппета означает, что можно указать долю подпоследовательностей ряда, которые резюмирует данный сниппет. Эксперименты авторов показывают, что поиск типичных подпоследовательностей с помощью сниппетов показывает адекватные результаты для временных рядов из широкого спектра предметных областей [9]. Однако алгоритм SnippetFinder, предложенный авторами, имеет высокую сложность: $O(n^2 \cdot (n - m)/m)$, где n – длина временного ряда, m – длина сниппета [9], – что дает низкую производительность при обработке временных рядов, насчитывающих от сотни тысяч элементов.

Проведенный обзор показывает, что актуальной является задача распараллеливания алгоритма SnippetFinder [9] для современных высокопроизводительных платформ. В данном исследовании предлагается новый параллельный алгоритм поиска сниппетов временного ряда для графического процессора.

2. ФОРМАЛЬНЫЕ ОБОЗНАЧЕНИЯ И ПОСТАНОВКА ЗАДАЧИ

2.1. Основные определения и нотация

В данном разделе приводятся обозначения и определения используемых терминов в соответствии с работой [9].

Временной ряд (time series) T представляет собой последовательность хронологически упорядоченных вещественных значений:

$$T = (t_1, \ldots, t_n),\ t_i \in \mathbb{R}. \tag{1}$$

Число n обозначается как $|T|$ и называется длиной ряда.

Подпоследовательность (subsequence) $T_{i,m}$ временного ряда T представляет собой непрерывное подмножество T из m элементов, начиная с позиции i:

$$T_{i,m} = (t_i, \ldots, t_{i+m-1}),\ 1 \leqslant m \ll n, 1 \leqslant i \leqslant n - m + 1. \tag{2}$$

Временной ряд T может быть логически разбит на сегменты – непересекающиеся подпоследовательности заданной длины m. Здесь и далее без существенного ограничения общности мы можем считать, что n кратно m, поскольку $m \ll n$ Множество сегментов ряда, имеющих длину $m \ll n$, обозначим как S_T^m, элементы этого множества как $S_1, \ldots, S_{n/m}$:

$$S_T^m = (S_1, \ldots, S_{n/m}),\ S_i = T_{m\cdot(i-1)+1,\,m}. \tag{3}$$

Концепция сниппетов (snippet) предложена Кеогом и др. в работе [9] и уточняет понятие типичных подпоследовательностей временного ряда следующим образом. Каждый сниппет представляет собой один из сегментов временного ряда. Со сниппетом ассоциируются его ближайшие соседи – подпоследовательности ряда, имеющие ту же длину, что и сниппет, которые более похожи на данный сниппет, чем на другие сегменты. Для вычисления схожести подпоследовательностей используется специализированная мера схожести, основанная на евклидовом расстоянии. Сниппе-

ты упорядочиваются по убыванию мощности множества своих ближайших соседей. Формальное определение сниппетов выглядит следующим образом. Обозначим множество сниппетов ряда T, имеющих длину m, как C_T^m, а элементы этого множества – как $C_1, \ldots, C_K$:

$$C_T^m = (C_1, \ldots, C_K),\ C_i \in S_T^m. \tag{4}$$

Число K $(1 \leqslant K \leqslant n/m)$ представляет собой параметр, задаваемый прикладным программистом, и отражает соответствующее количество наиболее типичных сниппетов. С каждым сниппетом ассоциированы следующие атрибуты: индекс сниппета, ближайшие соседи и значимость данного сниппета.

Индекс сниппета $C_i \in C_T^m$ обозначается как $C_i.index$ и представляет собой номер j сегмента S_j, которому соответствует подпоследовательность ряда $T_{m\cdot(j-1)+1,\,m}$.

Множество ближайших соседей сниппета $C_i \in C_T^m$ обозначается как $C_i.Neighbors$ и содержит подпоследовательности ряда, которые более похожи на данный сниппет, чем на другие сегменты ряда, в смысле меры схожести MPdist [7]:

$$\begin{gathered} C_i.Neighbours = \{T_{j,m} \mid S_{C_i.index} = \arg\min_{1 \leqslant r \leqslant n/m} \text{MPdist}(T_{j,m},\, S_r), \\ 1 \leqslant j \leqslant n - m + 1\}. \end{gathered} \tag{5}$$

Значимость сниппета $C_i \in C_T^m$ обозначается как $C_i.frac$ представляет собой долю множества ближайших соседей сниппета в общем количестве подпоследовательностей ряда, имеющих длину m:

$$C_i.frac = \frac{|C_i.Neighbours|}{n - m + 1}. \tag{6}$$

Сниппеты упорядочиваются по убыванию их значимости:

$$\forall C_i, C_j \in C_T^m : i < j \iff C_i.frac \geq C_j.frac. \tag{7}$$

2.2. Мера схожести MPdist

Мера MPdist [7], используемая для вычисления схожести подпоследовательностей при нахождении сниппетов, неформально определяется следующим образом. Два временных ряда равной длины m тем более похожи друг на друга в смысле меры MPdist, чем больше в каждом из них имеется подпоследовательностей заданной длины ℓ $(3 \leqslant \ell \leqslant m)$, близких друг к другу в смысле нормализованного евклидова расстояния. Мера MPdist устойчива к выбросам, шумам и пропущенным значениям во временном ряде, а также инвариантна к амплитуде, сдвигу и фазе временного ряда [7]. Отметим, что MPdist как симметричная неотрицательная функция является мерой, но не метрикой, поскольку для нее выполнены аксиомы тождества и симметрии, но не выполняется аксиома треугольника [7]. Формальное определение меры MPdist выглядит следующим образом.

Пусть имеются два временных ряда A и B равной длины $(|A| = |B| = m)$ и задан параметр ℓ $(3 \leqslant \ell \leqslant m)$, который мы будем называть значимой длиной подпоследовательности. Типичным диапазоном для выбора значимой длины подпоследовательности служит отрезок $[\lceil 0, 3m \rceil; \lceil 0, 8m \rceil]$ [7]. В дальнейшем изложении, упоминая параметр m в контексте вычисления меры MPdist, мы подразумеваем, что также задан параметр ℓ.

Далее в определении используется понятие матричного профиля временного ряда, предложенное Кеогом и др. в работе [22]. Матричным профилем (matrix profile) временных рядов A и B (с учетом значимой длины подпоследовательности ℓ) будем называть временной ряд, обозначаемый как P_{AB} и имеющий длину $m-\ell+1$, элементами которого являются нормализованные евклидовы расстояния от каждой подпоследовательности ряда A, имеющей длину ℓ, до ближайшей к ней в смысле указанного расстояния

подпоследовательности ряда B:

$$P_{AB} = (d_1, \ldots, d_{m-\ell+1}), d_i = \text{ED}_{\text{norm}}(A_{i,\ell}, B_{j,\ell}), B_{j,\ell} = \\ = \arg \min_{1 \leqslant q \leqslant m-\ell+1} \text{ED}_{\text{norm}}(A_{i,\ell}, B_{q,\ell}). \tag{8}$$

Нормализованное евклидово расстояние между двумя подпоследовательностями представляет собой евклидово расстояние между этими подпоследовательностями, подвергнутыми z-нормализации:

$$\text{ED}_{\text{norm}}(X, Y) = \text{ED}(\hat{X}, \hat{Y}) = \sqrt{\sum_{i=1}^{\ell} (\hat{x}_i - \hat{y}_i)^2}. \tag{9}$$

Z-нормализованная подпоследовательность $\hat{X}$ вычисляется из X следующим образом:

$$\hat{X} = (x_1, \ldots, x_\ell) : \hat{x}_i = \frac{x_i - \mu_x}{\sigma_x}, \\ \mu_x = \frac{1}{\ell} \sum_{i=1}^{\ell} x_i, \; \sigma_x = \sqrt{\frac{1}{\ell} \sum_{i=1}^{\ell} x_i^2 - \mu_x^2}. \tag{10}$$

Аналогичным образом определяется матричный профиль рассматриваемых рядов, взятых в порядке B и A, и обозначается как P_{BA}:

$$P_{BA} = (d_1, \ldots, d_{m-\ell+1}), d_i = \text{ED}_{\text{norm}}(B_{i,\ell}, A_{j,\ell}), A_{j,\ell} = \\ = \arg \min_{1 \leqslant q \leqslant m-\ell+1} \text{ED}_{\text{norm}}(B_{i,\ell}, A_{q,\ell}). \tag{11}$$

Выполним конкатенацию матричных профилей P_{AB} и P_{BA} и обозначим полученный временной ряд длины $2(m - \ell + 1)$ как P_{ABBA} (здесь и далее для обозначения конкатенации будем использовать символ $\odot$):

$$P_{ABBA} = P_{AB} \odot P_{BA}. \tag{12}$$

Далее, обозначим за $SortedP_{ABBA}$ ряд P_{ABBA}, в котором элементы упорядочены по возрастанию. Для вычисления схожести между рядами A и B в смысле меры MPdist используется k-е значение ряда $SortedP_{ABBA}$, где k является параметром. Типичным значением указанного параметра яв-

ляется 5% от $2m$, суммарной длины рядов A и B [7]. Однако, если величина значимой длины подпоследовательности ℓ существенно близка к длине ряда m, то конкатенация матричных профилей P_{ABBA} имеет длину менее 5% от $2m$. В этом случае значением меры MPdist полагается максимальное значение элемента в конкатенации матричных профилей P_{ABBA}. Нижеследующая формула формализует приведенные рассуждения:

$$\text{MPdist}(A, B, \ell) = \begin{cases} SortedP_{ABBA}(k), & |P_{ABBA}| > k \\ SortedP_{ABBA}(2(m - \ell + 1)), & |P_{ABBA}| \leqslant k \end{cases}, \quad (13)$$
$$k = \lceil 0.05 \cdot 2m \rceil = \lceil 0.1m \rceil.$$

2.3. Последовательный поиск сниппетов

Для дальнейшего изложения последовательного алгоритма поиска сниппетов SnippetFinder [9] будем использовать следующие определения и обозначения. MPdist-профилем временного ряда T длины n и временного ряда Q меньшей длины m ($m \ll n$) назовем вектор из $n-m+1$ элементов, каждый из которых представляет собой величину схожести соответствующей подпоследовательности ряда T, имеющей длину m, с рядом Q, также имеющим длину m, в смысле меры MPdist, и обозначим его как MPD:

$$MPD(Q, T, \ell) = (d_1, \ldots, d_{n-m+1}),\ d_i = \text{MPdist}(Q, T_{i,m}, \ell). \quad (14)$$

Обозначим MPdist-профиль ряда T и его сегмента S_i как D_i, тогда набор MPdist-профилей ряда T и всех его сегментов обозначим как D и определим следующим образом:

$$D = (D_1, \ldots, D_{n/m}),\ D_i = MPD(S_i, T, \ell). \quad (15)$$

Поиск сниппетов связан с построением кривой репрезентативности, которая состоит из $n - m$ точек и обозначается как M. Параметром построения данной кривой является D_{subset}, заданное непустое подмножество набора MPdist-профилей. Кривая M представляет собой набор точек,

в котором i-я точка показывает схожесть по мере MPdist между i-й подпоследовательностью ряда, имеющей длину m, и наиболее похожим на нее сегментом ряда, взятым из заданного подмножества сегментов:

$$\begin{gathered}\{d_i \mid d_i \in D_j\},\ D_{subset} \subset D.M(D_{subset}) = (M_1, \ldots, M_{n-m+1}),\\ M_i = \min_{D_j \in D_{subset}} \{d_i \mid d_i \in D_j\},\ D_{subset} \subset D.\end{gathered} \tag{16}$$

Площадь под кривой репрезентативности M обозначается как $ProfileArea$ и рассматривается как целевая функция поиска сниппетов:

$$ProfileArea(D_{subset}) = \sum_{i=1}^{n-m} M_i(D_{subset}). \tag{17}$$

Площадь под кривой репрезентативности обладает следующим интуитивно понятным свойством: при выборе в качестве сниппетов всех сегментов ряда и, соответственно, построении кривой репрезентативности по всем сегментам ряда, $ProfileArea = 0$.

Поиск сниппетов подразумевает такой выбор сокращенного множества сегментов ряда в качестве сниппетов, при котором значение $ProfileArea$ существенно приблизится к нулю. Поиск сниппетов реализуется как итеративный подбор множества сегментов ряда следующим образом. На первом шаге в качестве сниппета алгоритмом будет выбран сегмент, для которого значение $ProfileArea$ минимально:

$$step = 1: \quad C_1.index = \arg\min_{1 \leqslant j \leqslant n/m} ProfileArea(\{D_j\}). \tag{18}$$

На каждом следующем шаге MPdist-профиль сегмента, выбранного в качестве сниппета на предыдущем шаге, используется для построения кривой репрезентативности:

$$step = 2: \quad C_2.index = \arg\min_{1 \leqslant j \leqslant n/m} ProfileArea(\{D_{C_1.index},\ D_j\}). \tag{19}$$

Все последующие шаги выполняются аналогично второму шагу алгоритма до тех пор, пока не будет найдено K сниппетов, где число снип-

петов является параметром алгоритма:

$$step = i, 3 \leqslant i \leqslant K :$$
$$C_i.index = \arg \min_{1 \leqslant j \leqslant n/m} ProfileArea(\{D_{C_1.index}, \ldots, D_{C_{i-1}.index}, D_j\}). \tag{20}$$

После того, как сниппеты найдены, значимость каждого из них вычисляется следующим образом:

$$C_i.frac = \frac{|M(\{D_{C_1.index}, \ldots, D_{C_i.index}\}) \cap D_{C_i.index}|}{n-m+1}, \ 1 \leqslant i \leqslant K. \tag{21}$$

Алгоритм 2 показывает псевдокод последовательного поиска сниппетов описанным выше способом. Алгоритм SnippetFinder начинает работу с вычисления набора MPdist-профилей всех сегментов ряда (строка 2). Вспомогательные алгоритмы GetAllProfiles и MPdistProfile для вычисления набора MPdist-профилей и вычисления MPdist-профиля отдельного сегмента соответственно приведены в алгоритм 3 и алгоритм 1 соответственно. Далее производится поиск сниппетов по формулам 18–20 (строки 3–11). Алгоритм завершает работу вычислением значимости найденных сниппетов по формуле 21 (строки 12–14).

Алгоритм. 1. MPdistProfile (in $T,\ Q$; out MPD)

1: $MPD \leftarrow \emptyset$
2: **for** $i \leftarrow 1$ **to** $n - \ell$ **do**
3: $\quad d_i \leftarrow \text{MPdist}(T_{i,m}, Q, \ell)$
4: $\quad MPD \leftarrow MPD \cup d_i$
5: **return** MPD

Алгоритм. 2. SNIPPETFINDER (IN $T,\ m,\ K$; OUT C_T^m)

1: $C_T^m \leftarrow \emptyset$; $M \leftarrow \overline{+\infty}$
2: $D \leftarrow$GETALLPROFILES(T, m)
3: **while** $|C_T^m| \neq K$ **do**
4: $\quad minArea \leftarrow +\infty$
5: $\quad$ **for** $i \leftarrow 1$ **to** n/m **do**
6: $\quad\quad ProfileArea \leftarrow \sum_{j=1}^{n-m} \min(D_i(j), M_j)$
7: $\quad\quad$ **if** $ProfileArea < minArea$ **then**
8: $\quad\quad\quad minArea \leftarrow ProfileArea$; $idx \leftarrow i$
9: $\quad M \leftarrow (\min(D_{idx}(1), M_1), \ldots, \min(D_{idx}(n-m), M_{n-m})$
10: $\quad C \leftarrow T_{m\cdot(idx-1)+1,\,m}$; $C.index \leftarrow idx$
11: $\quad C_T^m \leftarrow C_T^m \cup C$
12: **for** $i \leftarrow 1$ **to** K **do**
13: $\quad f \leftarrow |\{neighbor \in D_{C_i.index} \mid neighbor = M_i\}|$
14: $\quad C_i.frac \leftarrow f/(n-m+1)$
15: **return** C_T^m

Алгоритм. 3. GETALLPROFILES (IN $T,\ m$; OUT D)

1: $D \leftarrow \emptyset$
2: **for** $i \leftarrow 1$ **to** n/m **do**
3: $\quad D_i \leftarrow$ MPDISTPROFILE$(T, T_{m\cdot(i-1)+1,\,m})$
4: $\quad D \leftarrow D \cup D_i$
5: **return** D

3. ПАРАЛЛЕЛЬНЫЙ АЛГОРИТМ ПОИСКА ЛЕЙТМОТИВОВ ВРЕМЕННОГО РЯДА

3.1. Описание архитектуры NVIDIA CUDA

Графический процессор (GPU) компании NVIDIA [18] представляет собой один из наиболее популярных в настоящее время многоядерных ускорителей. GPU имеет иерархическую архитектуру и состоит из симметричных потоковых мультипроцессоров (Streaming Multiprocessor, SM), каждый из которых, в свою очередь, состоит из симметричных CUDA-ядер. Современные GPU насчитывают тысячи CUDA-ядер, способных опередить по производительности центральные процессоры на задачах, допускающих массивно-параллельные вычисления в сочетании с векторной обработкой данных.

Параллельное приложение запускается на GPU как набор нитей, где каждая нить исполняется отдельным CUDA-ядром и предусмотрена следующая иерархия нитей. Верхним уровнем иерархии является сетка нитей (grid), которая состоит из одномерного или двумерного массива симметричных блоков нитей. Блок нитей (thread block) представляет собой массив нитей, размерность которого может варьироваться от 1 до 3. Внутри блока нити разбиваются на варпы (warp) – группы по 32 нити. Нити варпа исполняются в режиме SIMT (Single Instruction Multiple Threads): каждая нить выполняет одну и ту же инструкцию над назначенной ей частью разделяемых данных. При запуске приложения блоки нитей распределяются для исполнения между потоковыми мультипроцессорами и выполняются далее параллельно без возможности их синхронизации. Нити в пределах блока допускают синхронизацию и имеют доступ к разделяемой памяти блока. Для обмена данными между нитями разных блоков используется глобальная память GPU.

Параллельное приложение реализуется как набор вычислительных

CUDA-ядер. Вычислительное CUDA ядро (kernel) представляет собой функцию языка программирования C, которая исполняется на GPU. Запуск вычислительных ядер осуществляется с центрального процессора (называемого хостом). Помимо параметров подпрограммы, для вычислительного ядра также указываются параметры его запуска на графическом процессоре: используемое количество блоков, нитей и объем разделяемой памяти.

3.2. Структуры данных, принципы распараллеливания и реализация

Вычислительная схема алгоритма PSF имеет следующие отличия от оригинального алгоритма SnippetFinder. Вычисление набора MPdist-профилей (алгоритм 2, строка 2) выполняется более эффективно, чем в оригинальном вспомогательном алгоритме GetAllProfiles (см. алгоритм 3). Вместо одного последовательного шага, на котором выполняется вычисление MPdist-профиля между сегментом и каждой подпоследовательностью исходного ряда, мы можем выполнить последовательность из следующих шагов, каждый из которых может быть выполнен параллельно. Опишем указанную последовательность шагов для фиксированного сегмента ряда $S \in S_T^m$.

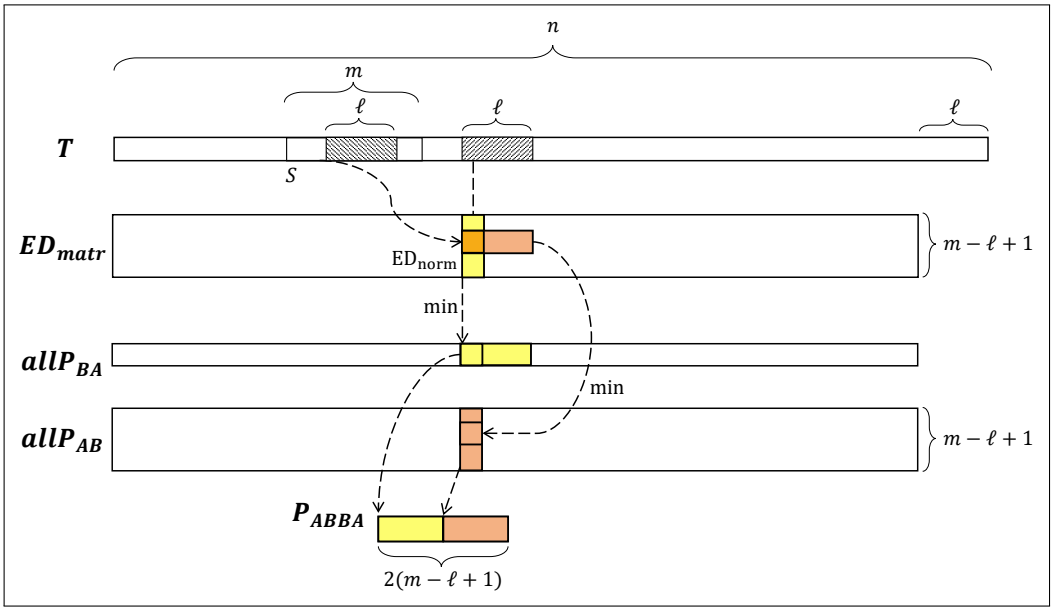


Рисунок 1 – Структуры данных алгоритма PSF

Структуры данных алгоритма PSF представлены на рисунке 1. Ключевой для распараллеливания структурой данных является матрица ED_{norm}-расстояний между каждой подпоследовательностью длины ℓ сегмента S и каждой подпоследовательностью длины ℓ исходного ряда. Обозначим указанную матрицу за ED_{matr}:

$$ED_{matr} \in \mathbb{R}^{(m-\ell+1)\times(n-\ell+1)} : \; ED_{matr}(i,j) = \text{ED}_{\text{norm}}(S_{i,\ell}, T_{j,\ell}). \tag{22}$$

На втором шаге в каждом столбце матрицы ED_{matr}, полученной на первом шаге, находится минимум. Обозначим вектор таких минимумов за $allP_{BA}$:

$$allP_{BA} \in \mathbb{R}^{n-\ell+1} : \; allP_{BA}(j) = \min_{1 \leqslant i \leqslant m-\ell+1} ED_{matr}(i,j). \tag{23}$$

На третьем шаге в каждой строке ED_{matr} выполняется поиск минимумов в скользящем окне длины ℓ. Обозначим матрицу таких минимумов за $allP_{AB}$:

$$allP_{AB} \in \mathbb{R}^{(m-\ell+1)\times(n-\ell+1)} : \; allP_{AB}(i,j) = \min_{j \leqslant c \leqslant j+m-\ell+1} ED_{matr}(i,c). \tag{24}$$

На четвертом шаге для каждой подпоследовательности ряда, имеющей длину ℓ, и сегмента S выполняется построение матричного профиля. Для построения одного матричного профиля выполняется сцепление соответствующих данной подпоследовательности столбца матрицы $allP_{AB}$ и подпоследовательности длины $m - \ell + 1$, входящей в вектор $allP_{BA}$. Результат сцепления обозначим как вектор P_{ABBA}:

$$\begin{gathered} P_{ABBA} \in \mathbb{R}^{2(m-\ell+1)} : \\ P_{ABBA}(T_{j,\ell}) = allP_{AB}(1,j) \odot \ldots \odot allP_{AB}(m-\ell+1,j) \odot \\ \odot allP_{BA}(j) \odot \ldots \odot allP_{BA}(m-\ell+1), \end{gathered} \tag{25}$$

где $1 \leqslant j \leqslant n - \ell + 1$. Для финального вычисления меры схожести MPdist между сегментом и подпоследовательностью необходимо выполнить сор-

тировку P_{ABBA} и взять k-е значение упорядоченного массива, как определено в формуле 13.

Алгоритм. 4. ParallelGetAllProfiles (in $T,\ m$; out D)

1: $D \leftarrow \emptyset$
2: **for** $i \leftarrow 1$ **to** n/m **do**
3: $\quad ED_{matr} \leftarrow$ EDmatrSCAMP(T, S_i, ℓ)
4: $\quad$ **for** $j \leftarrow 1$ **to** $n - \ell$ **do** ▷ PARALLEL
5: $\quad\quad allP_{BA}(j) \leftarrow \min_{1 \leqslant r \leqslant m-\ell+1} ED_{matr}(r, j)$
6: $\quad$ **for** $r \leftarrow 1$ **to** $m - \ell$ **do** ▷ PARALLEL
7: $\quad\quad$ **for** $q \leftarrow 1$ **to** $n - m + 1$ **do**
8: $\quad\quad\quad allP_{AB}(r, q) \leftarrow \min_{q \leqslant p \leqslant q+\ell} ED_{matr}(r, p)$
9: $\quad D_i \leftarrow$ParallelProfile$(allP_{AB}, allP_{BA})$
10: $\quad D \leftarrow D \cup D_i$
11: **return** D

Вспомогательный алгоритм ParallelGetAllProfiles, который реализует описанные выше шаги с помощью параллелизма, представлен в алгоритм 4. Циклический вызов алгоритма EDmatrSCAMP (строка 3) обеспечивает параллельное вычисление матрицы ED_{norm}-расстояний между заданным сегментом и каждой подпоследовательностью ряда. Параллелизм вычислений реализован на основе следующей техники, предложенной в работе [22]. Сначала вычисляется матрица центрированных сумм произведений значений ряда, обозначаемая как $\overline{QT}$:

$$\overline{QT}_{i,j} = \overline{QT}_{i-1,j-1} + df_i \cdot dg_j + df_j \cdot dg_i, \tag{26}$$

где df и dg – слагаемые центрированной суммы произведений:

$$df_0 = 0,\ df_i = \tfrac{1}{2}(t_{i+m-1} - t_{i-1}),$$
$$dg_0 = 0,\ dg_i = (t_{i+m-1} - \mu_i) + (t_{i-1} - \mu_{i-1}),\ \mu_i = \tfrac{1}{m}\sum_{j=i}^{i+m} t_j. \tag{27}$$

Далее полученные результаты используются для вычисления корреляции по Пирсону между подпоследовательностями ряда, обозначаемую как $P_{i,j}$:

$$P_{i,j} = \overline{QT}_{i,j} \cdot \tfrac{1}{\|T_{i,m} - \mu_i\|} \cdot \tfrac{1}{\|T_{j,m} - \mu_j\|}, \tag{28}$$

где $T_{i,m} - \mu_i = (t_i - \mu_i, \ldots, t_{i+m-1} - \mu_i), T_{j,m} - \mu_j = (t_j - \mu_j, \ldots, t_{j+m-1} - \mu_j)$ и $\|\cdot\|$ означает Евклидову норму вектора.

Наконец, значение корреляции по Пирсону между двумя подпоследовательностями ряда можно преобразовать в z-нормализованное евклидово расстояние между ними следующим образом:

$$\mathrm{ED}_{\mathrm{norm}}(T_{i,m}, T_{j,m}) = \sqrt{2m(1 - P_{i,j})}. \tag{29}$$

Описанная техника использует меньше вычислительных ресурсов по сравнению с прямолинейным выполнением z-нормализации подпоследовательностей и вычислением евклидова расстояния между ними.

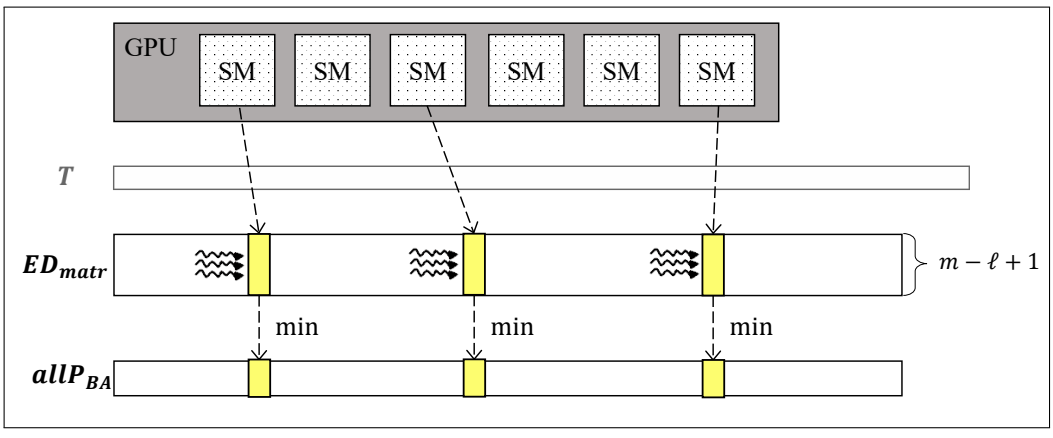


Рисунок 2 – CUDA ядро для вычисления $allP_{BA}$

Строки 4–5 алгоритма реализуют в соответствии с формулой 23 параллельное вычисление вектора $allP_{BA}$, содержащего минимумы по столб-

цам ED_{matr}. Соответствующее вычислительное ядро организуется следующим образом (рисунок 2). Формируется сетка нитей, состоящая из $n - m + 1$ блоков по $m - \ell + 1$ нитей в каждом блоке. Из глобальной памяти в разделяемую память каждого блока копируется один столбец матрицы ED_{matr}. Далее каждый блок выполняет поиск минимума посредством операции свертки.

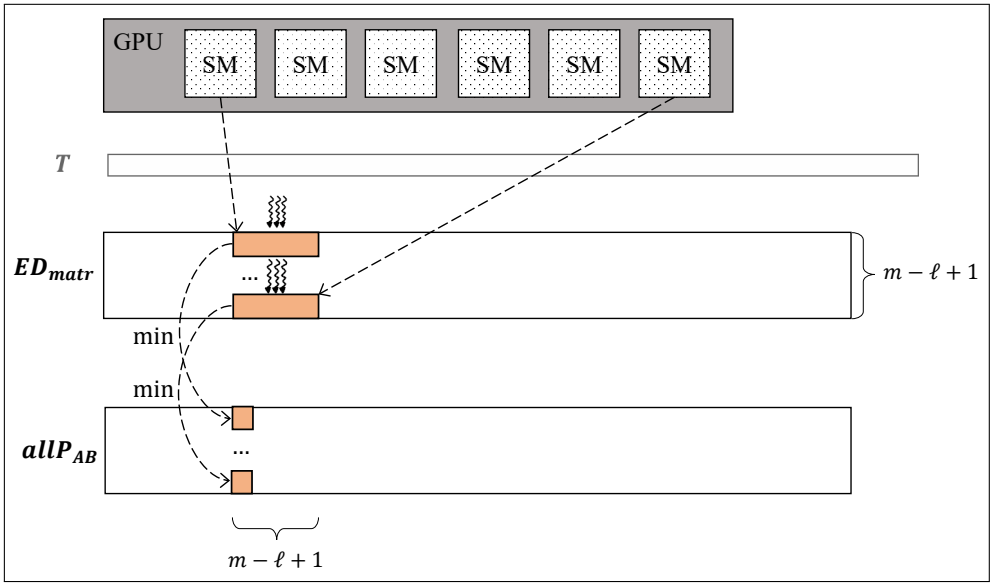


Рисунок 3 – CUDA ядро для вычисления $allP_{AB}$

Строки 6–8 реализуют в соответствии с формулой 24 параллельное вычисление матрицы $allP_{AB}$, содержащей построчные минимумы ED_{matr} в скользящих окнах длины ℓ. Соответствующее вычислительное ядро изображено на рисунке 3. Формируется сетка нитей, состоящая из одного блока с $m - \ell + 1$ нитями. Каждая нить блока выполняет вычисление минимума в скользящем окне длины ℓ для одной строки матрицы. Результирующая матрица данного шага записывается в глобальную память.

Далее вычисления продолжаются с помощью вспомогательного параллельного алгоритма ParallelProfile (см. алгоритм 5). В соответствии с формулой 25 выполняется параллельная конкатенация каждого столбца матрицы $allP_{AB}$ и всех подпоследовательностей длины $m - \ell$, входящих в вектор $allP_{BA}$.

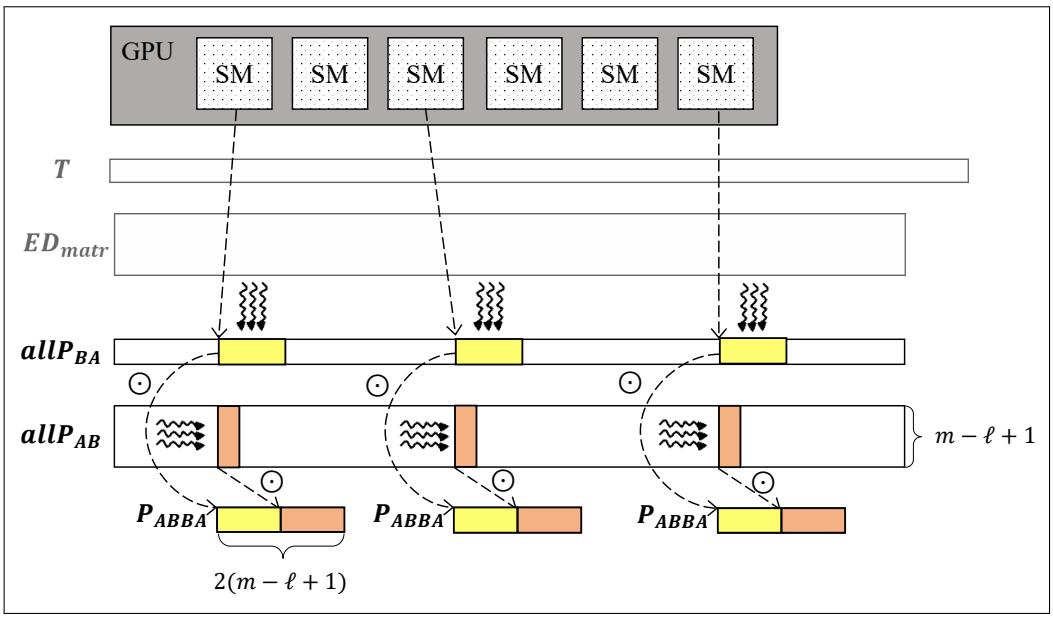


Рисунок 4 – CUDA ядро для вычисления P_{ABBA}

Соответствующее вычислительное ядро организуется следующим образом (рисунок 4). Формируется сетка нитей, состоящая из $n - m + 1$ блоков по $2(m - \ell + 1)$ нитей в каждом блоке. Каждый блок выполняет параллельное формирование матричного профиля P_{ABBA} для одного сегмента S. Половина нитей данного блока копирует из глобальной памяти в разделяемую память этого блока данные $allP_{AB}$, другая половина нитей копирует из глобальной памяти в разделяемую память этого блока данные $allP_{BA}$ для каждого столбца матрицы расстояний. Далее каждый блок производит сортировку P_{ABBA} и записывает его k-е значение (см. формулу 13) в глобальную память, формируя таким образом MPdist-профиль сегмента.

Алгоритм. 5. ParallelProfile (in $allP_{AB},\ allP_{BA}$; out P)

1: $P \leftarrow \emptyset$; $k \leftarrow \lceil 0.1 \cdot (m - \ell) \rceil$
2: **for** $i \leftarrow 1$ **to** $n - \ell$ **do** ▷ PARALLEL
3: $\quad P_{ABBA} \leftarrow allP_{AB}(i, m - \ell) \odot allP_{BA}(i)$
4: $\quad SortedP_{ABBA} \leftarrow \text{Sort}(P_{ABBA})$
5: $\quad P_i \leftarrow SortedP_{ABBA}(k)$
6: $\quad P \leftarrow P \cup P_i$
7: **return** P

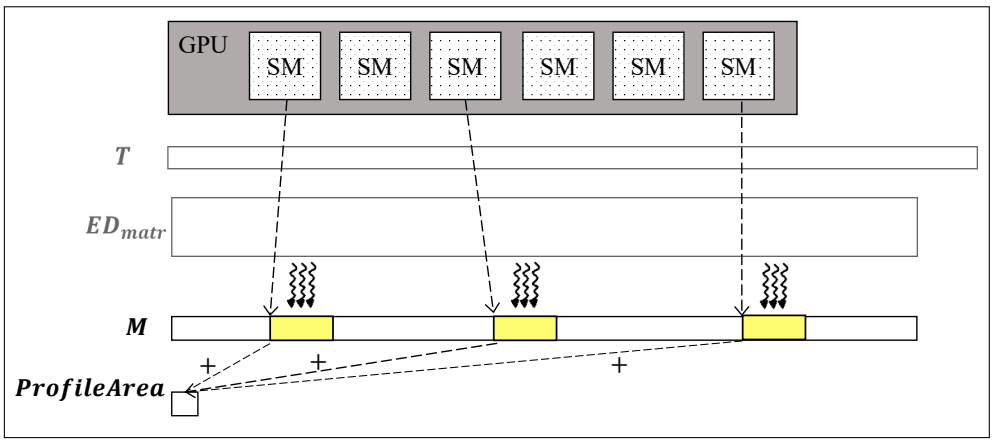


Рисунок 5 – CUDA ядро для вычисления P_{ABBA}

Распараллеливанию также подвергаются вычисление площади под кривой репрезентативности (алгоритм 2, строки 5–11) и вычисление значимости найденных сниппетов (алгоритм 2, строки 12–14). На рисунке 5 изображено соответствующее CUDA ядро.Для вычисления кривой используется сетка нитей, состоящая из n/m блоков по $n - m + 1$ нитей в каждом блоке. Каждый блок параллельно вычисляет минимальные значения кривой репрезентативности. Далее осуществляется суммирование значений кривой репрезентативности посредством операции свертки. После того, как найдено необходимое количество сниппетов, формируется сетка нитей, состоящая из K блоков по $n - m + 1$ нитей в каждом блоке. Указанная сетка путем сравнения значений MPdist-профилей сниппетов и кривой репрезентативности вычисляет значимость каждого сниппета.

4. ВЫЧИСЛИТЕЛЬНЫЕ ЭКСПЕРИМЕНТЫ

В данном разделе представлены результаты экспериментов по исследованию эффективности разработанных параллельных алгоритмов поиска типичных подпоследовательностей временного ряда для графического процессора.

4.1. Цели экспериментов

Таблица 1 – Временные ряды, задействованные в экспериментах

Название	**Длина ряда** n	**Длина сегмента** m	**Описание**
GreatBarbet	2 801	150	Физиологические показатели жизнедеятельности птиц
WildVTrainedBird	20 002	900	
SkipWalk	20 002	600	Показания носимого акселерометра во время различных видов физической активности человека
WalkRun	100 000	240	
IronAscDescWalk	87 906	2 800	
TiltABP	40 000	630	Показания кровяного давления человека во время быстрых наклонов

Для исследования эффективности разработанного параллельного алгоритма PSF нами были проведены вычислительные эксперименты. В экспериментах исследовалась производительность алгоритма при обработке временных рядов, резюмированных в таблице 1 для указанной длины сегмента m и длины значимой подпоследовательности $\ell = \lceil m/2 \rceil$. Временные ряды GreatBarbet, WildVTrainedBird, SkipWalk и TiltABP взяты из набора MixedBag [27], использованного для исследования эффективности оригинального последовательного алгоритма. В данном наборе каждый временной ряд имеет два предопределенных сниппета, которые соответствуют двум предопределенным активностям, и имеет место однократная смена указанных активностей. Временной ряд IronAscDescWalk представляет собой показания носимого акселерометра во время четырех различных видов физической активности человека (глажка белья, подъем по лестнице, спуск по лестнице, ходьба) и является фрагментом стандартизированного временного ряда PAMAP [20].

В экспериментах сравнивалась производительность предложенного параллельного алгоритма PSF, оригинального последовательного алгоритма SnippetFinder и параллельного алгоритма NaivePSF. При этом количество сниппетов соответствовало числу активностей, отражаемых временным рядом (параметр K=2 для временных рядов, взятых из набора MixedBag, и K=4 для временного ряда IronAscDescWalk).

Алгоритм *NaivePSF* представляет собой упрощенную версию PSF, в которой по формулам 8–12 вычисляются матричные профили между сегментами и всеми подпоследовательностями ряда при помощи отдельных вызовов фреймворка SCAMP [23]. Далее последовательно на центральном процессоре по формулам 13–15 вычисляются MPdist-профили всех сегментов и производится поиск сниппетов. Данный подход приводит к избыточным вычислениям. В результате работы фреймворка SCAMP производятся повторные вычисления евклидовых расстояний между подпоследовательностями (см. формулу 8) и поиск минимумов матриц расстояний (см. формулу 11).

Таблица 2 – Аппаратная платформа экспериментов

Характеристика	**Центральный процессор**	**Графический процессор**
Модель	Intel Xeon Gold 6254	NVIDIA Tesla V100 SXM2
Количество ядер	18	5120
Тактовая частота ядра, GHz	4,0	1,3
Оперативная память, GB	64	32
Пиковая производительность, TFLOPS	1,2	15,7

Аппаратная платформа экспериментов резюмирована в таблице 2. В экспериментах последовательный алгоритм SnippetFinder выполняется на центральном процессоре (на одном ядре). При выполнении параллельных алгоритмов PSF и NaivePSF предварительные вычисления средних значений подпоследовательностей (см. формулу 27) выполняются на централь-

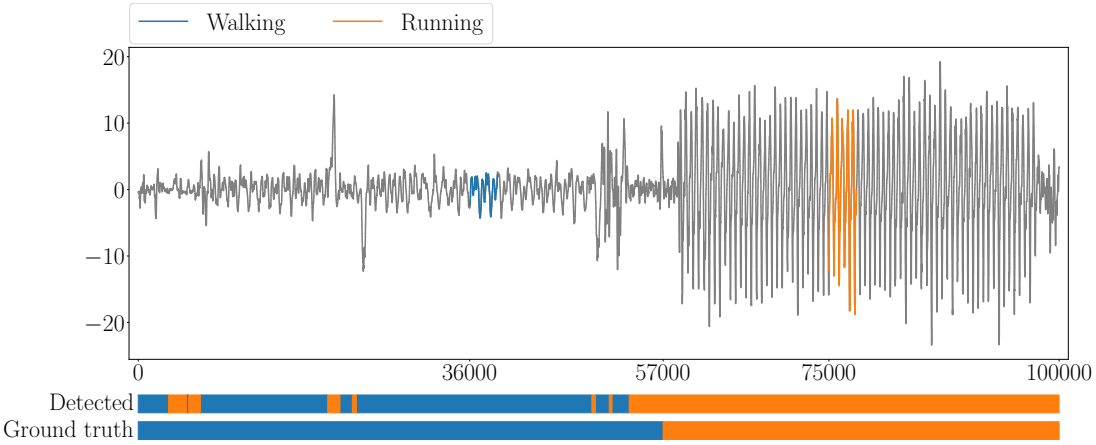
ном процессоре, остальные вычисления (формулы 8–21) выполняются на графическом процессоре.

4.2. Результаты экспериментов

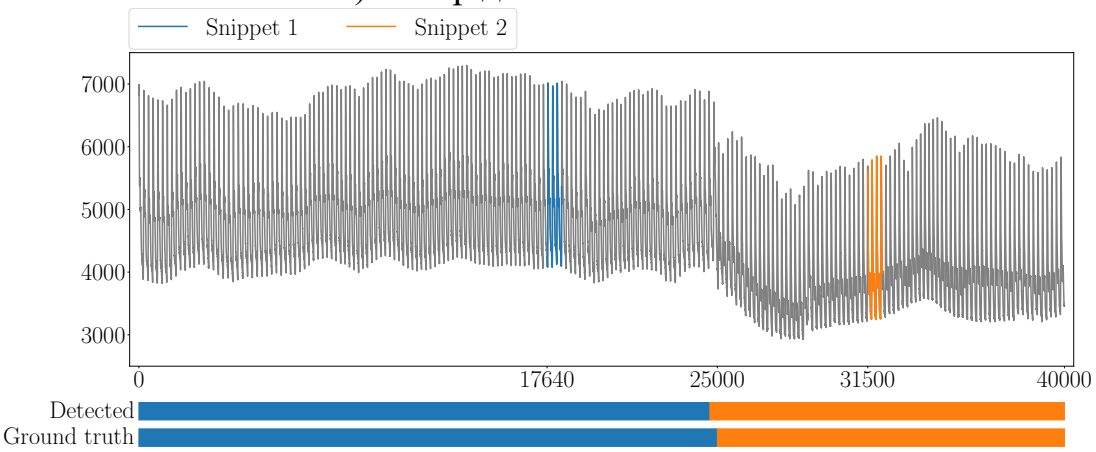
В экспериментах алгоритмы PSF и NaivePSF выдают результаты, идентичные результатам работы оригинального алгоритма SnippetFinder. На рисунке 6 приведены примеры результатов работы алгоритма PSF для временных рядов, задействованных в экспериментах. Точность поиска типичных подпоследовательностей указана в таблице 3.

Время работы алгоритмов для различных временных работ представлено на рисунке 7. Можно видеть, что параллельный алгоритм PSF уступает оригинальному последовательному алгоритму только в случае GreatBarbet, самого короткого временного ряда из рассмотренных. Это ожидаемый результат, поскольку в случае временного ряда относительно небольшой длины (примерно до десятка тысяч элементов) накладные расходы на передачу данных на GPU и инициализацию вычислительных ядер будут больше времени, затраченного на собственно вычисления. В остальных случаях PSF опережает оригинальный последовательный алгоритм минимум на порядок.

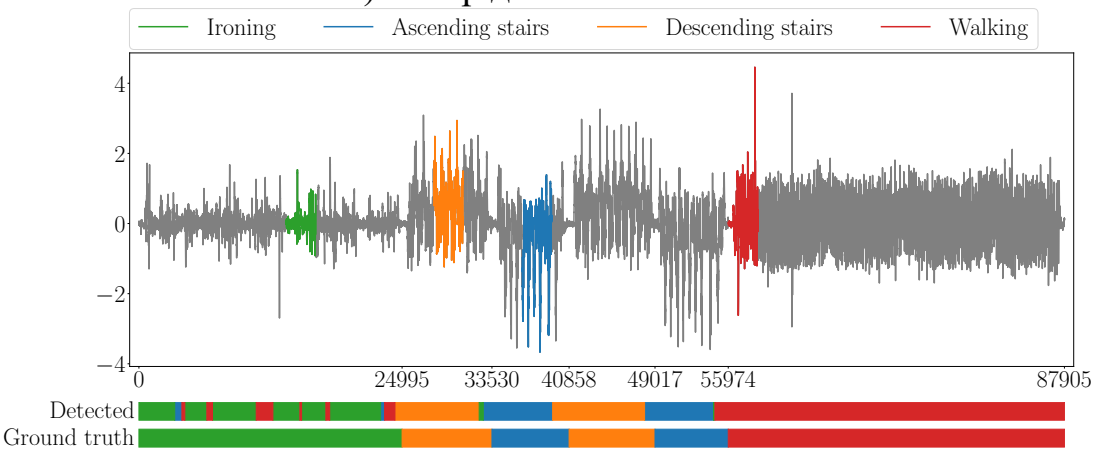
По полученным результатам экспериментов можно сделать вывод, что разработанный параллельный алгоритм имеет высокую производительность и точность, а также может применяться для временных рядов из широкого спектра предметных областей.



а) набор данных WalkRun



б) набор данных TiltABP



в) набор данных IronAscDescWalk

Рисунок 6 – Примеры результатов работы алгоритма PSF

Таблица 3 – Точность поиска сниппетов с помощью PSF

Временной ряд	Точность
GreatBarbet	0.97
WildVTrainedBird	0.94
SkipWalk	0.97
WalkRun	0.91
IronAscDescWalk	0.88
TiltABP	0.99

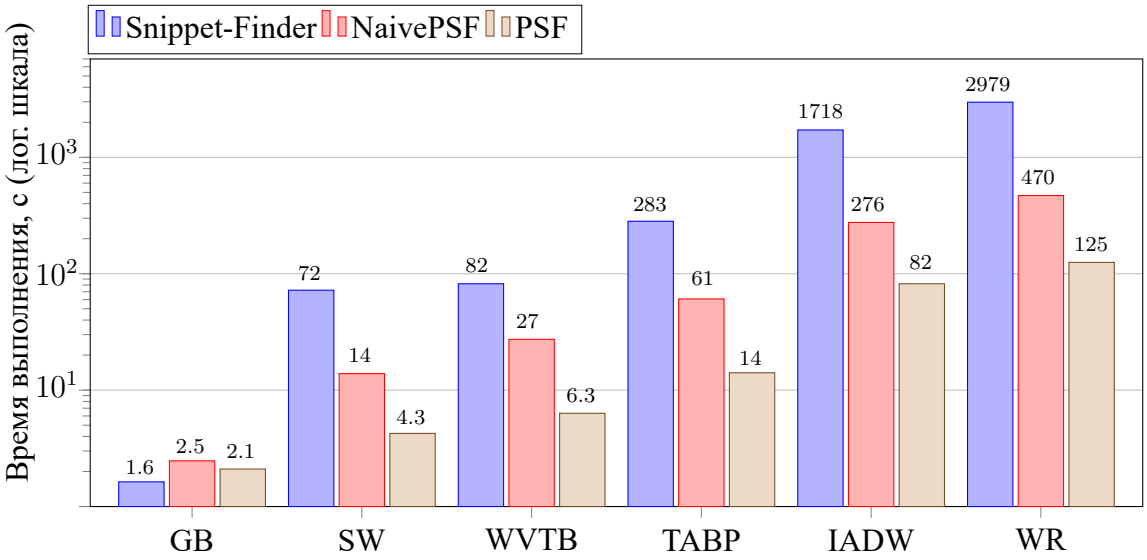


Рисунок 7 – Производительность поиска сниппетов

В случае временного ряда относительно небольшой длины (примерно до десятка тысяч элементов) PSF показывает практически одинаковое быстродействие, что и NaivePSF, поскольку накладные расходы данных алгоритмов одинаковы, а для короткого временного ряда избыточные вычисления в наивной версии параллельного алгоритма практически отсутствуют. Для временных рядов длины более десятка тысяч элементов PSF быстрее, чем NaivePSF, в 2–4 раза. Преимущество алгоритма PSF тем больше, чем больше длина исследуемого временного ряда, поскольку накладные расходы на вычисление матричных профилей между сегментами и подпоследовательностями ряда становятся более существенными.

4.2.1. Влияние длины сегмента на производительность

Были проведены эксперименты по определению зависимости производительности параллельного алгоритма от длины сегмента (параметр m) на синтетическом временном ряду RandomWalk. На рисунке 8 показаны результаты экспериментов. Можно увидеть, что соотношение показателей производительности среди алгоритмов сохранятся. Также производительность алгоритма немного повышается при увеличении длины сегмента. Так как временная сложность оригинального алгоритма составляет $O(n^2 \cdot (n - m)/m)$, где n – длина временного ряда, m – длина сегмента, общее количество операций стремится к нулю при возрастании длины сегмента.

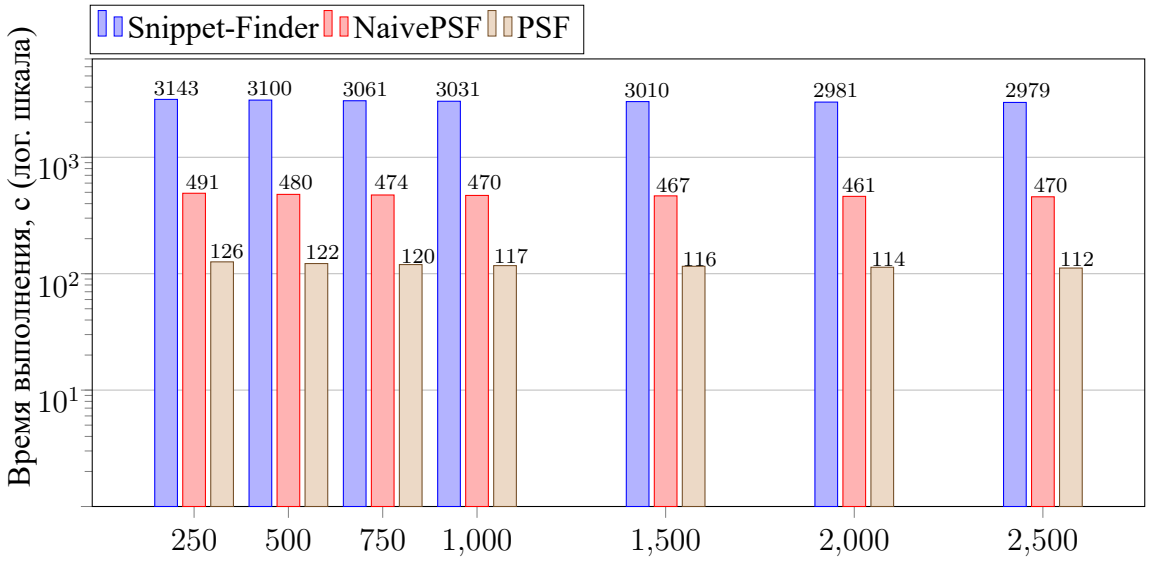


Рисунок 8 – Зависимость производительности алгоритма PSF от длины сегмента

4.2.2. Влияние длины подпоследовательности на производительность

На рисунке 9 показана зависимость производительности параллельного алгоритма от длины подпоследоватлеьности (параметр ℓ, см. раздел 2.2) для синтетического временного ряда RandomWalk с длиной сегмента $m = 2500$. Можно увидеть, что увеличение длины подпоследоватлеьности повышает производительность алгоритма. Это обусловлено тем, что использование больших значений ℓ дает меньшее количество строк в ED_{matr},

матрице расстояний между сегментами и подпоследовательностями. В результате это приводит к уменьшению структур данных $allP_{BA}$, $allP_{AB}$, $allP_{BA}$, и P_{ABBA}, на основе которых производятся вычисления в алгоритме.

Также на рисунке 10 показана средняя точность алгоритма PSF на всех временных рядах из набора данных MixedBag в зависимости от длины подпоследовательности. На диаграмме размаха изображены показатели точности для трех типичных значений параметра ℓ, откуда мы можем видеть, что $\ell = \lceil m/2 \rceil$ дает наибольшую точность.

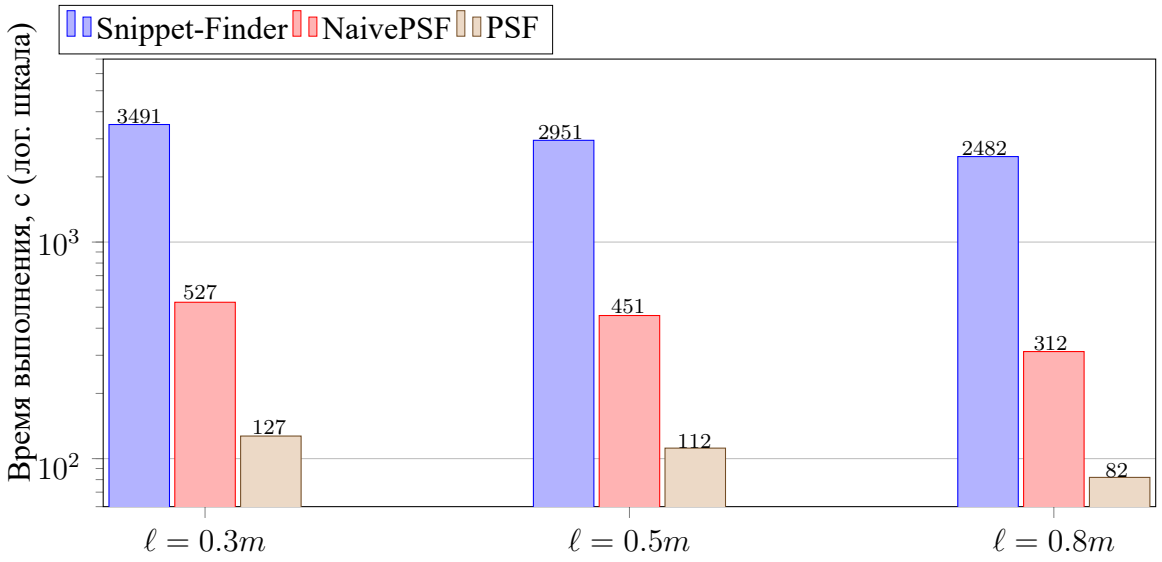


Рисунок 9 – Зависимость производительности алгоритма PSF от длины подпоследовательности

4.2.3. Применение $\mathrm{ED}^2_{\mathrm{norm}}$ вместо $\mathrm{ED}_{\mathrm{norm}}$

В экспериментах на применение $\mathrm{ED}^2_{\mathrm{norm}}$ вместо $\mathrm{ED}_{\mathrm{norm}}$ было проведено сравнение производительности и точности алгоритма PSF на всех временных ряда из набора данных MixedBag [27]. На рисунке 11 изображены диаграммы размаха точности алгоритма при использовании евклидовой метрики и ее квадрата, соответственно. По полученным результатам можно сделать вывод, что вместо метрики расстояния возможно заменить ее квадратом без существенной потери точности. $\mathrm{ED}^2_{\mathrm{norm}}$ показывает мень-

шее значение третьего квартиля и минимума в тех случаях, когда PSF имеет меньшую точность. В то же время эксперименты показали, что вычисление MPdist на основе квадрата метрики дает прирост производительности до 10% по сравнению с использованием обычного евклидова расстояния.

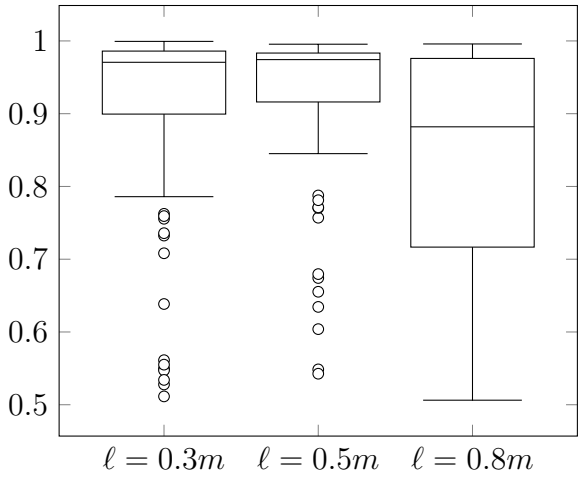


Рисунок 10 – Точность алгоритма PSF в зависимости от длины подпоследовательности

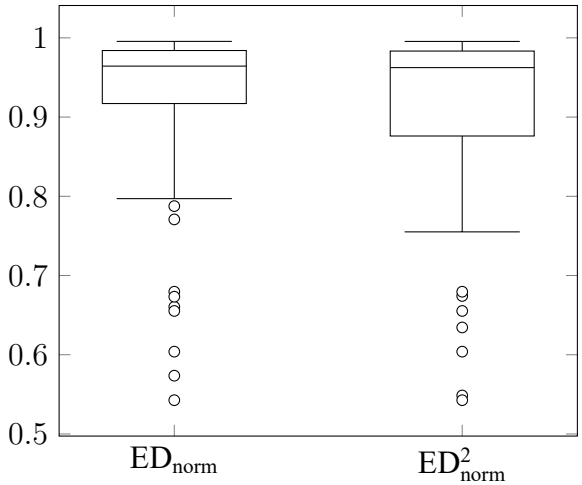


Рисунок 11 – Точность алгоритма PSF в зависимости от метрики расстояния

4.2.4. Поиск типичных подпоследовательностей показаний сенсора, установленного на промышленном оборудовании

Было проведено тематическое исследование с использованием данных виброакселерометра, установленного на малогабаритной дробильной установке. Показания записаны во время заброса двух материалов различ-

ной твердости: дунита и кирпича. Помимо дробления указанных материалов, записаны два других вида активности: установка выключена и холостой ход. Количество сниппетов соответствует общему числу активностей: $K = 4$. Длина сегмента соответствует пяти секундам: $m = 4000$. В таблице 4 представлена оценка эффективности поиска типичных подпоследовательностей.

Таблица 4 – Показатели качества поиска типичных подпоследовательностей

Активность	**Точность**	**Полнота**	**F1-мера**
Холостой ход	0.05	0.03	0.04
Дробление дунита	0.65	0.72	0.68
Дробление кирпича	0.55	0.54	0.55
Установка выключена	0.77	0.85	0.81

По данным в таблице 4 можно видеть, что дробление дунита и выключенное состояние дробилки имеют наиболее высокие точность и полноту распознавания. Интегрально наилучшим образом распознается дробление дунита, наихудшим – холостой ход. На рисунке 12 представлены исходная разметка ряда и результат поиска типичных подпоследовательностей при помощи алгоритма PSF.

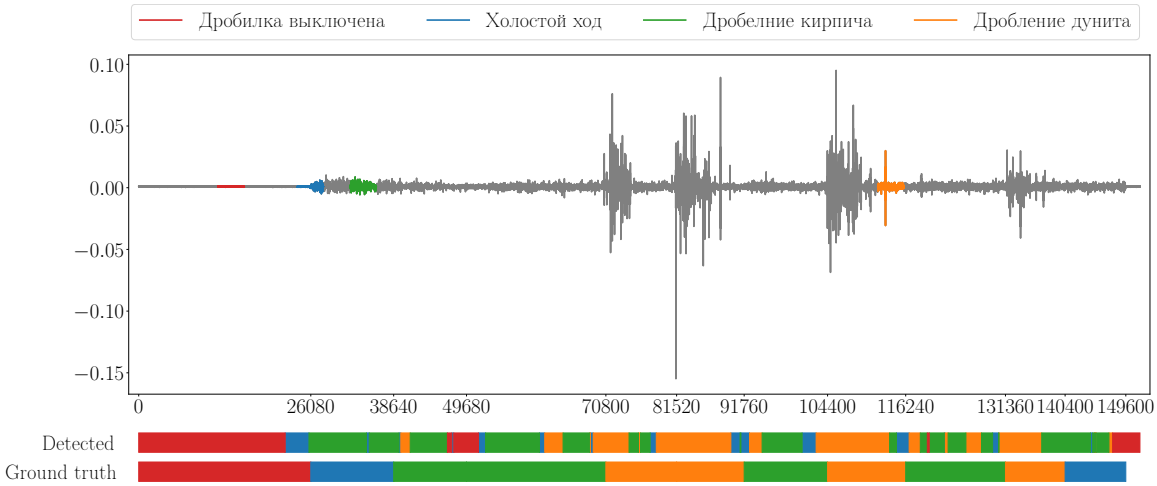


Рисунок 12 – Результат поиска типичных подпоследовательностей сенсорных данных

ЗАКЛЮЧЕНИЕ

Данная выпускная квалификационная работа была посвящена разработке параллельного алгоритма поиска типичных подпоследовательностей временного ряда для графического процессора.

В ходе выполнения работы были получены следующие основные результаты:

1) проведен обзор алгоритмов поиска типичных подпоследовательностей временного ряда;

2) изучены аппаратная архитектура и программная модель графического процессора NVIDIA;

3) спроектирован и реализован параллельный алгоритм поиска типичных подпоследовательностей для графического процессора;

4) проведены вычислительные эксперименты на реальных и синтетических данных, показавшие высокую производительность и точность разработанного алгоритма.

По результатам исследования опубликованы три научные статьи в изданиях, индексируемых в РИНЦ [2, 3], Scopus [24], и получено свидетельство о регистрации программы [1].

В рамках выпускной квалификационной работы был сделан доклад на XVI международной научной конференции «Параллельные вычислительные технологии (ПаВТ) 2022» (29 марта – 31 марта 2022 г., Дубна).

ЛИТЕРАТУРА

1. Свидетельство Роспатента о государственной регистрации программы для ЭВМ «PSF: программа для автоматического аннотирования временного ряда на графическом процессоре» № 2022619627 от 24.05.2022 г.

2. Цымблер М.Л., Гоглачев А.И. Поиск типичных подпоследовательностей временного ряда на графическом процессоре. // Вычислительные методы и программирование, 2021. – Т. 22. – № 4. – С. 344–359.

3. Цымблер М.Л., Гоглачев А.И. Применение параллельных вычислений для аннотирования сенсорных данных. // Параллельные вычислительные технологии – XVI международная конференция, ПаВТ’2022, г. Дубна, 29–31 марта 2022 г. Короткие статьи и описания плакатов. Челябинск: Издательский центр ЮУрГУ, 2022. – С. 141–148.

4. Bar-Joseph Z. A new approach to analyzing gene expression time series data. / Z. Bar-Joseph, G.K. Gerber, D.K. Gifford, T.S. Jaakkola, I. Simon. // Proceedings of the Sixth Annual International Conference on Computational Biology, RECOMB 2002, Washington, DC, USA, April 18-21, 2002. – P. 39–48.

5. Chadwick N.A., McMeekin D.A., Tan T. Classifying eye and head movement artifacts in EEG signals. // 5th IEEE International Conference on Digital Ecosystems and Technologies, IEEE DEST 2011, Daejeon, South Korea, May 31 - June 3, 2011. – P. 285–291.

6. Fu T. Pattern discovery from stock time series using self-organizing maps. / T. Fu, F. Chung, V. Ng, R. Luk. // Workshop Notes of the Workshop on Temporal Data Mining at the 7th ACM SIGKDD Int’l Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 26-29, 2001. – P. 27–37.

7. Gharghabi S. An ultra-fast time series distance measure to allow data mining in more complex real-world deployments. / S. Gharghabi, S. Imani, A.J. Bagnall, A. Darvishzadeh, E.J. Keogh. // Data Min. Knowl. Discov., 2020. – Vol. 34. – No. 4. – P. 1104–1135.

8. Han J., Kamber M. Data Mining: concepts and techniques. Morgan Kaufmann, 2006. – P. 743.

9. Imani S. Matrix Profile XIII: Time Series Snippets: A New Primitive for Time Series Data Mining. / S. Imani, F. Madrid, W. Ding, S.E. Crouter, E.J. Keogh. // 2018 IEEE International Conference on Big Knowledge, ICBK 2018, Singapore, November 17-18, 2018 / Ed. by X. Wu, Y. Ong, C.C. Aggarwal, H. Chen. – IEEE Computer Society, 2018. – P. 382–389.

10. Indyk P., Koudas N., Muthukrishnan S. Identifying Representative Trends in Massive Time Series Data Sets Using Sketches. // VLDB 2000, Proceedings of 26th International Conference on Very Large Data Bases, September 10-14, 2000, Cairo, Egypt / Ed. by A.E. Abbadi, M.L. Brodie, S. Chakravarthy, et al. – Morgan Kaufmann, 2000. – P. 363–372.

11. Keogh E.J., Lin J. Clustering of time-series subsequences is meaningless: implications for previous and future research. // Knowl. Inf. Syst., 2005. – Vol. 8. – No. 2. – P. 154–177.

12. Keogh E.J. Supporting exact indexing of arbitrarily rotated shapes and periodic time series under Euclidean and warping distance measures. / E.J. Keogh, L. Wei, X. Xi, M. Vlachos, S. Lee, et al. // VLDB J., 2009. – Vol. 18. – No. 3. – P. 611–630.

13. Kirk D.B. NVIDIA CUDA software and GPU parallel computing architecture. // Proceedings of the 6th International Symposium on Memory Management, ISMM 2007, Montreal, Quebec, Canada, October 21-22, 2007 / Ed. by G. Morrisett, M. Sagiv. – ACM, 2007. – P. 103–104.

14. Mantegna R. Hierarchical Structure in Financial Markets. // The European Physical Journal B: Condensed Matter and Complex Systems,

1999. – Vol. 11. – No. 1. – P. 193–197.

15. McGovern A. Identifying predictive multi-dimensional time series motifs: an application to severe weather prediction. / A. McGovern, D.H. Rosendahl, R.A. Brown, K. Droegemeier. // Data Min. Knowl. Discov., 2011. – Vol. 22. – No. 1-2. – P. 232–258.

16. Mueen A. Exact Discovery of Time Series Motifs. / A. Mueen, E.J. Keogh, Q. Zhu, S. Cash, M.B. Westover. // Proceedings of the SIAM International Conference on Data Mining, SDM 2009, Sparks, Nevada, USA, April 30 - May 2, 2009. – P. 473–484.

17. Müller M. Analysis and Retrieval Techniques for Motion and Music Data. // Eurographics 2009 - Tutorials, Munich, Germany, March 30 - April 3, 2009. – P. 249–255.

18. Owens J. GPU architecture overview. // International Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 2007, San Diego, California, USA, August 5-9, 2007. – P. 2.

19. Rebbapragada U. Finding anomalous periodic time series. / U. Rebbapragada, P. Protopapas, C.E. Brodley, C.R. Alcock. // Mach. Learn., 2009. – Vol. 74. – No. 3. – P. 281–313.

20. Reiss A., Stricker D. Introducing a New Benchmarked Dataset for Activity Monitoring. // 16th International Symposium on Wearable Computers, ISWC 2012, Newcastle, United Kingdom, June 18-22, 2012. – IEEE Computer Society, 2012. – P. 108–109.

21. Ye L., Keogh E.J. Time series shapelets: a new primitive for data mining. // Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Paris, France, June 28 - July 1, 2009 / Ed. by J.F.E. IV, F. Fogelman-Soulié, P.A. Flach, M.J. Zaki. – ACM, 2009. – P. 947–956.

22. Yeh C.M. Matrix Profile I: All Pairs Similarity Joins for Time Series: A Unifying View That Includes Motifs, Discords and Shapelets. / C.M. Yeh,

Y. Zhu, L. Ulanova, N. Begum, Y. Ding, et al. // IEEE 16th International Conference on Data Mining, ICDM 2016, Barcelona, Spain, December 12-15, 2016. – P. 1317–1322.

23. Zimmerman Z. Matrix Profile XIV: Scaling Time Series Motif Discovery with GPUs to Break a Quintillion Pairwise Comparisons a Day and Beyond. / Z. Zimmerman, K. Kamgar, N.S. Senobari, B. Crites, G.J. Funning, et al. // Proceedings of the ACM Symposium on Cloud Computing, SoCC 2019, Santa Cruz, CA, USA, November 20-23, 2019. – P. 74–86.

24. Zymbler M., Goglachev A. Fast Summarization of Long Time Series with Graphics Processor. // Mathematics, 2022. – Vol. 10. – No. 10.

25. Zymbler M., Kraeva Y. Discovery of Time Series Motifs on Intel Many-Core Systems. // Lobachevskii J. Math.,
2019. – Vol. 40. – No. 12. – P. 2124–2132.

26. Zymbler M., Kraeva Y. Parallel Algorithm for Time Series Motif Discovery on Graphic Processor. // Bulletin of the South Ural State University. Series: Computational Mathematics and Software Engineering,
2020. – Vol. 9. – No. 3. – P. 17–34.

27. Snippet-Finder supporting website, [Электронный ресурс]. URL: https://sites.google.com/site/snippetfinderinfo/, (дата обращения: 10.03.2022 г.).