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
Ведущий программист
ООО «ВОРТЕКСКОД»

_____ П.А. Михайлов

“____”_____ 2020 г.

ДОПУСТИТЬ К ЗАЩИТЕ

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

_____ Л.Б. Соколинский

“____”_____ 2020 г.

# Разработка параллельного алгоритма поиска лейтмотивов временного ряда для многоядерных ускорителей

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
ЮУрГУ – 02.04.02.2020.308-559.ВКР

Научный руководитель,
к.ф.-м.н., доцент
_____ М.Л. Цымблер

Автор работы,
студент группы КЭ-220
_____ Я.А. Краева

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

Челябинск-2020

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

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

**ЗАДАНИЕ**
**на выполнение выпускной квалификационной работы магистра**
студентке группы КЭ-220
Краевой Яне Александровне,
обучающейся по направлению
02.04.02 «Фундаментальная информатика и информационные технологии»
(магистерская программа «Технологии разработки высоконагруженных систем»)

1. **Тема работы** (утверждена приказом ректора от 24.04.2020 № 627)
Разработка параллельного алгоритма поиска лейтмотивов временного ряда для многоядерных ускорителей.
2. **Срок сдачи студентом законченной работы:** 05.06.2020.
3. **Исходные данные к работе**

3.1. Антонов А.С. Технологии параллельного программирования MPI и OpenMP. Учебное пособие. – М.: Издательство Московского университета, 2012. – 344 с.
3.2. Jeffers J., Reinders J., Sodani A. Intel Xeon Phi Processor High Performance Programming: Knights Landing Edition. – USA: Morgan Kaufmann, 2016. – 662 p.
3.3. Cheng J., Grossman M., McKercher T. Professional CUDA C Programming. 1st Edition. – United Kingdom: Wrox, 2014. – 528 p.
3.4. Farber R. Parallel Programming with OpenACC. 1st Edition. – USA: Morgan Kaufmann, 2016. – 326 p.
3.5. Mueen A., Keogh E.J., Zhu Q., Cash S., Westover M.B. Exact Discovery of Time Series Motifs // Proceedings of the SIAM International Conference on Data Mining, SDM 2009, Sparks, Nevada, USA, April 30 – May 2, 2009. – SIAM, 2009. – P. 473–484.

**4. Перечень подлежащих разработке вопросов**

4.1. Провести обзор последовательных и параллельных алгоритмов поиска лейтмотивов временного ряда.

4.2. Изучить аппаратную архитектуру и программную модель многоядерного процессора семейства Intel MIC и графического процессора NVIDIA GPU.

4.3. Спроектировать и реализовать параллельный алгоритм поиска лейтмотивов временного ряда для многоядерных ускорителей.

4.4. Провести вычислительные эксперименты по анализу эффективности разработанного алгоритма.

**5. Дата выдачи задания:** 09.02.2020.

**Научный руководитель,**
к.ф.-м.н., доцент М.Л. Цымблер

**Задание принял к исполнению** Я.А. Краева

**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 [17].

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 [31], компьютерная обработка речи и музыки [38], финансовое прогнозирование [15, 29], мониторинг показателей функциональной диагностики организма человека в медицине [8], исследования ДНК в биологии [3], классификация звезд в астрономии [21, 47] и др.

Поиск лейтмотивов временного ряда является одной из основных задач интеллектуального анализа временных рядов. *Лейтмотив временного ряда (time series motif)* представляет собой пару подпоследовательностей этого ряда заданной длины, наиболее похожих друг на друга [44]. В настоящее время поиск лейтмотивов является актуальной задачей в широком спектре приложений обработки временных рядов: биоинформатика [5], обработка речи [2], прогнозирование природных катаклизмов [31], неврология [37] и др. Кроме того, поиск лейтмотивов временного ряда часто используется как подпрограмма во многих задачах интеллектуального анализа временных рядов: классификации [22, 59], кластеризации [46], прогнозировании [53], визуализации [24], поиске аномалий [37], поиске ассоциативных правил [51] и др.

Поиск лейтмотивов методом полного перебора имеет временную сложность $O(n^2)$, где $n$ – длина временного ряда [44]. В силу этого в работах [10, 32, 33, 50, 52] был предложен ряд алгоритмов поиска *приближенного лейтмотива*, имеющих меньшую временную сложность $O(n)$ и $O(n \log n)$. Однако для ряда приложений, например, в сейсмологии [60],

недопустима потеря точности результирующего лейтмотива, даже за счет выигрыша во времени поиска. Алгоритм МК [37] является одним из самых быстрых последовательных алгоритмов поиска *точного лейтмотива* и сокращает время поиска до трех раз по сравнению с другими алгоритмами, однако его производительность значительно снижается на временных рядах, имеющих длину от сотен тысяч элементов [37].

Одной из основных тенденций развития современной процессорной техники является увеличение количества вычислительных ядер вместо тактовой частоты [12]. В настоящее время ускорители архитектур Intel MIC (Many Integrated Core) [11] и NVIDIA GPU (Graphics Processing Unit) [42] обеспечивают от сотен до тысяч процессорных ядер и значительно опережают традиционные процессоры по производительности. В соответствии с этим актуальной является задача разработки параллельных алгоритмов поиска лейтмотивов временного ряда на современных многоядерных ускорителях архитектур Intel MIC и NVIDIA GPU.

**Цель и задачи исследования**

Целью данной выпускной квалификационной работы является разработка параллельного алгоритма поиска лейтмотивов временного ряда для современных многоядерных ускорителей архитектур Intel MIC и NVIDIA GPU.

Для достижения поставленной цели необходимо решить следующие *задачи*:

1) провести обзор последовательных и параллельных алгоритмов поиска лейтмотивов временного ряда;

2) изучить аппаратную архитектуру и программную модель многоядерного процессора семейства Intel MIC и графического процессора NVIDIA GPU;

3) выполнить проектирование и реализацию параллельного алгоритма поиска лейтмотивов временного ряда для многоядерных ускорителей;

4) провести вычислительные эксперименты, исследующие эффективность разработанного алгоритма.

**Структура и объем работы**

Работа состоит из введения, пяти глав, заключения и списка используемой литературы.

Объем работы составляет 48 страниц, объем библиографии – 63 наименования.

**Содержание работы**

Первая глава, «Обзор работ по тематике исследования», содержит обзор и сравнительный анализ существующих последовательных алгоритмов поиска лейтмотивов временного ряда и подходов к их распараллеливанию для различных аппаратных платформ.

Во второй главе, «Описание аппаратно-программной архитектуры многоядерных ускорителей», приведено описание аппаратной архитектуры и программная модель многоядерного процессора семейства Intel MIC и графического процессора NVIDIA GPU.

Третья глава, «Формальные обозначения и постановка задачи», содержит формальные определения и обозначения, а также краткое описание последовательного алгоритма поиска лейтмотивов MK [37], используемого в качестве основы разработки параллельного алгоритма.

В четвертой главе, «Параллельный алгоритм поиска лейтмотивов временного ряда», дано детальное описание разработанных параллельных алгоритмов для многоядерного процессора архитектуры Intel MIC и графического процессора NVIDIA GPU.

В пятой главе, «Вычислительные эксперименты», приведены результаты вычислительных экспериментов по исследованию эффективности предложенных алгоритмов.

Заключение резюмирует результаты, полученные в рамках исследования.

# 1. ОБЗОР РАБОТ ПО ТЕМАТИКЕ ИССЛЕДОВАНИЯ

## 1.1. Последовательные алгоритмы

Метод грубой силы (brute-force) [26] является одним из наиболее очевидных решений задачи поиска лейтмотивов. Данный метод выполняет перебор всех возможных пар подпоследовательностей временного ряда, вычисляя расстояние (схожесть) между подпоследовательностями. Пара подпоследовательностей с наименьшим расстоянием является результирующим лейтмотивом. Достоинство данного метода – простота реализации и гарантированное нахождение точного лейтмотива. Очевидным недостатком является высокая временная сложность, которая составляет $O(N^2 \cdot m)$, где $N$ – количество подпоследовательностей временного ряда ($N = n - m + 1$), $m$ – длина подпоследовательности.

Поскольку метод полного перебора имеет квадратичную временную сложность относительно длины временного ряда, были предложены алгоритмы поиска *приближенного лейтмотива*, в котором подпоследовательности существенно похожи друг на друга, но не обязательно являются наиболее похожими между собой во временном ряде в целом. Данные алгоритмы используют специальные техники: индексирование [25, 55], дискретизация данных [10], хеширование [60] и др., – которые позволяют снизить сложность поиска лейтмотивов до линейной или логарифмической (относительно длины временного ряда).

Один из первых алгоритмов поиска приближенных лейтмотивов временного ряда был предложен в работе [10]. Данный алгоритм основан на вероятностном методе понижения размерности LPH (Locality-preserving hashing). Особенность данного метода заключается в сохранении расположения хешей объектов относительно друг друга в пространстве меньшей размерности. Для построения хешей используется алгоритм random projection (случайное проектирование) [6]. Объекты, которые имеют одинаковый хеш, принадлежат одной группе и являются лейтмотивами.

В работах [4, 26, 44, 50] были предложены алгоритмы, которые для ускорения поиска лейтмотивов используют метод символьной агрегатной аппроксимации (SAX, Symbolic Aggregate ApproXimation) [27]. Сначала подпоследовательности временного ряда сжимаются с помощью кусочно-

агрегирующей аппроксимации (PAA, Piesewise Aggregate Approximation), затем преобразуются в символьные подпоследовательности (SAX-слова). Временная сложность данного метода является квадратичной и зависит от выбранной длины SAX-слова.

В работе [7] преобразование подпоследовательностей в набор символов выполняется с помощью алгоритма iSAX (indexable Symbolic Aggregate approXimation) [50], являющийся улучшенной версией алгоритма SAX. Временная сложность этого метода линейная.

В работах [24, 49] авторы предложили алгоритмы для нахождения приблизительных лейтмотивов различных длин, которые используют грамматическую индукцию (grammar induction) для сжатия исходных данных. На первом шаге алгоритма выполняется дискретизация всех подпоследовательностей ряда с помощью алгоритма SAX. Далее на основе последовательности, образованной в результате конкатенации SAX-слов, генерируются грамматические правила (grammar rules) с помощью алгоритма Sequitur [41]. Грамматические правила представляют собой повторяющиеся шаблоны в последовательности, которые могут быть рассмотрены в качестве лейтмотивов ряда. Правила ранжируются по длине и частоте встречаемости в последовательности из SAX-слов. Лейтмотивами будут считаться те подпоследовательности, правила которых имеют максимальное значение частоты встречаемости. Лейтмотивы отображаются обратно в исходные подпоследовательности временного ряда. Методы, основанные на грамматической индукции, имеют линейную временную сложность.

В статье [52] предлагается алгоритм MD (Motif Discovery), который сначала преобразует исходные данные в последовательность символов, а затем, используя принцип минимальной длины описания (MDL, Minimum Description Length) [48], находит лейтмотив в одномерном временном ряде. Подходящая длина лейтмотива определяется автоматически алгоритмом. Кроме того, алгоритм может находить лейтмотивы в многомерных временных рядах, используя метод главных компонент (PCA, principal component analysis) [18]. Данный алгоритм имеет квадратичную временную сложность.

Однако для ряда приложений, например, в сейсмологии [60], недо-

пустима потеря точности результирующего лейтмотива, даже за счет выигрыша во времени поиска. В соответствии с этим был разработан ряд алгоритмов, направленных на поиск *точных лейтмотивов*.

В работе [56] Вилсон и др. представили алгоритм FLAME для нахождения точного лейтмотива в ДНК. Однако алгоритм FLAME является приближенным для дискретных временных рядов, значения которых представляют собой вещественные числа.

В работе [37] Муином, Кеогом и др. предложен алгоритм MK, который находит точный лейтмотив во временном ряде. Алгоритм MK основан на следующей идее. Рассмотрим подпоследовательности исходного ряда как конечное множество точек конечномерного метрического пространства. Выберем случайным образом некую точку данного множества и назовем ее опорной. Упорядочим точки исходного множества по возрастанию их расстояния до опорной точки и назовем такой порядок линейным. Для любой пары точек, отличных от опорной, верно следующее утверждение: если точки находятся близко друг к другу в исходном пространстве, то они также будут располагаться близко в линейном порядке (обратное утверждение неверно). Данное свойство вкупе с неравенством треугольника позволяет отбрасывать пары подпоследовательностей, заведомо не являющихся лейтмотивом, без вычисления расстояния. Для большего сокращения пространства поиска алгоритм использует несколько опорных точек. Эксперименты показывают, что алгоритм MK позволяет в разы ускорить поиск лейтмотива полным перебором [37].

В работе [35] Муин, Кеог и др. продолжили свое исследование и представили алгоритм поиска лейтмотивов DAME (Disk Aware Motif Enumeration) для случая, когда временной ряд не помещается в оперативную память и хранится на диске. Алгоритм использует стратегию «разделяй и властвуй» и обеспечивает высокую производительность за счет относительно небольшого количества последовательных обращений к диску.

В работе [34] было предложено два алгоритма: SBF (Smart Brute Force) и MOEN (MOtif ENumerator), – однократный запуск каждого из которых позволяет найти все точные лейтмотивы заданного ряда, имеющие длину до $\lceil \frac{N}{2} \rceil$. Алгоритм SBF представляет собой улучшение метода пол-

ного перебора и сравнивает пары подпоследовательностей ряда в определенном порядке. Для нахождения расстояния между текущей парой подпоследовательностей SBF использует расстояние, вычисленное на предыдущем шаге алгоритма, уменьшая с помощью этого приема временную сложность вычисления расстояния с $O(m)$ до $O(1)$. Алгоритм MOEN позволяет находить неповторяющиеся лейтмотивы различных длин в больших временных рядах. Ускорение достигается благодаря использованию техники нижних границ, позволяющей отбрасывать пары подпоследовательностей, заведомо не являющихся лейтмотивами, без вычисления расстояний между подпоследовательностями. Нижняя граница расстояния между длинными подпоследовательностями вычисляется на основе расстояния между более короткими подпоследовательностями, вычисленного ранее. Алгоритм MOEN не требует входных параметров и является одним из наиболее эффективных алгоритмов обнаружения лейтмотивов, однако он имеет квадратичную временную сложность относительно длины ряда.

В работе [25] предложен двухфазный алгоритм Quick-Motif, сочетающий технику отбрасывания бесперспективных лейтмотивов [37] и технику инкрементного вычисления корреляции [34]. Сначала алгоритм с помощью преобразования PAA [28] трансформирует каждую подпоследовательность ряда в PAA-представление и группирует полученные последовательные представления в минимальный ограничительный прямоугольник (MBR, minimum bounding rectangle). Далее выполняется построение R-дерева Гильберта, с помощью которого отбрасываются бесперспективные пары прямоугольников. Благодаря комбинированию нескольких техник, алгоритм Quick-Motif опережает алгоритмы MK [37] и SBF [34] по быстродействию до трех раз и может использоваться для поиска лейтмотивов в режиме онлайн.

### 1.2. Параллельные алгоритмы

В работе [40] Наранг и др. предложили параллельный алгоритм Par-MK для точного обнаружения лейтмотива на SMP системах для случая, когда данные полностью помещаются в оперативную память. Для распараллеливания Par-MK использует нити стандарта POSIX [43]. Каждая нить об-

рабатывает часть отсортированного массива, содержащего расстояния до опорной подпоследовательности. Алгоритм также распараллеливает цикл просмотра пар подпоследовательностей, отстоящих друг от друга в линейном порядке, выполняемый по величине этого отступа. Нити, которые обрабатывают разные значения отступа, но одну и ту же часть массива расстояний, выполняются на ядрах, которые разделяют кэш второго уровня.

Для устранения дисбаланса загрузки нитей из-за непредсказуемого и неравномерного количества подпоследовательностей, отбрасываемых различными нитями, авторы предложили две версии своего алгоритма: Par-MK-SLB и Par-MK-DLB (соответственно статическая и динамическая балансировка нагрузки). В вычислительных экспериментах на реальном временном ряде длины $|T| = 1.8 \cdot 10^5$ при длине лейтмотива $m = 200$, лучшая версия алгоритма продемонстрировала сверхлинейное ускорение на 32 ядрах благодаря динамической балансировке нагрузки в сочетании с превосходной производительностью кэша второго уровня. Однако на синтетическом временном ряде длины $|T| = 5 \cdot 10^4$ при длине лейтмотива $m = 1024$ ускорение лучшей версии уменьшилось более чем в 8.5 раза из-за снижения производительности кэша за счет большей длины лейтмотива. В своей дальнейшей работе авторы планировали модернизировать алгоритм для многоядерных ускорителей, но это исследование не было проведено.

### 1.3. Алгоритмы на основе матричного профиля

Концепция матричного профиля предложена Кеогом и др. в работе [58]. Матричный профиль временного ряда представляет собой структуру данных, описывающую основные характеристики этого ряда, позволяющие решить большой набор задач интеллектуального анализа данных в указанном ряде, в т.ч. поиск лейтмотивов.

Матричный профиль заданного временного ряда определяется как временной ряд, $i$-м элементом которого является величина евклидова расстояния между $i$-й подпоследовательностью исходного ряда и ее ближайшим соседом (наиболее похожей подпоследовательностью) [58]. Вычисление матричного профиля предполагает нахождение матрицы расстояний, элементами которой являются евклидовы расстояния между всеми парами

подпоследовательностей временного ряда. Строка матрицы расстояний называется профилем расстояний (distance profile). Матричный профиль содержит минимумы каждого столбца матрицы расстояний.

В настоящее время существует три последовательных алгоритма для нахождения лейтмотивов временного ряда, которые основаны на вычислении матричного профиля: STAMP [57], STOMP [62] и SCRIMP++ [61].

Алгоритм STAMP (Scalable Time series Anytime Matrix Profile) [57] вычисляет профили расстояний в случайном порядке. Для нахождения профиля расстояния используется алгоритм MASS (Mueen's Algorithm for Similarity Search) [57]. Алгоритм MASS вычисляет z-нормализованные евклидовы расстояния между подпоследовательностями, используя их скалярные произведения. Скалярные произведения вычисляются с помощью быстрого преобразования Фурье (FFT, Fast Fourier Transform) за время $O(n \log n)$. Временная сложность алгоритма STAMP $O(n^2 \log n)$.

В отличие от STAMP, алгоритм STOMP [62] последовательно вычисляет профили расстояний. За счет использования зависимости между последовательными профилями расстояний сокращается количество повторяющихся вычислений. Временная сложность алгоритма STOMP составляет $O(n^2)$.

Алгоритм SCRIMP++ [61] состоит из двух фаз, представляющих собой отдельные алгоритмы PreSCRIMP и SCRIMP. Сначала с помощью алгоритма PreSCRIMP вычисляется приближенный матричный профиль. Алгоритм PreSCRIMP основан на технике последовательного сохранения соседства (Consecutive Neighborhood Preserving, CNP), которая заключается в следующем. Из временного ряда выбираются подпоследовательности, отстоящие друг от друга на интервал $s$. На основе вычисленного профиля расстояния для каждой выбранной подпоследовательности $T_{i,m}$ находится ее ближайший сосед $T_{j,m}$. Далее в соответствии с техникой CNP предполагается, что ближайшим соседом для подпоследовательности $T_{i+k,m}$ будет подпоследовательность $T_{j+k,m}$, $|k| < s$, где $s$ – окрестность позиции подпоследовательностей $T_{i,m}$ и $T_{j,m}$. Алгоритм вычисляет расстояния между этими парами подпоследовательностей и обновляет матричный профиль, если вычисленное расстояние меньше соответствующего элемента

матричного профиля. Далее для уточнения матричного профиля и получения результирующего лейтмотива используется алгоритм SCRIMP, который вычисляет в случайном порядке диагонали матрицы расстояний. Элементы диагонали могут быть вычислены посредством использования ранее вычисленных элементов этой же диагонали. Временная сложность алгоритма составляет $O(n^2)$.

В работе [62] предложены две параллельные версии алгоритма STOMP для графических процессоров NVIDIA: GPU-STOMP и GPU-STOMP$_{\text{OPT}}$. Для распараллеливания алгоритмов используется технология CUDA. Параллельный алгоритм GPU-STOMP включает в себя следующие шаги. Сначала с центрального процессора копируется временной ряд в глобальную память GPU и выделяется память на GPU под необходимые структуры данных. Далее центральный процессор запускает CUDA-ядро на $N$ нитях для вычисления значений средних арифметических и стандартных отклонений, скалярных произведений, инициализации матрицы расстояний, вычисления матричного профиля и индекса матричного профиля. Затем выполняется цикл по подпоследовательностям временного ряда, где вызывается два CUDA-ядра для нахождения строки матрицы расстояния и элемента матричного профиля. После завершения вычислений матричный профиль и индекс матричного профиля копируются обратно в память центрального процессора.

В оптимизированном алгоритме GPU-STOMP$_{\text{OPT}}$ решается проблема накладных расходов на запуск ядер и синхронизацию нитей. Вместо запуска CUDA-ядра для каждой строки матрицы расстояний ядро вызывается только один раз для генерации всей матрицы расстояний. Кроме того, используется иная схема вычисления матрицы расстояний: нить вычисляет диагональ матрицы расстояний. Эксперименты показали, что GPU-STOMP$_{\text{OPT}}$ обеспечивает более чем 4-кратное ускорение по сравнению с GPU-STOMP.

В работе [61] представлен алгоритм SCAMP (SCAlable Matrix Profile), который является улучшением алгоритма GPU-STOMP$_{\text{OPT}}$ [62] и заключается в следующем. На первом шаге алгоритма хост разбивает матрицу расстояний на фрагменты (tile) и помещает их в глобальную очередь.

Далее каждый рабочий берет фрагмент из очереди, вычисляет локальный матричный профиль и индекс матричного профиля и пересылает результаты мастеру. После обработки рабочими всех фрагментов хост объединяет результаты для нахождения глобального матричного профиля и индекса матричного профиля. Для проведения экспериментов алгоритм был запущен на кластере Amazon EC2 облачной платформы AWS (Amazon Web Services), узлы которого оснащены графическими процессорами NVIDIA Tesla V100. Авторы сравнили быстродействие алгоритма SCAMP и GPU-STOMP$_{OPT}$ на одной аппаратной платформе. Эксперименты показали, что алгоритм SCAMP опережает GPU-STOMP$_{OPT}$ в несколько раз благодаря введенным оптимизациям.

В работе [14] представлена параллельная реализация алгоритма SCRIMP [61] для многоядерного процессора Intel Xeon Phi KNL. Распараллеливание алгоритма основано на параллелизме по данным, векторизации вычислений и использованию гибридной архитектуры памяти, включающей память HBM с высокой пропускной способностью и память DDR4. Алгоритм SCRIMP может быть легко распараллелен, потому что вычисление диагоналей матрицы расстояний может выполняться независимо. Поскольку диагонали имеют разную длину, то, для того чтобы не было дисбаланса нагрузки между нитями, была создана очередь для динамического распределения диагоналей между нитями. Как только нить заканчивает вычисление, ей из очереди назначается новая диагональ для обработки. Авторы предложили метод для эффективного вычисления матричного профиля и индекса матричного профиля, позволяющий избежать избыточной синхронизации нитей. Кроме того, вычисления скалярных произведений и z-нормализованных евклидовых расстояний векторизуются компилятором. Чтобы получить преимущество от максимальной доступной пропускной способности памяти HBM и DDR4, приватные и наиболее часто используемые переменные хранятся в памяти HBM, а глобальные данные, предназначенные только для чтения, в памяти DDR4. Эксперименты показали, что разработанный параллельный алгоритм превосходит последовательный до 190 раз. Реализация алгоритма, использующая память HBM и DDR4, превзошла в 5 раз реализацию, использующую только DDR4.

# 2. ОПИСАНИЕ АППАРАТНО-ПРОГРАММНОЙ АРХИТЕКТУРЫ МНОГОЯДЕРНЫХ УСКОРИТЕЛЕЙ

## 2.1. Описание аппаратной архитектуры Intel MIC и технологии программирования OpenMP

Процессор Intel Xeon Phi основан на архитектуре MIC (Intel Many Integrated Core), которая выполняет более одного триллиона операций в секунду (TFLOPS) с плавающей запятой. Основой архитектуры MIC является использование большого количества энергоэффективных вычислительных ядер архитектуры x86 с малой тактовой частотой (по сравнению с обычными многоядерными процессорами Intel). Первое поколение процессоров архитектуры MIC Knights Corner (KNC) [19] и второе поколение Knights Landing (KNL) [20] состоят из 57–61 и 64–72 физических ядер соответсвенно.

Каждое ядро поддерживает технологию *Hyper-Threading* и способно выполнять до четырех потоков одновременно. Поддерживается выполнение 32- и 64-битного кода, совместимого с архитектурой Intel64. Ядро содержит 2 конвейера (U-конвейер и V-конвейер), с помощью которых может выполняться 2 инструкции за такт, 512-битный блок векторных вычислений (Vector Processor Unit, VPU), специальный набор инструкций x87 для работы с математическими вычислениями. Ядро имеет собственную двухуровневую кэш-память, состоящую из 32 Кб кэша первого уровня L1 и 512 Кб когерентного кэша второго уровня L2. Локальная память обладает высокой пропускной способностью. Все ядра процессора, контроллеры памяти GDDR5 и компонент SBOX, реализующий клиентскую логику PCI Express, соединены высокоскоростной двунаправленной кольцевой шиной.

Процессоры Intel Xeon Phi поддерживают набор *SIMD (Single Instruction, Multiple Data)* инструкций и содержат 32 векторных регистра шириной 512 бит, что позволяет загружать в них одновременно 16 32-битных или 8 64-битных целочисленных или вещественных чисел и выполнять над ними одну векторную инструкцию. Кроме того, наличие 32 512-битных векторных регистров помогает компенсировать латентность обращения к данным.

*Векторизация циклов* является одним из ключевых условий достижения высокой производительности вычислительных программ на параллельных архитектурах [1]. Векторизация циклов заключается в преобразовании компилятором последовательности скалярных операторов из тела цикла в один векторный оператор.

Эффективность векторизации циклов, однако, может быть существенно снижена в силу не выровненного доступа к данным в оперативной памяти, который порождает эффект разделения цикла (*loop peeling*) [1]. Если начальный адрес массива не выровнен на ширину векторного регистра (т.е. на количество элементов, которые могут быть загружены в векторный регистр), то компилятор разбивает цикл на три части. Первая часть итераций, которые обращаются к памяти с начального адреса до первого выровненного адреса, и третья часть итераций с последнего выровненного адреса до конечного адреса векторизуются отдельно.

Одним из наиболее популярных средств программирования многопоточных приложений на многопроцессорных системах с общей памятью является технология *OpenMP* [30]. В стандарт OpenMP входят набор директив компилятора, библиотечные процедуры и переменные окружения.

Модель вычислений в OpenMP основана на порождении и объединении нитей (*fork-join*). Программа начинается выполнением одной нити, называемой нитью-мастером (master). При входе в параллельную область нить-мастер порождает семейство дочерних (slave) нитей. Все порожденные нити, включая нить-мастер, исполняют один и тот же код параллельной области. Когда нити группы завершают инструкции в параллельной области, они синхронизируются неявным барьером и уничтожаются. Только нить-мастер продолжает выполнение после завершения параллельной области. В результате такого подхода программа представляется в виде набора последовательных (однопотоковых) и параллельных (многопотоковых) участков программного кода.

Использование в технологии OpenMP потоков для организации параллелизма позволяет учесть преимущества многопроцессорных вычислительных систем с общей памятью. Прежде всего, потоки одной и той же параллельной программы выполняются в общем адресном пространстве,

что обеспечивает возможность использования общих данных для параллельно выполняемых потоков без каких-либо трудоемких межпроцессорных передач сообщений (в отличие от процессов в технологии MPI для систем с распределенной памятью). Также данные могут хранится в локальной памяти нити. Такие данные уничтожаются при выходе программы из параллельной области. Кроме того, управление потоками требует меньше накладных расходов для ОС по сравнению с процессами.

Для указания компилятору, как организовать параллельное выполнение некоторого блока программного кода, применяются специальные директивы, которые начинаются с *#pragma omp*.

## 2.2. Описание аппаратной архитектуры NVIDIA GPU и технологии программирования OpenACC

Графический процессор (GPU) компании NVIDIA [42] 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, соответствующим всем нитям, является *сетка нитей (grid)*, которая состоит из одномерного или двумерного массива симметричных блоков нитей. *Блок нитей (thread block)* представляет собой $d$-мерный ($1 \leqslant d \leqslant 3$) массив нитей. Внутри блока нити логически разделяются на группы по 32 нити – *варпы (warp)*. Нити варпа исполняются в режиме *SIMT (Single Instruction Multiple Threads)*, когда каждая нить выполняет одну и ту же инструкцию над собственной порцией общих данных.

При запуске приложения блоки нитей распределяются для исполне-

ния между потоковыми мультипроцессорами и выполняются далее параллельно без возможности их синхронизации. Нити в пределах блока допускают синхронизацию и имеют доступ к разделяемой памяти, отведенной для данного блока. Для передачи данных между потоками, принадлежащих разным блокам, используется глобальная память ускорителя.

В настоящее время для графических процессоров разработаны технологии параллельного программирования CUDA [9], OpenCL [39] и OpenACC [13], последняя из которых применена в данном исследовании. *OpenACC* представляет собой открытый стандарт параллельного программирования для создания гетерогенных параллельных программ, задействующих как центральный, так и графический процессоры. Стандарт включает библиотеку функций, переменные окружения и директивы компилятора для определения участков исходного кода программы, которые необходимо выполнить на графическом процессоре.

Модель исполнения OpenACC-приложения предусматривает иерархию нитей и соответствующие уровни параллелизма. Одна или более нитей составляют *бригаду (gang)*, исполняемую на одном потоковом мультипроцессоре. В рамках бригады определяется одна или более симметричных групп нитей – *рабочих (worker)*. Внутри рабочего нити исполняются в режиме SIMT, обеспечивая еще один, *векторный (vector)* уровень параллелизма.

Одной из основных директив компилятора в технологии OpenACC является *#pragma acc parallel loop*, которая распараллеливает цикл с фиксированным количеством повторений, равномерно распределяя итерации цикла между нитями бригад для исполнения (при отсутствии между итерациями цикла зависимостей по данным). Указанная директива может быть дополнена одним или несколькими ключевыми словами *gang*, *worker*, *vector*, которые обяжут компилятор применить в данном цикле соответствующие уровни параллелизма.

# 3. ФОРМАЛЬНЫЕ ОБОЗНАЧЕНИЯ И ПОСТАНОВКА ЗАДАЧИ

## 3.1. Формальные обозначения

Введем формальные определения и обозначения, которые используются в алгоритме, выполняющий поиск лейтмотивов временного ряда.

*Временной ряд* представляет собой хронологически упорядоченную последовательность числовых значений. Формальное определение временного ряда представлено в формуле (1). Число $n$ обозначается $|T|$ и называется длиной временного ряда.

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

*Подпоследовательность* $T_{i,\,m}$ временного ряда $T$ представляет собой непрерывное подмножество $T$, состоящее из $m$ элементов и начинающееся с позиции $i$ (см. формулу (2)):

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

В качестве *функции расстояния* между подпоследовательностями ряда возьмем неотрицательную симметричную функцию Dist, которая задана формулой (3):

$$\text{Dist} : \mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R}. \tag{3}$$

Пара непересекающихся подпоследовательностей $\{T_{i,\,m},\ T_{j,\,m}\}$ длины $m$ временного ряда $T$ называется *лейтмотивом (motif)*, если они имеют наибольшую схожесть по сравнению с другими парами подпоследовательностей $\{T_{a,\,m},\ T_{b,\,m}\}$ этого временного ряда (см. формулу (4)):

$$\begin{gathered} \forall a, b, i, j\ \ \text{Dist}(T_{i,\,m},\ T_{j,\,m}) \leqslant \text{Dist}(T_{a,\,m},\ T_{b,\,m}), \\ |i - j| \geqslant w,\ |a - b| \geqslant w,\ w > 0, \end{gathered} \tag{4}$$

где $w$ – параметр, определяющий минимальный промежуток, на который должны отстоять друг от друга подпоследовательности в лейтмотиве. Дан-

ный параметр позволяет отбрасывать лейтмотивы, которые состоят из взаимопересекающихся подпоследовательностей и потому не имеют практической ценности [44].

В качестве функции расстояния между двумя подпоследовательностями $T_{i,\,m}$ и $T_{j,\,m}$ используется *z-нормализованное евклидово расстояние* $\mathrm{ED}_{\mathrm{norm}}$ [36]. Данная функция расстояния $\mathrm{ED}_{\mathrm{norm}}$: $\mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R}$ задает метрику на $m$-мерном метрическом пространстве и позволяет быстро вычислять евклидово расстояние между подпоследовательностями без предварительного выполнения z-нормализации. *Z-нормализованное евклидово расстояние* $\mathrm{ED}_{\mathrm{norm}}$ между двумя подпоследовательностями $T_{i,\,m}$ и $T_{j,\,m}$ вычисляется по формуле (5):

$$\mathrm{ED}_{\mathrm{norm}}\left(T_{i,\,m}, T_{j,\,m}\right) = \sqrt{2m\Big(1 - \frac{QT_{i,j} - m\mu_i\mu_j}{m\sigma_i\sigma_j}\Big)}, \tag{5}$$

где $m$ – длина подпоследовательности,
$\mu_i$ и $\mu_j$ – среднее арифметическое $T_{i,\,m}$ и $T_{j,\,m}$,
$\sigma_i$ – стандартное отклонение $T_{i,\,m}$ и $T_{j,\,m}$,
$QT_{i,j}$ – скалярное произведение $T_{i,\,m}$ и $T_{j,\,m}$.

Среднее арифметическое $\mu_i$ и стандартное отклонение $\sigma_i$ подпоследовательности $T_{i,\,m}$ вычисляется по формулам (6) и (7) соответственно:

$$\mu_i = \frac{1}{m}\sum_{k=0}^{m-1} t_{i+k}, \tag{6}$$

$$\sigma_i = \sqrt{\frac{1}{m}\sum_{k=0}^{m-1} t_{i+k}^2 - \mu_i^2}. \tag{7}$$

Скалярное произведение $QT_{i,j}$ двух подпоследовательностей $T_{i,\,m}$ и $T_{j,\,m}$ вычисляется по формуле (8):

$$QT_{i,j} = \sum_{k=0}^{m} t_{i+k}t_{j+k}. \tag{8}$$

### 3.2. Последовательный алгоритм

Алгоритм MK [37] представляет собой один из самых быстрых последовательных алгоритмов для нахождения точного лейтмотива во временном ряде. Алгоритм сокращает пространство поиска лейтмотивов на основе введения т.н. опорных подпоследовательностей.

*Опорная подпоследовательность* представляет собой случайно выбранную подпоследовательность исходного ряда. Алгоритм вычисляет расстояние от каждой подпоследовательности ряда до опорной и упорядочивает все подпоследовательности в порядке возрастания вычисленных расстояний. Полученный порядок называется *линейным*. Затем используется следующее свойство: если два объекта конечномерного метрического пространства близки друг другу, то они также должны быть близки в линейном порядке (обратное утверждение неверно) [37]. В соответствии с неравенством треугольника расстояние между подпоследовательностями лейтмотива в линейном порядке является нижней границей расстояния между этими подпоследовательностями в пространстве $\mathbb{R}^m$ (см. формулу (9)):

$$\text{ED}(ref,\, T_{i,m}) - \text{ED}(ref,\, T_{j,m}) \leqslant \text{ED}(T_{i,m},\, T_{j,m}),\ |i-j| \geqslant w,\ w > 0, \quad (9)$$

где $ref$ – опорная подпоследовательность,
$\{T_{i,m},\, T_{j,m}\}$ – пара подпоследовательностей, отличных от $ref$.

Далее для того, чтобы различать два вышеупомянутых вида расстояний, расстояние между подпоследовательностями в пространстве $\mathbb{R}^m$ мы будем называть *истинным расстоянием*.

Переменная алгоритма $bsf$ *(best-so-far)* представляет собой текущее минимальное истинное расстояние между подпоследовательностями лейтмотива, и обновляется алгоритмом, как только найдена пара подпоследовательностей, истинное расстояние между которыми меньше, чем $bsf$.

Просматривая подпоследовательности ряда в соответствии с их линейным порядком, алгоритм вычисляет нижние границы. Если нижняя граница превышает $bsf$, то истинное расстояние также превысит этот порог. Поэтому пара соответствующих подпоследовательностей заведомо не яв-

ляется лейтмотивом и может быть отброшена без вычисления истинного расстояния. Если пара подпоследовательностей не была отброшена, выполняется вычисление истинного расстояния. Если полученное значение меньше $bsf$, пороговое значение заменяется вычисленным.

Действуя таким образом, алгоритм просматривает все возможные пары подпоследовательностей, которые отстоят друг от друга в линейном порядке на величину $offsef$ ($1 \leqslant offset \leqslant N - 1$). Просмотр продолжается до тех пор, пока не будет достигнуто такое значение $offset$, для которого не существует пар подпоследовательностей, нижние границы которых больше, чем $bsf$, после чего алгоритм завершается.

Для получения более узких нижних границ, позволяющих отбрасывать больше бесперспективных пар подпоследовательностей, алгоритм использует более одной опорной подпоследовательности. Опорная подпоследовательность с наибольшим стандартным отклонением используется для сортировки по возрастанию расстояний между этой опорной подпоследовательностью и всеми прочими подпоследовательностями исходного ряда. Если хотя бы одна из нижних границ больше, чем $bsf$, то соответствующая пара подпоследовательностей отбрасывается. Алгоритм завершается, если все нижние границы всех пар подпоследовательностей, отстоящих друг от друга на величину $offset$, больше, чем $bsf$.

Число опорных подпоследовательностей берется существенно меньшим, чем количество подпоследовательностей в исходном ряде. Как показывают эксперименты [37], количество опорных подпоследовательностей в диапазоне от 5 до 60 обеспечивает стабильное сокращение времени поиска по сравнению с другими алгоритмами в 2–3 раза независимо от типа данных и длины временного ряда, а также длины искомого лейтмотива.

# 4. ПАРАЛЛЕЛЬНЫЙ АЛГОРИТМ ПОИСКА ЛЕЙТМОТИВОВ ВРЕМЕННОГО РЯДА

## 4.1. Реализация структур данных

Вектор средних арифметических подпоследовательностей представляет массив $M \in \mathbb{R}^N$, содержащий средние арифметические всех подпоследовательностей временного ряда.

Вектор стандартных отклонений подпоследовательностей представляет массив $\Sigma \in \mathbb{R}^N$, содержащий стандартные отклонения всех подпоследовательностей временного ряда.

Обозначим количество опорных подпоследовательностей за $r$ $(0 < r \ll N)$. Тогда *матрица опорных подпоследовательностей* $Ref \in \mathbb{R}^{r \times (m+pad)}$ определяется по формуле (10) следующим образом:

$$Ref(i, \cdot) = T_{i_k, m} : \; T_{i_k, m} \in T,\, 1 \leqslant k \leqslant r,\, i_k = random(1..N), \quad \forall p \neq q \; i_p \neq i_q. \tag{10}$$

*Индекс опорных подпоследовательностей* $I_{Ref} \in \mathbb{N}^r$ для каждой опорной подпоследовательности хранит позицию ее начала во временном ряде.

*Матрица расстояний* $D \in \mathbb{R}^{r \times N}$ предназначена для хранения истинного расстояния между каждой опорной подпоследовательностью и каждой подпоследовательностью ряда и определяется по формуле (11) следующим образом:

$$D(i, j) = \text{ED}_{\text{norm}}(T_{I_{Ref}(i), m}, T_{j, m}). \tag{11}$$

*Вектор стандартных отклонений опорных подпоследовательностей* представляет собой массив $SD \in \mathbb{R}^r$, который для каждой опорной подпоследовательности содержит стандартное отклонение ее истинного расстояния до каждой подпоследовательности ряда.

*Индекс стандартных отклонений* представляет собой массив $I_{SD} \in \mathbb{N}^r$, содержащий уникальные числа в диапазоне от 1 до $r$, где числа соответствуют номерам опорных подпоследовательностей в $Ref$, упорядоченных по убыванию их стандартного отклонения.

*Индекс подпоследовательностей* $I_S \in \mathbb{N}^N$ представляет собой массив, который содержит позиции подпоследовательностей во временном ряде $T$, упорядоченные по возрастанию их истинных расстояний до опорной подпоследовательности, имеющей наибольшее стандартное отклонение.

*Индекс лейтмотива* $I_M \in \mathbb{N}^{N\times 2}$ предназначен для хранения позиций двух подпоследовательностей в $T$, которые являются потенциальным лейтмотивом и в индексе подпоследовательностей $I_S$ отстоят друг от друга на величину $offset$.

*Матрица нижних границ* $LB \in \mathbb{R}^{r\times N}$ содержит значения нижних границ между каждой опорной подпоследовательностью и каждым возможным лейтмотивом и в соответствии с (9) вычисляется по формуле (12):

$$LB(i,j) = |D\big(I_{SD}(i), I_M(j,1)\big) - D\big(I_{SD}(i), I_M(j,2)\big)|. \qquad (12)$$

*Битовая карта* представляет собой массив $B \in \mathbb{B}^N$, в котором для каждого потенциального лейтмотива хранится результат конъюнкции проверок превышения текущим значением порога $bsf$ каждой из нижних границ. Если элемент битовой карты равен `FALSE`, то соответствующий лейтмотив отбрасывается без вычисления истинного расстояния. Битовая карта определяется следующим образом (см. формулу (13)):

$$B(i) = \wedge_{j=1}^{r}(LB(i,j) < bsf). \qquad (13)$$

### 4.2. Параллельный алгоритм поиска лейтмотивов для многоядерного процессора архитектуры Intel MIC

Разработанный алгоритм поиска лейтмотива (см. алгоритм 1) состоит из двух стадий: предварительная обработка данных и нахождение лейтмотива.

**Алгоритм 1.** PhiMotifDiscovery (in $T,\ n,\ r$; out $motif$)

▷ **Фаза предварительной обработки данных**

$Ref \leftarrow r$ случайно выбранных подпоследовательностей из $T$

$I_{Ref} \leftarrow r$ индексов элементов в $Ref$

$M, \Sigma \leftarrow$ ComputeMeanStd($T$, $m$)

$D \leftarrow$ CalculateDistances($T$, $M$, $\Sigma$, $I_{Ref}$)

$SD \leftarrow$ CalculateStdDev($D$); $I_{SD} \leftarrow$ Sort($SD$); $I_S \leftarrow$ Sort($D(I_{SD}(1), \cdot)$)

$bsf \leftarrow \min(d_{i,j})$

▷ **Фаза поиска лейтмотива**

#pragma omp parallel

$start \leftarrow thread_{num} \cdot N$; $end \leftarrow thread_{num} \cdot (N + 1)$

#pragma omp for schedule (dynamic)

**for all** $offset \in 1..N$ **do**

$bsf_{local} \leftarrow bsf$; $motif_{local} \leftarrow \{\infty; \infty; bsf_{local}\}$

$I_M \leftarrow$ GeneratePairs($I_S$, $offset$)

$LB \leftarrow$ CalculateLowerBounds($D$, $I_{Ref}$)

$B(i) \leftarrow \wedge_{i=start}^{end} LB(i, j) < bsf$

$abandon \leftarrow \vee_{i=start}^{end} B(i)$; $abandon \leftarrow$ **not** $abandon$

▷ **Отбрасывание бесперспективных кандидатов**

**if** $abandon$ **then**

#pragma omp cancel for

**else**

**for all** $i \in start..end$ **do**

**if** $B(i)$ **then**

$QT \leftarrow$ ComputeDotProduct($T_{I_M(i,1),m}$, $T_{I_M(i,2),m}$)

$d \leftarrow \mathrm{ED}_{\mathrm{norm}}(QT, M, \Sigma, I_M)$

**if** $d < bsf_{local}$ **then**

$bsf_{local} \leftarrow d$; $motif_{local} \leftarrow \{I_M(i, 1); I_M(i, 2); bsf_{local}\}$

#pragma omp critical

**if** $bsf_{local} < bsf$ **then**

$bsf \leftarrow bsf_{local}$; $motif \leftarrow motif_{local}$

#pragma omp cancellation point for

**return** $motif$

### 4.2.1. Предварительная обработка данных

На этапе предварительной обработки данных формируются вектора $M$ и $\Sigma$, содержащие средние арифметические и стандартные отклонения подпоследовательностей соответственно, матрица случайно выбранных опорных подпоследовательностей $Ref$ и его индекс $I_{Ref}$. Затем алгоритм вычисляет матрицу расстояний $D$ и вектор стандартных отклонений $SD$. Далее формируются индекс подпоследовательностей $I_S$ и индекс стандартных отклонений $I_{SD}$.

Порог $bsf$ инициализируется минимумом матрицы расстояний $D$. Вышеописанные вычисления реализованы в виде распараллеленных циклов, которые легко векторизуются компилятором.

### 4.2.2. Поиск лейтмотива

На этапе обнаружения алгоритм выполняет поиск лейтмотива в цикле по переменной смещения $offset$. Распараллеливание цикла осуществляется с помощью стандартной директивы OpenMP *#pragma omp for*. Отбрасывание бесперспективных лейтмотивов приводит к неравномерной вычислительной загрузке нитей. Таким образом, чтобы повысить эффективность параллельного алгоритма, используется директива *#pragma schedule (dynamic)*, обеспечивающая динамическое разбиение итераций цикла между нитями. Каждая из следующих структур данных, а именно индекс $I_M$, матрица нижних границ $LB$ и битовая карта $B$, разбивается на сегменты равного размера, и в процессе сканирования каждый сегмент обрабатывается отдельной нитью. Каждая нить вычисляет свой собственный локальный порог $bsf_{local}$, инициализированный значением $bsf$.

Каждая нить выполняет следующую последовательность действий. Сначала индекс лейтмотива $I_M$ заполняется индексами подпоследовательностей возможного лейтмотива, которые смещены друг от друга на $offset$ в индексе $I_S$. Затем, согласно неравенству треугольника, вычисляются нижние границы для потенциальных лейтмотивов и заполняется матрица нижних границ $LB$. Далее вычисляется битовая карта $B$ на основе матрицы нижних границ. Если дизъюнкция всех элементов битовой карты равна *TRUE* (то есть все нижние границы для всех пар подпоследовательно-

стей, находящихся друг от друга на значение $offset$, больше, чем текущий $bsf$), то лейтмотив найден, и остальные кандидаты могут быть отброшены. Выход из внешнего цикла осуществляется с помощью директив OpenMP *#pragma omp cancel for* и *#pragma omp cancellation point for*. Как только найден лейтмотив, то нить, которая первая достигла директивы *#pragma omp cancel for*, завершает цикл и активирует директиву *#pragma omp cancellation point for*. Остальные нити, дойдя до активной директивы *#pragma omp cancellation point for*, также выходят из цикла. Если лейтмотив не найден, то алгоритм вычисляет истинное расстояние между подпоследовательностями в каждом возможном лейтмотиве, для которого соответствующий элемент битовой карты равен *TRUE*. В случае, если истинное расстояние меньше, чем $bsf_{local}$, то $bsf_{local}$ обновляется значением истинного расстояния. Затем алгоритм находит $bsf$ как минимум всех значений $bsf_{local}$ и соответственно обновляет лейтмотив. Критическая секция используется для правильного обновления общей переменной $bsf$. Вычисление нижних границ, битовой карты, евклидова расстояния и критерия остановки поиска лейтмотива эффективно векторизуется компилятором, что повышает общую производительность разработанного алгоритма.

### 4.3. Параллельный алгоритм для поиска лейтмотива для графического процессора NVIDIA GPU

Предлагаемая параллельная реализация поиска лейтмотива временного ряда представлена в алгоритме 2. Входными данными алгоритма являются временной ряд $T$, длина искомого лейтмотива $m$, минимальный промежуток между подпоследовательностями лейтмотива $w$ и количество опорных подпоследовательностей $r$. Алгоритм возвращает найденный лейтмотив в виде кортежа, состоящего из двух подпоследовательностей и величины расстояния между ними.

Алгоритм выполняется следующим образом. Сначала на центральном процессоре формируются основные структуры данных: временной ряд $T$, матрица опорных подпоследовательностей $Ref$, а также их индексы $I_S$ и $I_{Ref}$ соответственно, – а затем с помощью директивы OpenACC *#pragma acc data* передает их на графический процессор. Затем выполня-

ются фазы предварительной обработки данных и нахождения лейтмотива, рассматриваемые ниже в разделах 4.3.1 и 4.3.2 соответственно.

**Алгоритм 2.** GPUMotifDiscovery (in $T$, $m$, $w$, $r$; out $motif$)

▷ **Инициализация**

$Ref \leftarrow r$ случайно выбранных подпоследовательностей из $T$
$I_{Ref} \leftarrow r$ индексов элементов в $Ref$
#pragma acc data create($M$, $\Sigma$, $LB$, $SD$, $B$, $I_M$) copyin($T$, $D$, $I_{Ref}$, $I_S$, $N$, $m$, $w$, $r$) copyout($motif$)
{

▷ **Фаза предварительной обработки данных**

$M, \Sigma \leftarrow$ ComputeMeanStd($T$, $m$)
$D \leftarrow$ CalculateDistances($T$, $M$, $\Sigma$, $I_{Ref}$)
$SD \leftarrow$ CalculateStdDev($D$)
$I_{SD} \leftarrow$ Sort($SD$); $I_S \leftarrow$ Sort($D(I_{SD}(1), \cdot)$)
$bsf \leftarrow$ InitBSF($D$, $I_{Ref}$, $bsf$)
$I_M(\cdot, 1) \leftarrow$ GeneratePairs($I_S$, $offset$)

▷ **Фаза поиска лейтмотива**

**for all** $offset \in 1..N$ **do**
  $abandon \leftarrow$ `FALSE`
  $I_M(\cdot, 2) \leftarrow$ GeneratePairs($I_S$, $offset$)
  $LB \leftarrow$ CalculateLowerBounds($D$, $I_{Ref}$, $I_M$)
  $B, abandon \leftarrow$ Verify($LB$, $I_M$, $B$, $abandon$, $bsf$)
  **if** $abandon$ **then**
    **break**
  **else**
    $bsf, motif \leftarrow$ UpdateBSF($T$, $M$, $\Sigma$, $B$, $I_M$, $bsf$, $motif$)

}
**return** $motif$

### 4.3.1. Предварительная обработка данных

Фаза предварительной обработки данных предполагает вычисление структур данных, описанных выше в разделе 4.1: векторов средних ариф-

метических $M$ и стандартных отклонений $\Sigma$ подпоследовательностей временного ряда $T$, матрица расстояний $D$, вектор стандартных отклонений опорных подпоследовательностей $SD$ и индекс стандартных отклонений $I_{SD}$, а также вычисление начального значения порога $bsf$ и генерация индексов левых частей предполагаемых лейтмотивов $I_M(\cdot, 1)$.

Нахождение векторов средних арифметических $M$ и стандартных отклонений $\Sigma$ подпоследовательностей временного ряда $T$ (см. алгоритм 3) реализуется с помощью двух вложенных циклов. Внешний цикл по подпоследовательностям временного ряда распараллеливается на уровне бригады с помощью директивы *#pragma acc parallel loop gang*. Внутренние циклы по элементам подпоследовательностей, выполняющий вычисление среднего арифметического и стандартного отклонения в соответствии с (6) и (7), распараллеливаются на уровне вектора с помощью директивы *#pragma acc parallel loop vector*.

**Алгоритм 3.** ComputeMeanStd (in $T$, $m$; out $M$, $\Sigma$)

#pragma acc parallel loop gang
**for all** $i \in 1..N$ **do**
  $mean \leftarrow 0$; $std \leftarrow 0$
  #pragma acc loop vector reduction(+: $mean$, $std$)
  **for all** $j \in 1..m$ **do**
    $mean \leftarrow mean + T(i+j)$
    $std \leftarrow std + T(i+j)^2$
  $mean \leftarrow \frac{mean}{m}$; $std \leftarrow \frac{std}{m}$; $std \leftarrow \sqrt{(std - mean^2)}$
  $M(i) \leftarrow mean$
  $\Sigma(i) \leftarrow std$
**return** $M$, $\Sigma$

Вычисление матрицы расстояний (см. алгоритм 4) предполагает два вложенных цикла, внешний из которых выполняется параллельно бригадами нитей, а внутренний распараллеливается на векторном уровне. Поскольку циклы независимы и количество итераций во внутреннем цикле больше, чем во внешнем, используется стандартный прием, обеспечива-

ющий бóльшую степень параллелизма: свертка указанных циклов в один, выполняемая добавлением в директиву распараллеливания внешнего цикла атрибута *collapse(2)*. Внутренний цикл, выполняющий вычисление скалярного произведения двух подпоследовательностей, распараллеливается с помощью стандартной директивы компилятора OpenACC *#pragma acc parallel loop vector* с использованием атрибута *reduction*, который суммирует все найденные нитями устройства частичные суммы.

**Алгоритм 4.** CalculateDistances (in $S_T^m$, $I_{Ref}$; out $D$)

```
#pragma acc parallel loop gang collapse(2)
for all i ∈ 1..r do
    for all j ∈ 1..N do
        d ← 0; QT ← 0
        #pragma acc loop vector reduction(+: QT)
        for all k ∈ 1..m do
            QT ← QT + T(I_Ref(i) + k) · T(j + k)
        D(i, j) ← ED_norm(QT, M, Σ, I_Ref)
return D
```

Вычисление вектора стандартных отклонений (см. алгоритм 5) организуется как два вложенных цикла: внешний – по опорным подпоследовательностям, внутренний – по элементам матрицы расстояний. Поскольку количество опорных подпоследовательностей существенно меньше, чем количество нитей, запущенных на графическом процессоре (см. раздел 3.2), распараллеливанию подвергается только внутренний цикл, на бригадном и векторном уровнях.

В инициализации $bsf$ минимумом матрицы расстояний $D$ (см. алгоритм 6) применяется двухуровневое распараллеливание вычислений (бригадное и векторное), а также прием свертки циклов, рассмотренный выше.

### 4.3.2. Поиск лейтмотива

Фаза поиска лейтмотива представляет собой просмотр пар подпоследовательностей, отстоящих друг от друга на $offset$ позиций и реализуется с помощью цикла по указанной переменной. На каждой итерации цикла

индекс лейтмотива $I_M(\cdot, 2)$ заполняется индексами подпоследовательностей из правой части возможного лейтмотива, которые смещены на $offset$ позиций относительно индекса $I_M(\cdot, 1)$ в индексе $I_S$.

**Алгоритм 5.** CalculateStdDev (in $D$; out $SD$)

**for all** $i \in 1..r$ **do**
  $mean \leftarrow 0$; $std \leftarrow 0$
  #pragma acc parallel loop gang vector reduction(+: $mean$, $std$)
  **for all** $j \in 1..N$ **do**
    $mean \leftarrow mean + D(i, j)$
    $std \leftarrow std + D(i, j)^2$
  $mean \leftarrow \frac{mean}{N-1}$; $std \leftarrow \frac{std}{N-1}$
  $SD(i) \leftarrow \sqrt{(std - mean^2)}$
**return** $SD$

**Алгоритм 6.** InitBSF (in $D$, $I_{Ref}$; out $bsf$)

$bsf \leftarrow +\infty$
#pragma acc parallel loop gang vector collapse(2) reduction(min: $bsf$)
**for all** $i \in 1..r$ **do**
  **for all** $j \in 1..N$ **do**
    **if** $|j - I_{Ref}(i)| \geqslant w$ **then**
      $bsf \leftarrow \min(bsf,\ D(i, j))$
**return** $bsf$

Затем в соответствии с неравенством треугольника осуществляется параллельное вычисление нижних границ потенциальных лейтмотивов и заполнение матрицы нижних границ $LB$ (см. алгоритм 7), выполняемые с помощью двухуровневое распараллеливания и свертки циклов.

Далее осуществляется проверка потенциальных лейтмотивов путем вычисления битовой карты и нахождение условия останова поиска лейтмотива $abandon$ (см. алгоритм 8). Если дизъюнкция всех элементов битовой карты дает в результате *TRUE* (все нижние границы для всех пар подпоследовательностей, отстоящих друг от друга на $offset$ позиций, больше,

чем текущее значение порога $bsf$), то результирующий лейтмотив найден, а остальные кандидаты могут быть отброшены. Проверка реализуется посредством двух вложенных циклов: внешний – по потенциальным лейтмотивам и внутренний – по элементам матрицы нижних границ. Внешний цикл распараллеливается на уровнях бригад и вектора, а с помощью конструкции *reduction* обеспечивается нахождение условия останова алгоритма $abandon$. Поскольку каждая порождаемая внешним циклом нить вычисляет несколько элементов битовой карты, для корректного результата внутренний цикл должен исполняться последовательно, что обеспечивается директивой компилятора *#pragma acc parallel seq*.

**Алгоритм 7.** CalculateLowerBounds (in $D$, $I_{Ref}$, $I_M$; out $LB$)

#pragma acc parallel loop gang vector collapse(2)
**for all** $i \in 1..r$ **do**
  **for all** $j \in 1..N - offset - 1$ **do**
    $LB(i,j) \leftarrow |D\big(I_{Ref}(i), I_M(j,1)\big) - D\big(I_{Ref}(i), I_M(j,2)\big)|$
**return** $LB$

**Алгоритм 8.** Verify (in $LB$, $I_M$, $bsf$; in out $B$, $abandon$)

#pragma acc parallel loop gang vector reduction(OR: $abandon$)
**for all** $i \in 1..N - offset - 1$ **do**
  $B(i) \leftarrow$ TRUE
  **if** $|I_M(i,1) - I_M(i,2)| \geqslant w$ **then**
    #pragma acc loop seq
    **for all** $j \in 1..r$ **do**
      $B(i) \leftarrow B(i)$ **and** $(LB(j,i) < bsf)$
    $abandon \leftarrow abandon$ **or** $B(i)$
$abandon \leftarrow$ **not** $abandon$
**return** $B$, $abandon$

Если описанная выше процедура проверки не выявила лейтмотив, то выполняется обновление порога $bsf$ следующим образом (см. алгоритм 9). Перед нахождением истинного расстояния, вычисляется скалярное произ-

ведение двух подпоследовательностей в каждом потенциальном лейтмотиве, у которого соответствующий элемент битовой карты равен *TRUE*. Вычисление скалярного произведения распараллеливается на векторном уровне. Далее вычисляется истинное расстояние в соответствии с (5). Порог обновляется вычисленным значением истинного расстояния, если оно меньше текущего значения $bsf$. Цикл, выполняющий вычисление истинного расстояния, распараллеливается на уровнях бригады и рабочего.

**Алгоритм 9.** UpdateBSF (in $T$, $M$, $\Sigma$, $B$, $I_M$; in out $bsf$; out $motif$)

```
#pragma acc parallel loop gang worker
for all i ∈ 1..N − offset − 1 do
    if B(i) and |I_M(i,1) − I_M(i,2)| ⩾ w then
        d ← 0; QT ← 0
        #pragma acc loop vector reduction(+: QT)
        for all k ∈ 1..m do
            QT ← QT + T(I_M(i,1) + k) · T(I_M(i,2) + k)
        d ← ED_norm(QT, M, Σ, I_M)
        if bsf > d then
            bsf ← d
            motif ← {I_M(i,1); I_M(i,2); bsf}
return bsf, motif
```

## 5. ВЫЧИСЛИТЕЛЬНЫЕ ЭКСПЕРИМЕНТЫ

В данном разделе представлены результаты экспериментов по исследованию эффективности разработанных параллельных алгоритмов поиска лейтмотивов временного ряда для многоядерного процессора Intel Xeon Phi и графического процессора, описанных в разделе 4.

### 5.1. Цели экспериментов

Для исследования эффективности разработанного алгоритма были проведены вычислительные эксперименты. В качестве аппаратной платформы экспериментов использовались вычислительные мощности суперкомпьютера «Торнадо ЮУрГУ» [23] и графический процессор NVIDIA GeForce RTX 2080 Ti [63], характеристики которых представлены в таблице 1.

**Табл. 1.** Технические характеристики аппаратной платформы экспериментов

| Характеристика | Хост | Intel MIC | NVIDIA GPU |
|---|---|---|---|
| Модель | X5680 | Xeon Phi SE10X | GeForce GTX 2080Ti |
| Кол-во физ. ядер | 2×6 | 61 | 4 352 (68 SMs) |
| Гиперпоточность | 2× | 4× | — |
| Кол-во лог. ядер | 24 | 244 | — |
| Тактовая частота, ГГц | 3.33 | 1.1 | 1.35 |
| Пиковая произв-ть, TFLOPS | 0.371 | 1.076 | 11 |
| Память, Гб | 24 | 8 | 11 |

В экспериментах исследовались производительность и масштабируемость алгоритмов поиска лейтмотивов временного ряда. Под производительностью понимается время работы алгоритма без учета времени загрузки данных в память и выдачи результата. Масштабируемость параллельного алгоритма означает его способность адекватно адаптироваться к увеличению параллельно работающих вычислительных элементов (процессов, процессоров, нитей и др.) и характеризуется ускорением и параллельной эффективностью, которые определяются следующим образом [54].

Ускорение и эффективность параллельного алгоритма, выполняемого на $k$ вычислительных элементах, рассчитываются по формулам (14)

и (15) соответственно следующим образом:

$$s(k) = \frac{t_1}{t_k}, \tag{14}$$

$$e(k) = \frac{s(k)}{k}, \tag{15}$$

где $t_1$ – время выполнения последовательного алгоритма,
$t_k$ – время выполнения параллельного алгоритма на $k$ вычислительных элементах,
$k$ – количество вычислительных элементов, на которых выполняется параллельный алгоритм.

Вычислительными элементами в версии для многоядерного процессора Intel Xeon Phi являются нити, для графического процессора – потоковые мультипроцессоры.

В экспериментах использовались синтетический и реальный временные ряды, состоящие из $4 \cdot 10^5$ элементов. Генерация синтетического временного ряда осуществлена на основе модели случайных блужданий (Random Walk) [45]. Реальный временной ряд взят из работы [16] и представляет собой сигналы ЭКГ, снятые с дискретизацией 128 Гц. Количество опорных подпоследовательностей в экспериментах взято $r = 10$.

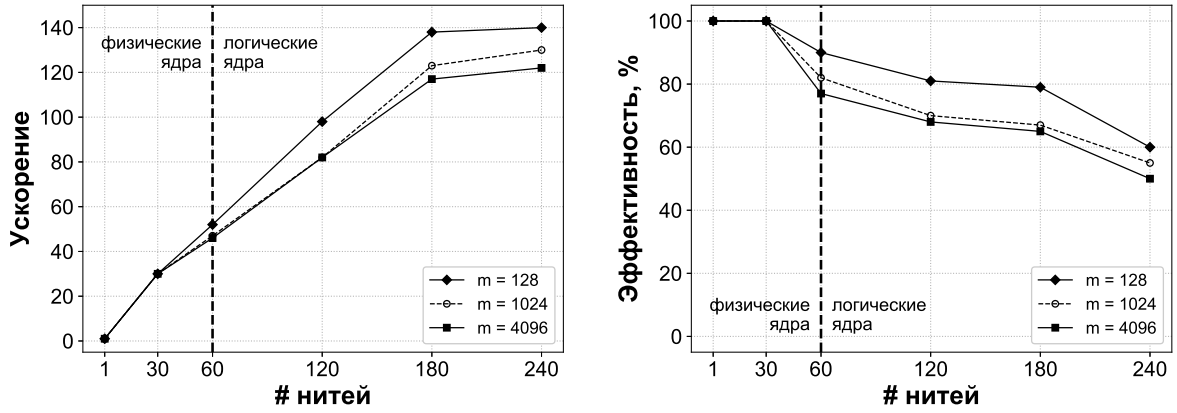
### 5.2. Результаты экспериментов

Результаты экспериментов по исследованию масштабируемости алгоритма для многоядерного процессора Intel Xeon Phi представлены на рисунке 1.

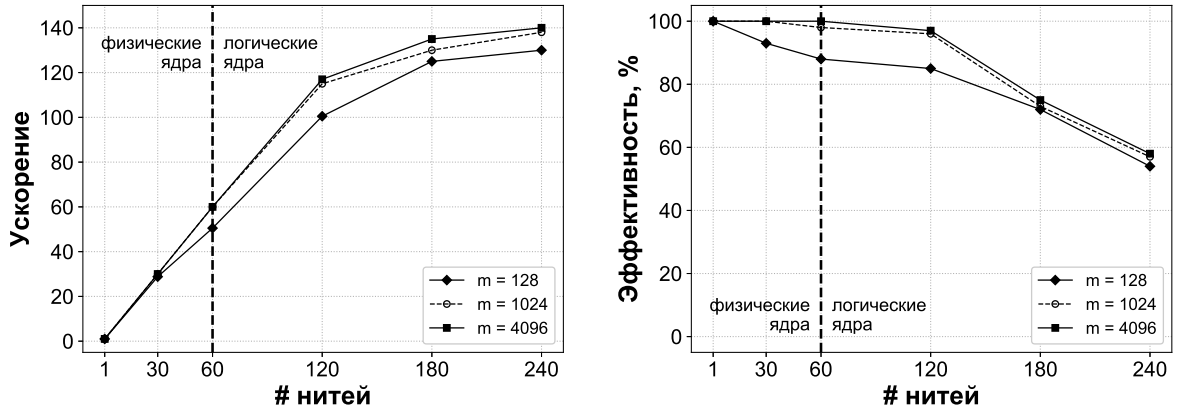
Можно видеть, что разработанный параллельный алгоритм демонстрирует близкое к линейному ускорение и параллельную эффективность от 80 до 100 процентов (в зависимости от длины найденного лейтмотива), если количество нитей, на которых запущен алгоритм, совпадает с количеством физических ядер системы. При увеличении количества нитей, запускаемых на одном физическом ядре системы, ускорение становится сублинейным, равно как наблюдается и падение параллельной эффективности.

Результаты экспериментов алгоритма для графического процессора

представлены на рисунке 2. Можно видеть, что разработанный параллельный алгоритм демонстрирует ускорение, близкое к линейному, и параллельную эффективность от 50 до 100 процентов (в зависимости от длины искомого лейтмотива).




(а) синтетический ряд Random Walk ($n = 4 \cdot 10^5$)



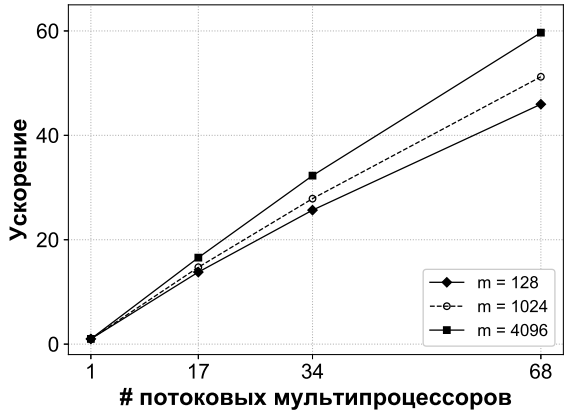

(б) реальный ряд ЭКГ ($n = 4 \cdot 10^5$)

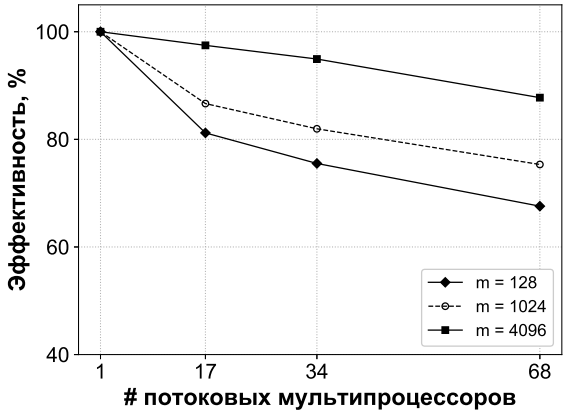
**Рис. 1.** Ускорение и параллельная эффективность алгоритма для многоядерного процессора Intel Xeon Phi

При этом лучшие показатели разработанного параллельного алгоритма для двух различных аппаратных платформ ожидаемо наблюдаются при больших значениях длины искомого лейтмотива, обеспечивающих алгоритму большую вычислительную нагрузку.

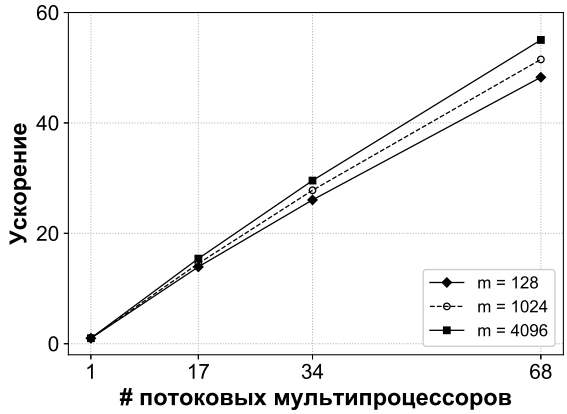
Результаты экспериментов по исследованию производительности алгоритма на различных платформах Intel, представлены в таблицах 2 и 3. Можно видеть, что разработанный параллельный алгоритм поиска лейтмотивов значительно превосходит последовательный алгоритм МК. Также разработанный алгоритм работает быстрее на системах с большим количе-

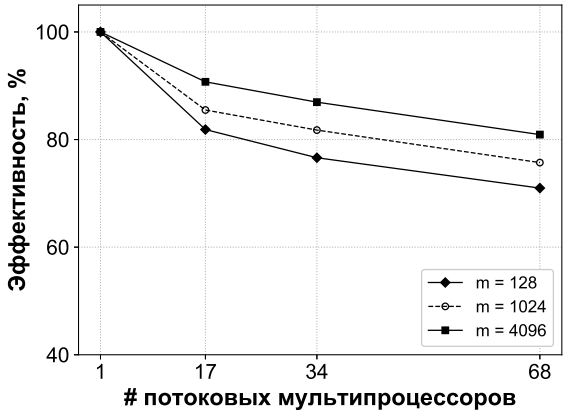
ством ядер. В то же время, разработанный алгоритм работает быстрее на MIC, чем на узле с двумя процессорами Intel Xeon.




(а) синтетический ряд Random Walk ($n = 4 \cdot 10^5$)




(б) реальный ряд ЭКГ ($n = 4 \cdot 10^5$)

**Рис. 2.** Ускорение и параллельная эффективность алгоритма для графического процессора

**Табл. 2.** Быстродействие алгоритма поиска лейтмотивов на синтетическом ряде Random Walk

| **Длина лейтмотива** | **MK** | **Разработанный алгоритм** | | |
|---|---|---|---|---|
| | **Хост (1 ядро)** | **Хост (12 ядер, 24 нити)** | **MIC (61 ядро, 244 нити)** | **GPU (4 352 нити)** |
| 128 | 1 908.3 | 974.7 | 787.1 | 26.8 |
| 1 024 | 20 083.6 | 1 870.2 | 662.7 | 5.1 |

**Табл. 3.** Быстродействие алгоритма поиска лейтмотивов на реальном ряде ЭКГ

| **Длина лейтмотива** | **МК** | **Разработанный алгоритм** | | |
|---|---|---|---|---|
| | **Хост (1 ядро)** | **Хост (12 ядер, 24 нити)** | **MIC (61 ядро, 244 нити)** | **GPU (4 352 нити)** |
| 128 | 2 143.7 | 1 036.3 | 820.4 | 71.3 |
| 1 024 | 13 274.9 | 1 575.8 | 877.4 | 69.3 |

## ЗАКЛЮЧЕНИЕ

Данная выпускная квалификационная работа была посвящена разработке параллельного алгоритма поиска лейтмотивов временного ряда для многоядерных ускорителей.

В ходе выполнения работы были получены следующие основные результаты:

1) проведен обзор последовательных и параллельных алгоритмов поиска лейтмотивов временного ряда;

2) изучены аппаратная архитектура и программная модель многоядерного процессора семейства Intel MIC и графического процессора NVIDIA GPU;

3) спроектирован и реализован параллельный алгоритм поиска лейтмотивов временного ряда для многоядерных ускорителей;

4) проведены вычислительные эксперименты на реальных и синтетических данных, показавшие высокую масштабируемость разработанного алгоритма как на платформе Intel MIC, так и на платформе GPU.

По результатам исследования опубликованы две научные статьи в изданиях, индексируемых в Scopus и РИНЦ:

1. Zymbler M., Kraeva Ya. Discovery of Time Series Motifs on Intel Many-Core Systems. // Lobachevskii Journal of Mathematics. – 2019. – Vol. 40. – No. 12. – P. 2124–2132. – URL: https://doi.org/10.1134/S199508021912014X.

2. Цымблер М.Л., Краева Я.А. Параллельный алгоритм поиска лейтмотивов временного ряда для графического процессора. // Параллельные вычислительные технологии – XIV международная конференция (ПаВТ’2020): Короткие статьи и описания плакатов (Пермь, 31 марта – 2 апреля 2020 г.). – Челябинск: Издательский центр ЮУрГУ, 2020. – С. 298–311.

В рамках выпускной квалификационной работы был сделан доклад на XIV международной научной конференции «Параллельные вычислительные технологии (ПаВТ) 2020» (31 мая – 2 апреля 2020 г., Пермь).

**ЛИТЕРАТУРА**

1. Bacon D.F., Graham S.L., Sharp O.J. Compiler Transformations for High-Performance Computing. // ACM Comput. Surv. – 1994. – Vol. 26. – No. 4. – P. 345–420. – URL: https://doi.org/10.1145/197405.197406.

2. Balasubramanian A., Wang J., Prabhakaran B. Discovering Multidimensional Motifs in Physiological Signals for Personalized Healthcare. // J. Sel. Topics Signal Processing. – 2016. – Vol. 10. – No. 5. – P. 832–841. – URL: https://doi.org/10.1109/JSTSP.2016.2543679.

3. 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. – 2002. – P. 39–48. – URL: https://doi.org/10.1145/565196.565202.

4. Begum N., Keogh E.J. Rare Time Series Motif Discovery from Unbounded Streams. // Proc. VLDB Endow. – 2014. – Vol. 8. – No. 2. – P. 149–160. – URL: https://doi.org/10.14778/2735471.2735476.

5. Brown A.E.X. A dictionary of behavioral motifs reveals clusters of genes affecting Caenorhabditis elegans locomotion. / A.E.X. Brown, E.I. Yemini, L.J. Grundy, T. Jucikas, W.R. Schafer. // Proceedings of the National Academy of Sciences. – 2013. – Vol. 110. – No. 2. – P. 791–796. – URL: https://doi.org/10.1073/pnas.1211447110.

6. Buhler J., Tompa M. Finding Motifs Using Random Projections. // J. Comput. Biol. – 2002. – Vol. 9. – No. 2. – P. 225–242. – URL: https://doi.org/10.1089/10665270252935430.

7. Castro N., Azevedo P.J. Multiresolution Motif Discovery in Time Series. // Proceedings of the SIAM International Conference on Data Mining, SDM 2010, April 29 - May 1, 2010, Columbus, Ohio, USA. – 2010. – P. 665–676. – URL: https://doi.org/10.1137/1.9781611972801.73.

8. 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. – IEEE, 2011. – P. 285–291. – URL: https://doi.org/10.1109/DEST.2011.5936640.

9. Cheng J., Grossman M., McKercher T. Professional CUDA C Programming. 1 ed. – Wrox, 2014. – P. 528.

10. Chiu B.Y., Keogh E.J., Lonardi S. Probabilistic discovery of time series motifs. // Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, August 24-27, 2003. – 2003. – P. 493–498. – URL: https://doi.org/10.1145/956750.956808.

11. Duran A., Klemm M. The Intel® Many Integrated Core Architecture. // 2012 International Conference on High Performance Computing & Simulation, HPCS 2012, Madrid, Spain, July 2-6, 2012. – 2012. – P. 365–366. – URL: https://doi.org/10.1109/HPCSim.2012.6266938.

12. Fang J., Varbanescu A.L., Sips H.J. Sesame: A User-Transparent Optimizing Framework for Many-Core Processors. // Proceedings of the 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing, CCGrid 2013, Delft, Netherlands, May 13-16, 2013. – IEEE, 2013. – P. 70–73. – URL: https://doi.org/10.1109/CCGrid.2013.79.

13. Farber R. Parallel Programming with OpenACC. 1 ed. – Morgan Kaufmann, 2016. – P. 326.

14. Fernandez I. Accelerating time series motif discovery in the Intel Xeon Phi KNL processor. / I. Fernandez, A. Villegas, E. Gutiérrez, O.G. Plata. // The Journal of Supercomputing. – 2019. – Vol. 75. – No. 11. – P. 7053–7075. – URL: https://doi.org/10.1007/s11227-019-02923-5.

15. 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. – 2001. – P. 27–37.

16. Goldberger A.L. PhysioBank, PhysioToolkit, and PhysioNet. / A.L. Goldberger, L.A.N. Amaral, L. Glass, J.M. Hausdorff, P.C. Ivanov, et al. // Circulation. – 2000. – Vol. 101. – No. 23. – P. e215–e220. – URL: https://doi.org/10.1161/01.CIR.101.23.e215.

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

18. Heras D.B. Principal Component Analysis on Vector Computers. / D.B. Heras, J.C. Cabaleiro, V.B. Pérez, P. Costas, F.F. Rivera. // Vector and Parallel Processing - VECPAR'96, Second International Conference, Porto, Portugal, September 25-27, Selected Papers. – 1996. – P. 416–428. – URL: https://doi.org/10.1007/3-540-62828-2_133.

19. Jeffers J., Reinders J. Intel Xeon Phi Coprocessor High Performance Programming. 1st ed. – San Francisco, CA, USA : Morgan Kaufmann Publishers Inc., 2013. – URL: https://doi.org/10.5555/2523262.

20. Jeffers J., Reinders J., Sodani A. Intel Xeon Phi Processor High Performance Programming: Knights Landing Edition. 2nd ed. – San Francisco, CA, USA : Morgan Kaufmann Publishers Inc., 2016. – URL: https://doi.org/10.5555/3050856.

21. 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. – URL: https://doi.org/10.1007/s00778-008-0111-4.

22. Kier N.V., Anh D.T. An Effective Implementation of Motif-Based Time Series Classification. // 2019 IEEE-RIVF International Conference on Computing and Communication Technologies, RIVF 2019, Danang, Vietnam, March 20-22, 2019. – 2019. – P. 1–6. – URL: https://doi.org/10.1109/RIVF.2019.8713620.

23. Kostenetskiy P., Semenikhina P. SUSU Supercomputer Resources for Industry and fundamental Science. // Proceedings of 2018 Global Smart Industry Conference, GloSIC, Chelyabinsk, Russia, November 13-15, 2018. – 2018. – P. 1–7. – URL: https://doi.org/10.1109/GloSIC.2018.8570068.

24. Li Y., Lin J., Oates T. Visualizing Variable-Length Time Series Motifs. // Proceedings of the Twelfth SIAM International Conference on Data Mining, Anaheim, California, USA, April 26-28, 2012. – 2012. – P. 895–906. – URL: https://doi.org/10.1137/1.9781611972825.77.

25. Li Y. Quick-motif: An efficient and scalable framework for exact motif discovery. / Y. Li, L.H. U, M.L. Yiu, Z. Gong. // 31st IEEE International Conference on Data Engineering, ICDE 2015, Seoul, South Korea, April

13-17, 2015. – 2015. – P. 579–590. – URL: https://doi.org/10.1109/ICDE.2015.7113316.

26. Lin J. Finding Motifs in Time Series. / J. Lin, E. Keogh, S. Lonardi, P. Patel. // Proceedings of the 2nd Workshop on Temporal Data Mining at the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Edmonton, Alberta, Canada, July 23-26, 2002. – 2002.

27. Lin J. A symbolic representation of time series, with implications for streaming algorithms. / J. Lin, E.J. Keogh, S. Lonardi, B.Y. Chiu. // Proceedings of the 8th ACM SIGMOD workshop on Research issues in data mining and knowledge discovery, DMKD 2003, San Diego, California, USA, June 13, 2003. – 2003. – P. 2–11. – URL: https://doi.org/10.1145/882082.882086.

28. Lin J. Experiencing SAX: a novel symbolic representation of time series. / J. Lin, E.J. Keogh, L. Wei, S. Lonardi. // Data Min. Knowl. Discov. – 2007. – Vol. 15. – No. 2. – P. 107–144. – URL: https://doi.org/10.1007/s10618-007-0064-z.

29. 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. – URL: https://doi.org/10.1007/s100510050929.

30. Mattson T. S08 - Introduction to OpenMP. // Proceedings of the 2006 ACM/IEEE Conference on Supercomputing, SC '06, Tampa, FL, USA, November 11-17, 2006. – 2006. – P. 209. – URL: https://doi.org/10.1145/1188455.1188673.

31. 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. – URL: https://doi.org/10.1007/s10618-010-0193-7.

32. Meng J. Mining Motifs from Human Motion. / J. Meng, J. Yuan, M. Hans, Y. Wu. // Eurographics 2008 - Short Papers, Crete, Greece, April 14-18, 2008. – 2008. – P. 71–74. – URL: https://doi.org/10.2312/egs.20081024.

33. Minnen D. Discovering Multivariate Motifs using Subsequence Density Estimation and Greedy Mixture Learning. / D. Minnen, C.L.I. Jr., I.A. Essa, T. Starner. // Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence, Vancouver, British Columbia, Canada, July 22-26, 2007. – 2007. – P. 615–620.

34. Mueen A., Chavoshi N. Enumeration of time series motifs of all lengths. // Knowl. Inf. Syst. – 2015. – Vol. 45. – No. 1. – P. 105–132. – URL: https://doi.org/10.1007/s10115-014-0793-4.

35. Mueen A., Keogh E.J., Shamlo N.B. Finding Time Series Motifs in Disk-Resident Data. // ICDM 2009, The Ninth IEEE International Conference on Data Mining, Miami, Florida, USA, December 6-9, 2009. – 2009. – P. 367–376. – URL: https://doi.org/10.1109/ICDM.2009.15.

36. Mueen A., Keogh E.J., Young N.E. Logical-shapelets: an expressive primitive for time series classification. // Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Diego, CA, USA, August 21-24, 2011. – 2011. – P. 1154–1162. – URL: https://doi.org/10.1145/2020408.2020587.

37. 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. – 2009. – P. 473–484. – URL: https://doi.org/10.1137/1.9781611972795.41.

38. Müller M. Analysis and Retrieval Techniques for Motion and Music Data. // Eurographics 2009 - Tutorials, Munich, Germany, March 30 - April 3, 2009. – 2009. – P. 249–255. – URL: https://doi.org/10.2312/egt.20091072.

39. Munshi A. OpenCL Programming Guide. 1 ed. / A. Munshi, B.R. Gaster, T.G. Mattson, J. Fung, D. Ginsburg. – Addison-Wesley, 2011. – P. 646.

40. Narang A., Bhattacherjee S. Parallel Exact Time Series Motif Discovery. // Euro-Par 2010 - Parallel Processing, 16th International Euro-Par Conference, Ischia, Italy, August 31 - September 3, 2010, Proceedings, Part II. – 2010. – P. 304–315. – URL: https://doi.org/10.1007/978-3-642-15291-7_28.

41. Nevill-Manning C.G., Witten I.H. Identifying Hierarchical Structure in Sequences: A Linear-Time Algorithm. // J. Artif. Int. Res. – 1997. – Vol. 7. – No. 1. – P. 67–82. – URL: https://doi.org/10.5555/1622776.1622780.

42. Owens J. GPU architecture overview. // International Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 2007, San Diego, California, USA, August 5-9, 2007, Courses. – 2007. – P. 2. – URL: https://doi.org/10.1145/1281500.1281643.

43. Padua D.A. POSIX Threads (Pthreads). // Encyclopedia of Parallel Computing. – 2011. – P. 1592–1593. – URL: https://doi.org/10.1007/978-0-387-09766-4_447.

44. Patel P. Mining Motifs in Massive Time Series Databases. / P. Patel, E.J. Keogh, J. Lin, S. Lonardi. // Proceedings of the 2002 IEEE International Conference on Data Mining, ICDM 2002, Maebashi City, Japan, December 9-12, 2002. – 2002. – P. 370–377. – URL: https://doi.org/10.1109/ICDM.2002.1183925.

45. Pearson K. The Problem of the Random Walk. // Nature. – 1905. – Vol. 72. – No. 294. – URL: https://doi.org/10.1038/072342a0.

46. Phu L., Anh D.T. Motif-Based Method for Initialization the K-Means Clustering for Time Series Data. // AI 2011: Advances in Artificial Intelligence - 24th Australasian Joint Conference, Perth, Australia, December 5-8, 2011. Proceedings. – 2011. – P. 11–20. – URL: https://doi.org/10.1007/978-3-642-25832-9_2.

47. 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. – URL: https://doi.org/10.1007/s10994-008-5093-3.

48. Rissanen J. Stochastic Complexity in Statistical Inquiry. – World Scientific, 1998. – Vol. 15 of World Scientific Series in Computer Science. – URL: https://doi.org/10.1142/0822.

49. Senin P. GrammarViz 2.0: A Tool for Grammar-Based Pattern Discovery in Time Series. / P. Senin, J. Lin, X. Wang, T. Oates, S. Gandhi, et al. // Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2014, Nancy, France, September 15-19, 2014.

Proceedings, Part III. – 2014. – P. 468–472. – URL: https://doi.org/10.1007/978-3-662-44845-8_37.

50. Shieh J., Keogh E.J. *i*SAX: indexing and mining terabyte sized time series. // Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Las Vegas, Nevada, USA, August 24-27, 2008. – 2008. – P. 623–631. – URL: https://doi.org/10.1145/1401890.1401966.

51. Shokoohi-Yekta M. Discovery of Meaningful Rules in Time Series. / M. Shokoohi-Yekta, Y. Chen, B.J.L. Campana, B. Hu, J. Zakaria, et al. // Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Sydney, NSW, Australia, August 10-13, 2015. – 2015. – P. 1085–1094. – URL: https://doi.org/10.1145/2783258.2783306.

52. Tanaka Y., Iwamoto K., Uehara K. Discovery of Time-Series Motif from Multi-Dimensional Data Based on MDL Principle. // Mach. Learn. – 2005. – Vol. 58. – No. 2-3. – P. 269–300. – URL: https://doi.org/10.1007/s10994-005-5829-2.

53. Truong C.D., Tin H.N., Anh D.T. Combining motif information and neural network for time series prediction. // IJBIDM. – 2012. – Vol. 7. – No. 4. – P. 318–339. – URL: https://doi.org/10.1504/IJBIDM.2012.051734.

54. Voevodin V.V., Voevodin Vl.V. Parallel computing. – BHV-St.Petersburg, 2002. – P. 608.

55. Wang L., Chng E., Li H. A tree-construction search approach for multivariate time series motifs discovery. // Pattern Recognit. Lett. – 2010. – Vol. 31. – No. 9. – P. 869–875. – URL: https://doi.org/10.1016/j.patrec.2010.01.005.

56. Wilson D.R., Martinez T.R. Reduction Techniques for Instance-Based Learning Algorithms. // Mach. Learn. – 2000. – Vol. 38. – No. 3. – P. 257–286. – URL: https://doi.org/10.1023/A:1007626913721.

57. 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. – 2016. – P. 1317–1322. – URL: https://doi.org/10.1109/ICDM.2016.0179.

58. Yeh C.M. Time series joins, motifs, discords and shapelets: a unifying view that exploits the matrix profile. / C.M. Yeh, Y. Zhu, L. Ulanova, N. Begum, Y. Ding, et al. // Data Min. Knowl. Discov. – 2018. – Vol. 32. – No. 1. – P. 83–123. – URL: https://doi.org/10.1007/s10618-017-0519-9.

59. Yin M.S., Tangsripairoj S., Pupacdi B. Variable Length Motif-Based Time Series Classification. // Recent Advances in Information and Communication Technology - Proceedings of the 10th International Conference on Computing and Information Technology, IC2IT 2014, Angsana Laguna, Phuket, Thailand, May 8-9, 2014. – 2014. – P. 73–82. – URL: https://doi.org/10.1007/978-3-319-06538-0_8.

60. Yoon C.E. Earthquake detection through computationally efficient similarity search. / C.E. Yoon, O. O'Reilly, K.J. Bergen, G.C. Beroza. // Science Advances. – 2015. – Vol. 1. – No. 11. – URL: https://doi.org/10.1126/sciadv.1501057.

61. Zhu Y. Matrix Profile XI: SCRIMP++: Time Series Motif Discovery at Interactive Speeds. / Y. Zhu, C.M. Yeh, Z. Zimmerman, K. Kamgar, E.J. Keogh. // IEEE International Conference on Data Mining, ICDM 2018, Singapore, November 17-20, 2018. – 2018. – P. 837–846. – URL: https://doi.org/10.1109/ICDM.2018.00099.

62. Zhu Y. Matrix Profile II: Exploiting a Novel Algorithm and GPUs to Break the One Hundred Million Barrier for Time Series Motifs and Joins. / Y. Zhu, Z. Zimmerman, N.S. Senobari, C.M. Yeh, G.J. Funning, et al. // IEEE 16th International Conference on Data Mining, ICDM 2016, Barcelona, Spain, December 12-15, 2016. – 2016. – P. 739–748. – URL: https://doi.org/10.1109/ICDM.2016.0085.

63. Спецификации видеокарты GeForce RTX 2080 Ti [Электронный ресурс]. URL: https://www.nvidia.com/ru-ru/geforce/graphics-cards/rtx-2080-ti/, (дата обращения: 10.12.2019).