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

РАБОТА ПРОВЕРЕНА

Рецензент
Ведущий инженер-программист
компании “NAUMEN”, к.т.н.

_____ А.Е. Попов

“____”_____ 2019 г.

ДОПУСТИТЬ К ЗАЩИТЕ

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

_____ Л.Б. Соколинский

“____”_____ 2019 г.

РАЗРАБОТКА ПАРАЛЛЕЛЬНОГО АЛГОРИТМА ПОИСКА ДИССОНАНСОВ ВРЕМЕННОГО РЯДА ДЛЯ МНОГОЯДЕРНОГО ПРОЦЕССОРА INTEL XEON PHI

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
ЮУрГУ – 02.04.02.2019.115-011.ВКР

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

_____ М.Л. Цымблер

Автор работы,
студент группы КЭ-220

_____ А.В. Поляков

Ученый секретарь
(нормоконтролер)

_____ О.Н. Иванова

“____”_____ 2019 г.

Челябинск–2019

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное 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

09.02.2019

ЗАДАНИЕ
на выполнение выпускной квалификационной работы магистра
студенту группы КЭ-220
Полякову Андрею Владимировичу,
обучающемуся по направлению
02.04.02 «Фундаментальная информатика и информационные технологии»
(магистерская программа «Технологии разработки высоконагруженных систе»)

1. **Тема работы** (утверждена приказом ректора от 25.04.2019 № 899)
   Разработка параллельного алгоритма поиска диссонансов временного ряда для многоядерного процессора Intel Xeon Phi
2. **Срок сдачи студентом законченной работы:** 05.06.2019.
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. Keogh E.J., et al. Finding the most unusual time series subsequence: algorithms and applications. Knowl. Inf. Syst. 2007. vol. 11, no. 1. pp. 1–27.
   - 3.4. Keogh, E., Lin, J., Fu, A., HOT SAX: Efficiently finding the most unusual time series subsequence, In Proc. ICDM (2005)
   - 3.5. Huang, T., Zhu, Y., Mao, Y., Li, X., Liu, M., Wu, Y., ... & Dobbie, G. (2016). Parallel Discord Discovery. In Advances in Knowledge Discovery and Data Mining (pp. 233-244). Springer International Publishing.
4. **Перечень подлежащих разработке вопросов**
   - 4.1. Провести обзор последовательных и параллельных алгоритмов поиска диссонансов временных рядов.
   - 4.2. Изучить аппаратную архитектуру и программную модель процессора Intel Xeon Phi (Knights Landing).
   - 4.3. Спроектировать и реализовать параллельный алгоритм поиска диссонансов временного ряда для многоядерных процессоров Intel Xeon Phi.
   - 4.4. Провести вычислительные эксперименты по анализу эффективности разработанного алгоритма.

5. **Дата выдачи задания:** 09.02.2019.

Научный руководитель

к.ф.-м.н., доцент М.Л. Цымблер

Задание принял к исполнению А.В. Поляков

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Актуальность темы

В интеллектуальном анализе данных *(Data Mining)* большой интерес вызывают временные ряды. Данные, представленные в виде временных рядов, широко распространены во многих предметных областях: в науке (температура, влажность воздуха), экономике (ежедневные цены на акций, курсы валют), медицине (электрокардиограммы), биологии (ДНК), астрономии.

Под *временным рядом (time series)* понимают последовательность хронологически упорядоченных вещественных значений. Наличие у каждого элемента ряда временно́й метки, неотделимой от значения, существенным образом влияет на постановку и решение задач интеллектуального анализа временных рядов [6, 7].

Важной задачей интеллектуального анализа временных рядов является *поиск диссонансов (discord)* временного ряда. Диссонанс является уточнением понятия аномальной подпоследовательности временного ряда. Задача поиска диссонансов встречается в широком спектре предметных областей, связанных с временными рядами: медицина [10], экономика [28], моделирование климата [26] и др. Понятие диссонанса предложено Кеогом (Keogh) в 2005 г. в работе [10] и определяется как подпоследовательность ряда, которая имеет максимальное расстояние до ближайшего соседа. Ближайшим соседом подпоследовательности является та подпоследовательность ряда, которая не пересекается с данной и имеет минимальное расстояние до нее.

Как отмечается в работе Е. Кеога и др. [11], диссонансы временного ряда для интеллектуального анализа данных особенно привлекательны как детекторы аномалий, т.к. требуют только один интуитивный параметр (длина подпоследовательности), в отличие от большинства других алгоритмов поиска аномалий, требующих много параметров (от 3 до 7 неинтуитивных параметров).

Процесс выявления аномалий в важнейших системах настоятельно рекомендуется автоматизировать, чтобы в процессе мониторинга больших систем была возможность автоматически выполнять действия, корректи-

рующие состояние системы при выявлении аномалий, предотвращая тем самым, например, аварийные ситуации в системе. Также поскольку реальные данные могут быть огромного объема, необходимо задействовать все ресурсы вычислительной системы, на которой они обрабатываются (параллельное выполнение обработки, использование общей и дисковой памяти и др.). Актуальными являются исследования, посвященные использованию параллельных вычислений для решения данной задачи на кластерных системах, многоядерных процессорах и GPU. Для параллельной реализации алгоритма поиска диссонансов ряда перспективным является использование сопроцессоров Intel Xeon Phi.

Нахождение диссонансов временного ряда находят свое применение в следующих областях:

1) медицина (электрокардиограммы, и др.) – отслеживание изменений состояния пациента и поиск патологий;

2) дата-центры и компьютеры – мониторинг состояния памяти, загрузки ресурсов ВС, отслеживание сбоев;

3) отслеживание изменений температуры и климата, погодных аномалий.

Для нахождения диссонанса необходимо находить евклидовы расстояния между подпоследовательностями временного ряда. Нахождение расстояний между подпоследовательностями независимы друг от друга, поэтому есть необходимость в распараллеливании. Поэтому объектом исследования данной работы является процессор Intel Xeon Phi.

Intel Xeon Phi основан на архитектуре Intel Many Integrated Core (MIC), которая выполняет более одного триллиона операций в секунду (TFLOPS) с плавающей запятой. Процессор Intel Xeon Phi включает до 61 процессорных ядер, основанных на архитектуре x86 и соединенных высокопроизводительной встроенной кольцевой шиной. Каждое ядро содержит 512-битный блок векторных вычислений (vector processor unit, VPU), так что может обрабатывать 16 32-битных целых чисел на каждом такте. Также Intel Xeon Phi поддерживает параллелизм уровня потоков для ускорения сложных вычислительных задач с использованием таких технологий параллельного программирования, как OpenMP, MPI.

Существует два поколения Intel Xeon Phi. Второе поколение Phi, Knights Landing (KNL) [18], отличается от первого поколения Phi, Knights Corner (KNC) [5], тем, что может выступать в качестве автономного процессора без центрального процессора.

В данной работе рассматривается случай, когда все данные размещаются в оперативой памяти Intel Xeon Phi. Данный случай является практически значимым во многих предметных областях, например, в медицине, металлургии и др.

Цель и задачи исследования

Целью данной выпускной квалификационной работы является разработка параллельного алгоритма поиска диссонансов временного ряда для многоядерных процессоров Intel Xeon Phi (Knights Landing).

Для достижения поставленной цели необходимо было решить следующие *задачи*:

1) провести обзор последовательных и параллельных алгоритмов поиска диссонансов временных рядов;

2) изучить аппаратную архитектуру и программную модель процессора Intel Xeon Phi (Knights Landing);

3) спроектировать и реализовать параллельный алгоритм поиска диссонансов временного ряда для многоядерных процессоров Intel Xeon Phi (Knights Landing);

4) провести вычислительные эксперименты по анализу эффективности разработанного алгоритма.

Структура и объем работы

Работа состоит из введения, 4 глав, заключения и списка используемой литературы. Объем работы составляет 43 страницы, объем библиографии – 30 наименований.

Содержание работы

Первая глава, «Обзор работ по тематике исследования», содержит обзоры существующих последовательных и параллельных алгоритмов поиска диссонансов во временном ряде для различных вычислительных систем. В каждом алгоритме выявлены плюсы и минусы.

Вторая глава, «Формальные обозначения и постановка задачи», содержит формальные определения и обозначения, описание задачи поиска диссонанса во временном ряде. Приведены техники ускорения поиска диссонанса, и описан последовательный алгоритм HOTSAX, на основе которого разработан параллельный алгоритм PhiDD.

Третья глава, «Параллельный алгоритм PhiDD», содержит детальное описание разработанного параллельного алгоритма PhiDD для многоядерного процессора Intel Xeon Phi.

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

В заключении приведены основные итоги проделанной работы.

1. ОБЗОР РАБОТ ПО ТЕМАТИКЕ ИССЛЕДОВАНИЯ

В данном разделе рассматриваются последовательные и параллельные реализации поиска диссонансов на различных вычислительных системах.

1.1. Последовательные алгоритмы

Наиболее простым с точки зрения реализации является наивный алгоритм «в лоб» (в статье Кеога [11] данный алгоритм назван «BRUTE-FORCE алгоритм»). Реализация данного алгоритма представлена в алг. 1.

Алгоритм 1. BRUTE-FORCE(in $T,\ n$; out $pos_{bsf}, dist_{bsf}$)

```
1:  for all C_i ∈ S_T^n do
2:      dist_min ← ∞
3:      for all C_j ∈ S_T^n and |i − j| ⩾ n do
4:          dist ← ED(C_i, C_j)
5:          if dist < dist_min then
6:              dist_min ← dist
7:          end if
8:      end for
9:      if dist_min > dist_bsf then
10:         dist_bsf ← dist_min
11:         pos_bsf ← i
12:     end if
13: end for
14: return {pos_bsf, dist_bsf}
```

Алгоритм заключается в последовательном переборе всех возможных подпоследовательностей временного ряда заданной длины n и нахождения для каждой из них расстояния до ближайшей к ней несамопересекающейся подпоследовательности временного ряда длины n. Та из подпоследовательностей, для которой это расстояние максимально и есть диссонанс временного ряда.

Данный алгоритм требует всего один входной параметр - необходимую длину подпоследовательности, прост в реализации и позволяет получить точный результат. Однако он имеет фатальный недостаток для задач

обработки больших данных - временная сложность алгоритма $O(m_2)$, что является неприемлемым при работе с достаточно большими наборами данных.

Время работы алгоритма можно улучшить на основе следующих двух наблюдений [10].

1) Во внутреннем цикле нет необходимости находить ближайшего соседу для текущей подпоследовательности. Как только найдена подпоследовательность, расстояние до которой от исследуемой подпоследовательности меньше, чем $dist_{bsf}$, можно считать, что исследуемая подпоследовательность не является диссонансом, выполнение внутреннего цикла можно прервать и перейти к следующей подпоследовательности во внешнем цикле.

2) Скорость работы зависит от того, в каком порядке внешний цикл предлагает кандидатов на роль диссонанса для проверки во внутреннем цикле, и от того, в каком порядке внутренний цикл проходится по остальным подпоследовательностям, в попытках найти подпоследовательность, которая расположена к текущей ближе, чем $dist_{bsf}$ и досрочно завершить цикл.

HOTSAX-алгоритм был впервые предложен E. Keogh и кратко описывается в [10]. HOTSAX использует две эвристики для аугментации входных данных для получения отсортированных наборов подпоследовательностей, по которым проходятся внешний и внутренний циклы из переборного алгоритма (алг. 1). Эвристика для внешнего цикла применяется один раз, а эвристика для внутреннего цикла учитывает текущую обрабатываемую во внешнем цикле подпоследовательность и генерирует новый порядок обхождения остальных подпоследовательностей на каждой итерации внешнего цикла. Данный алгоритм был взят за основу разработанного в данной работе параллельного алгоритма поиска диссонансов.

Янков и др. [24] представили модификацию алгоритма HOTSAX — двухфазный алгоритм поиска диссонансов, ориентированный на данные, размещенные на диске. На первой фазе производится отбор подпоследовательностей-кандидатов, часть которых отсеивается в ходе второй фазы. Алгоритм требует двух линейных сканирований данных на

диске. Авторы предложили алгоритм, который требует только двух проходов по базе данных. Однако для этого алгоритма необходим диапазон, в котором может быть расположен диссонанс. Кроме того, как и алгоритму поиска диссонансов Кеога и др., данному алгоритму требуется также и длина потенциального диссонанса временного ряда. Алгоритм способен обрабатывать временные ряды, которые не могут быть размещены в оперативной памяти, однако до половины общего времени выполнения занимают дисковые операции чтения и записи данных.

Некоторые решения используют методы аппроксимации, которые не требуют вычисления расстояния для необработанных временных рядов. Программа GrammarViz [17] – инструмент визуализации временного ряда, который позволяет одновременно обнаруживать как частые, так и редкие (аномальные) паттерны. GrammarViz использует trie (древовидную структуру данных, поиск в которой происходит за константное время), чтобы декодировать частоту появления для всех паттернов в их дискретизированной форме.

Подобно тому, как это реализовано в GrammarViz, Чен (Chen) и др. [4] также рассматривают аномалии как наиболее редкие временные ряды. Авторы используют вспомогательный счетчик для вычисления показателя аномальности каждого паттерна. Хотя определение аномалий у Чена и др. аналогично диссонансам, их техника требует больше входных параметров, таких как точность наклона e, количество паттернов аномалий k, минимальный порог и др. Кроме того, аномалии, обсуждаемые в их работе, содержат только две точки. Вей (Wei) и др. [20] предлагает другой метод, который использует растровые изображения временных рядов для измерения их сходства.

1.2. Параллельные алгоритмы

Параллельные алгоритмы поиска диссонансов временного ряда – наиболее перспективное направление исследования алгоритмов поиска диссонансов. На данный момент существует довольно мало работ, в которых исследуются способы распараллелить алгоритм поиска диссонансов. Для распараллеливания задачи обнаружения диссонансов ее необходимо

разбить на независимые подзадачи, для возможности их решения на различных вычислительных узлах. В статье Кеога [11] говорится, что метод ”разделяй и властвуй”может давать некорректные результаты для задачи поиска диссонансов временного ряда. Чтобы избежать этого, необходимо найти возможности для разбиения задачи на подзадачи. Однако диссонансы временного ряда имеют следующую особенность: диссонансы, найденные в частях временного ряда не комбинируются [10], и не являются диссонансами всего временного ряда. Для того, чтобы была возможность разделить данные и вычисления между узлами кластера используется другое определение диссонанса, введенное в работе Янкова и др. [24]: диссонансами называются подпоследовательности, расстояние до ближайшего соседа которых больше некоторой константы r.

Янков и др. [24, 25] предложили способ сделать свой двухфазный алгоритм нахождения диссонансов из работы [24] распределенным, используя технологию Map-Reduce.

Вудбридж (Woodbridge) и др. [21] представили подход для обнаружения диссонансов в данных ЭКГ, основанный на применении параллельной СУБД в составе программно-аппаратного комплекса IBM PureData for Analytics. Параллельная СУБД предполагает горизонтальную фрагментацию таблицы с данными ЭКГ пациентов по узлам кластерной вычислительной системы. На каждом узле выполняется извлечение данных ЭКГ и поиск аномалий. Поиск аномалий реализуется как хранимая процедура на языке pgPL/SQL, которая использует рассмотренный выше алгоритм HOTSAX [10, 11].

Распределенный алгоритм поиска диссонансов *PDD (Parallel Discord Discovery)*, который предложили Хуанг (Huang) и др. в работе [8] использует вычислительный кластер на основе фреймворка Apache Spark и хранит временной ряд в распределенной файловой системе HDFS (Hadoop Distributed File System) как набор фрагментов. Один из узлов кластера объявляется мастером, остальные — рабочими. Первая фаза заключается в поиске каждым узлом локальных оценок диссонанса — расстояний от подпоследовательности до ее ближайшего соседа (наиболее похожей подпоследовательности). По завершении обработки своих

фрагментов рабочие пересылают локальные оценки диссонансов мастеру, который находит соответствующие глобальные оценки диссонансов (как минимум локальных оценок). Далее мастер находит приблизительную оценку глобального диссонанса как максимум глобальных оценок диссонансов и рассылает ее рабочим. На второй фазе алгоритм сканирует все подпоследовательности и итеративно уточняет значение оценки приблизительной оценки глобального диссонанса. При этом пересекающиеся подпоследовательности временного ряда объединяются в один пакет-подпоследовательность для передачи на другие вычислительные узлы кластера. Эксперименты, проведенные на 10 вычислительных узлах, показали, что алгоритм *PDD* в семь раз быстрее, чем последовательный алгоритм *HOTSAX* [10, 11], и в два раза эффективнее использует вычислительные ресурсы, чем алгоритм Янкова и др. [24, 25].

Для нахождения ближайшего соседа для подпоследовательности, она и любые другие неперекрывающиеся подпоследовательности временного ряда передаются в один или более вычислительных узлов. Однако, расход времени на передачу двух подпоследовательностей и на вычисление расстояния между ними имеет один и тот же порядок величины. Это приводит к низкому *коэффициенту связи вычислений (CCR)* и, следовательно, к нерациональному использованию вычислительных ресурсов. Необходимо улучшить CCR. Суть улучшения состоит в том, чтобы устранить повторную передачу данных, которые уже были переданы до этого. Например, если мы задали длину скользящего окна 100 и передали подпоследовательность соответствующей длины, то следующая подпоследовательность будет содержать 99 элементов, которые уже были переданы и всего 1 новый элемент. Если же передать 299 элементов, то мы передадим $299 - 100 + 1 = 200$ подпоследовательностей длины 100. Следовательно, CCR улучшится в $(200/299)/(1/100) = 67$ раз.

Первым шагом поиска *локальных оценок диссонанса (DDE)* является оценка расстояния до ближайшего соседа диссонанса. Эта оценка вместе с ранним отказом от дальнейших вычислений может значительно сократить количество вызовов функции вычисления расстояния. Чем ближе оценка к истине, тем меньше раз вычисляется функция расстояния. Для реали-

зации такого подхода обычно используется некоторый массив индексов, однако, т.к. в нашем случае данные делятся на несколько сегментов, то создание централизованного массива с индексами для распределенных данных неэффективно. Алгоритм аппроксимирует каждую подпоследовательность символьным представлением, обозначаемым как A, затем делит их на группы по сходству и вычисляет частоту каждой группы по количеству представлений A, входящих в нее.

Следующим шагом алгоритм выбирает группу с наименьшим количеством членов, обозначаемую как A_d. Затем пробегается по каждому члену этой группы и находит локальные оценки расстояний до ближайшего соседа. Минимальную среди них называют глобальной для этой подпоследовательности, максимальную среди них всех - общей глобальной. Число подпоследовательностей в рассматриваемой группе A_d влияет на качество работы алгоритма. Это число может быть довольно мало, тогда оценка будет плохой. Опытным путем Хуанг (Huang) и др. [8], выяснили, что всего таких групп схожести должно быть примерно 2–10. Далее алгоритм рассчитывает все расстояния до ближайшего соседа и обновляет позицию и значение расстояния для диссонанса. Это самая трудоемкая часть всего алгоритма, поэтому данные передаются, обычно, батчами (пакетами), также используется схема раннего отказа от дальнейших вычислений.

В ходе работы алгоритма может возникать дисбаланс нагрузки вычислительных узлов, некоторые из них могут закончить вычисление и простаивать. Состояние бездействия ухудшает использование вычислительного ресурса. Его можно уменьшить, выделив дополнительные массивы и используя общую входную очередь. В каждом раунде PDD создает общую очередь, которая содержит пакеты в несколько раз больше вычислительных узлов. Как только вычислительный узел заканчивает обработку массива, ему присваивается следующий массив из общей очереди. Круг завершается, когда обрабатываются все пакеты в общей очереди.

1.3. Методы Machine Learning для поиска аномалий временного ряда

Поиск диссонансов временного ряда с помощью методов машинного обучения имеет некоторые особенности, которые перечислены ниже. Сначала необходимо предобработать данные, привести их к виду, готовому для анализа [20]. Обучение модели и оптимизация на основе нормального или ненормального поведения сильно зависит от задачи и имеющихся данных. Большинство методом машинного обучения требуют большого числа неинтуитивных параметров, а некоторые из них требуют предварительного обучения на тестовых данных.

Задача классификации подразумевает определение категории новых экземпляров данных на основе набора изначально размеченных, учебных данных. При этом может иметь место мультиклассовость и вероятностный подход. В случае задачи обнаружения аномалий можно говорить о бинарной классификации (нормальное или ненормальное поведение данных). Далее перечислены основные методы классификации, требующие предварительного обучения на тестовых данных (обучение «с учителем»).

1) *Classification Tree (дерево решений)* строится по предикатам на признаках: в узлах дерева – предикаты, в листьях – метки классов. Существует несколько способов обучения, самый популярный из них – C4.5. Это не самый точный алгоритм, однако, его точность выше, чем, например, у байесовой сети [2].

2) *Fuzzy logic (нечеткая логика)* – форма модели, выведенная из теории нечетких множеств, строящаяся на неких приблизительных рассуждениях из предметной области, выведенных из обычной логики предикатов. Решающие правила строятся на некоторых статистиках данных [15].

3) *Naive Bayes Network (байесова сеть)* – вероятностная модель графа, использующаяся для восстановления статистических или естественных зависимостей между признаками. Качество такого подхода хуже, чем, например, у решающих деревьев, однако скорость выполнения и обучения выше, поэтому на больших выборках есть смысл использовать именно байесову сеть [19].

4) *Genetic Algorithm (генетические алгоритмы)* – эта модель принадлежит к классу эволюционных алгоритмов, которые находят широкое

применение в различных сферах, ввиду таких свойств, как устойчивость к шуму и точность результатов. Практика показала хорошее качество данного метода и на задачах обнаружения аномалий [15].

5) *Neural Networks (нейронные сети)* – набор полносвязных слоев с различными функциями активации. Это одна из самых мощных моделей машинного обучения, способная восстанавливать самые сложные зависимости в данных [3]. В задачах классификации нейронные сети могут строить сложные нелинейные разделяющие гиперповерхности.

6) *Support Vector Machine* – модель машинного обучения, способная восстанавливать сложные зависимости в данных и хорошо показавшая себя в задаче обнаружения аномалий [15].

Кластеризация – это разделение данных по группам похожих объектов. Каждая группа состоит из объектов, аналогичных друг другу, а в разных группах объекты, обычно, отличаются [2]. Методы кластеризации способны обнаруживать какие-либо зависимости (в том числе аномальные) в данных без какой-либо априорной информации. Далее описаны некоторые методы кластеризации, применяемые для обнаружения аномалий с их кратким описанием.

1) *K-means (k-средних)* – этот метод проводит кластеризацию на k непересекающихся классов, основанную на расстоянии между объектами, определяемыми признаками. Параметр k заранее задается исследователем [23]. В публикации [1] этот метод использовали для разделения временных интервалов с нормальным и аномальным трафиком.

2) *K-medoids* – алгоритм, схожий с предыдущим, но при поиске центра кластеров на каждой итерации ищет их не как среднее по группе, а как *медоиду* (такой элемент, различие которого с остальными минимально). Такой подход к задаче кластеризации более робастный (т.е. устойчивый к случайным воздействиям, шуму) нежели k-средних. Сравнения показывают, что на деле, в задаче обнаружения аномалий он работает лучше [15].

3) *EM Clustering* – алгоритм, являющийся расширением метода k-средних. Вместо метки кластера на основе его среднего значения, каждому элементу присваиваются степени вероятности его принадлежности к каж-

дому кластеру. Подробнее метод описан здесь [23]. Он показывает большую точность, чем два предыдущих метода [15].

4) *Outlier Detection Algorithms (алгоритмы обнаружения выбросов)* – это техника обнаружения шаблона в данных, который не соответствует ожидаемому, нормальному поведению. Выброс можно определить как точку в данных, которая сильно отличается от остальных по какой-либо метрике. Существует несколько различных подходов к решению задачи обнаружения выбросов на основе кластеризации и каждый из них лучше подходит в том или ином случае. Один из подходов использует расстояние между элементами в данных [1]. Он основан на методе k ближайших соседей и использует заранее заданную метрику. Чем дальше элемент от своих соседей, тем выше вероятность того, что это – выброс. Этот подход эффективен в обнаружении атак типа Denial of Service (DoS). Другой подход основан на плотности распределения данных. Метрические методы обнаружения выбросов зависят от общего распределения данных (заданного набора точек метрического пространства). Обычно, это распределение неравномерное и такой подход затруднителен. Основная идея метода, основанного на плотности распределения, состоит в том, чтобы сопоставить каждому элементу степень уверенности в том, что он является выбросом, это называется *Local Outlier Factor (LOF)*. Ввиду разнородности плотности распределения данных, каждый элемент рассматривается лишь в окружении элементов, лежащих в некоторой его окрестности. Применение этих методов к сетевым данным сопряжено с некоторыми сложностями, что описано в [15].

Иногда, вместо одного алгоритма лучше использовать комбинированный подход из разных моделей, чтобы компенсировать их недостатки друг другом. Такие методы называются *гибридными*. Далее перечислены некоторые из этих методов.

1) *Каскад алгоритмов обучения с учителем*: часто, для получения хорошего качества классификации разумно использовать большое количество слабо обученных моделей, а результат рассматривать как голосование всех полученных алгоритмов (взвешенное или обычное). Чаще всего в качестве базовых алгоритмов используют решающие деревья, реже – SVM.

2) *Комбинирование обучения с учителем и без*: предобработку данных проводят при помощи обучения без учителя (например, удаляют заранее неверные значения, дополняют пропуски или убирают слишком похожие элементы), а затем выполняют обучение с учителем. Сочетание алгоритма k-средних и решающих деревьев дает хорошее качество на задаче обнаружения аномалий.

В целом, гибридные подходы показывают себя с хорошей стороны в плане точности (например, SVM + NN), но хуже в плане производительности [13]. Решение того, стоит ли определенный прирост качества затраченных дополнительно ресурсов, зависит от решаемой задачи и предметной области.

2. ФОРМАЛЬНЫЕ ОБОЗНАЧЕНИЯ И ПОСТАНОВКА ЗАДАЧИ

2.1. Формальные обозначения

Введем формальные определения и обозначения, которые используются в алгоритме, выполняющий поиск диссонанса во временном ряде.

Временной ряд (time series) T представляет собой последовательность вещественных значений, каждое из которых ассоциировано с отметкой времени, взятых в хронологическом порядке: $T = (t_1, t_2, \ldots, t_m)$, $t_i \in \mathbb{R}$. Число m обозначается $|T|$ и называется длиной ряда.

Подпоследовательность (subsequence) $T_{i,n}$ временного ряда T представляет собой непрерывное подмножество T из n элементов, начиная с позиции i: $T_{i,n} = (t_i, t_{i+1}, \ldots, t_{i+n-1}), 1 \leqslant n \leqslant m, 1 \leqslant i \leqslant m - n + 1$. Множество всех подпоследовательностей ряда T, имеющих длину n, обозначается как S_T^n.

Евклидово расстояние определяется следующим образом:

$$ED(Q, C) = \sqrt{\sum_{i=1}^{n}(q_i - c_i)^2}. \tag{1}$$

Z-нормализацией временного ряда T называется временной ряд $\hat{T} = (\hat{t}_1, \ldots, \hat{t}_m)$, элементы которого вычисляются следующим образом:

$$\hat{t}_i = \frac{t_i - \mu}{\sigma},\ \mu = \frac{1}{m}\sum_{i=1}^{m} t_i,\ \sigma = \sqrt{\frac{1}{m}\sum_{i=1}^{m} t_i^2 - \mu^2}. \tag{2}$$

Z-нормализация [16] позволяет сравнивать формы рядов, отличных по амплитуде. После нормализации среднее арифметическое временного ряда приблизительно равно 0, а среднеквадратичное отклонение близко к 1.

Дадим формальное определение задачи поиска диссонансов в соответствии с терминологией и обозначениями в работе [11].

Подпоследовательности $T_{i,n}$ и $T_{j,n}$ ряда T называются *непересекающимися (non-self match)*, если $|i - j| \geqslant n$. Подпоследовательность, которая является непересекающейся к данной подпоследовательности C, обозначается как M_C.

Пусть функция $Dist : S_T^n \times S_T^n \to \mathbb{R}$ удовлетворяет аксиомам тождества ($\forall X \in S_T^n \; Dist(X, X) = 0$) и симметрии ($\forall X, Y \in S_T^n \; Dist(X, Y) = Dist(Y, X)$). Тогда подпоследовательность D ряда T является *диссонансом (discord)*, если $\forall C, M_C \in T$ $\min(Dist(D, M_D)) > \min(Dist(C, M_C))$. Другими словами, некая подпоследовательность ряда является диссонансом, если она имеет максимальное расстояние до ближайшей непересекающейся с ней подпоследовательностью.

2.2. Последовательный алгоритм HOTSAX

Метод поиска диссонансов во временном ряде, предложенный в работах [10, 11], предполагает предварительную z-нормализацию исходного ряда по формулам (2) и использование евклидовой метрики (1) в качестве функции расстояния.

Подпоследовательности ряда подвергаются *кусочно-агрегатной аппроксимации (PAA, Piesewise Aggregate Approximation)* [14]. Для подпоследовательности $C = (c_1, c_2, \ldots, c_n)$ кусочно-агрегатным представлением (PAA-представлением) является вектор $\overline{C} = (\overline{c}_1, \ldots, \overline{c}_w)$, где *степень агрегации* $w \leqslant n$ — параметр, а координаты вектора вычисляются следующим образом:

$$\overline{c}_i = \frac{w}{n} \cdot \sum_{j=\frac{n}{w}\cdot(j-1)+1}^{\frac{n}{w}\cdot j} c_j. \tag{3}$$

Далее PAA-представление подвергается кодированию с помощью *символьной агрегатной аппроксимации (SAX, Symbolic Aggregate ApproXimation)* [14]. Символьным представлением (SAX-представлением) подпоследовательности $C = (c_1, c_2, \ldots, c_n)$ является *слово* $\hat{C} = (\hat{c}_1, \hat{c}_2, \ldots, \hat{c}_w)$, получаемое следующим образом. Пусть имеется символьный алфавит $\mathcal{A} = (\alpha_1, \alpha_2, \ldots, \alpha_{|\mathcal{A}|})$, где запись $|\mathcal{A}|$ означает мощность алфавита (α_1 =’a’, α_2 =’b’ и т.д.). Тогда

$$\hat{c}_i = \alpha_i \Leftrightarrow \beta_{j-1} \leqslant \hat{c}_i < \beta_j. \tag{4}$$

В формуле (4) числа β_i представляют собой *точки разделения (breakpoints)* [14], определяемые как упорядоченный список чисел $\mathcal{B} =$

$(\beta_0, \beta_1, \ldots, \beta_{|\mathcal{A}|-1}, \beta_{|\mathcal{A}|})$, где $\beta_0 = -\infty$ и $\beta_{|\mathcal{A}|} = +\infty$, а площадь под кривой нормального распределения $N(0,1)$ между β_i и β_{i+1} равна $\frac{1}{|\mathcal{A}|}$. Точки разделения для различных значений параметра мощности алфавита могут быть получены из статистических таблиц [10, 11].

Последовательная реализация поиска диссонансов, предложенная Кеогом и др. [10, 11], представлена в алг. 2.

Алгоритм 2. HOTSAX(in $T,\ n$; out $pos_{bsf}, dist_{bsf}$)

1: **for all** $C_i \in S_T^n$ **do**
2: $dist_{min} \leftarrow \infty$
3: **for all** $C_j \in S_T^n$ **and** $|i - j| \geqslant n$ **do**
4: $dist \leftarrow \mathrm{ED}(C_i, C_j)$
5: **if** $dist < dist_{bsf}$ **then**
6: **break**
7: **end if**
8: **if** $dist < dist_{min}$ **then**
9: $dist_{min} \leftarrow dist$
10: **end if**
11: **end for**
12: **if** $dist_{min} > dist_{bsf}$ **then**
13: $dist_{bsf} \leftarrow dist_{min}$
14: $pos_{bsf} \leftarrow i$
15: **end if**
16: **end for**
17: **return** $\{pos_{bsf}, dist_{bsf}\}$

3. ПАРАЛЛЕЛЬНЫЙ АЛГОРИТМ PHIDD

3.1. Краткое описание параллельного алгоритма phiDD

Последовательный алгоритм *HOTSAX* [10] по построению не обеспечивает векторизацию и выравнивание данных. Алгоритм обрабатывает все подпоследовательности, одну за другой. Данная схема не позволяет сделать код векторизованным. В соответствии с этим в рамках данной работы был спроектирован новый параллельный алгоритм *phiDD* поиска диссонансов во временном ряде для многоядерного процессора Intel Xeon Phi, использующий иную схему организации вычислений и данных. Распараллеливание выполнено с помощью технологии программирования OpenMP. Алгоритм состоит из двух стадий: подготовка и поиск. На стадии подготовки выполняется построение вспомогательных матричных структур данных, обеспечивающих эффективную векторизацию вычислений на платформе Intel Xeon Phi KNL. На стадии поиска алгоритм находит диссонанс с помощью построенных структур. Проведены вычислительные эксперименты, исследующие производительность и масштабируемость алгоритма на реальных наборах данных. Результаты экспериментов показали хорошую масштабируемость алгоритма *PhiDD* и превосходство в производительности над алгоритмами-аналогами.

Ключевыми условиями достижения максимальной производительности вычислительного приложения на Intel Xeon Phi являются векторизация и обработка выравненных данных в памяти.

Векторизация заключается в одновременном выполнении однотипных операций над несколькими элементами массива. Векторизация упаковывает данные в вектора и заменяет скалярные операции на операции с векторами.

Ускорение доступа к данным и эффективную векторизацию циклов обеспечивает выравнивание данных. Выравнивание данных – это изменение положения переменных в памяти таким образом, чтобы они были выравнены относительно некоторой величины. Для процессора Intel Xeon Phi память должна быть выравнена на границе 64 байт [9].

3.2. Проектирование алгоритма

Предлагаемая компоновка данных в оперативной памяти вычислительного узла системы обеспечивает представление временного ряда и вспомогательных данных алгоритма в виде выровненных в памяти матриц, циклы обработки которых векторизуются компилятором.

Выравнивание данных выполняется следующим образом. Пусть обработка некой подпоследовательности $T_{i,n}$ ряда T осуществляется с использованием векторного регистра, вмещающего w вещественных чисел. Если длина подпоследовательности не кратна w, то подпоследовательность дополняется фиктивными нулевыми элементами. Обозначим количество фиктивных элементов за $pad = w - (n \bmod w)$, тогда *выровненная подпоследовательность* $\tilde{T}_{i,n}$ определяется следующим образом:

$$\tilde{T}_{i,n} = \begin{cases} t_i, t_{i+1}, \ldots, t_{i+n-1}, \underbrace{0, 0, \ldots, 0}_{pad}, & if\ n \bmod w > 0 \\ t_i, t_{i+1}, \ldots, t_{i+n-1}, & otherwise. \end{cases} \tag{5}$$

Для обеспечения векторизации вычислений выровненные подпоследовательности временного ряда сохраняются в виде матрицы. *Матрица подпоследовательностей* $S_T^n \in \mathbb{R}^{N \times (n+pad)}$ определяется следующим образом:

$$S_T^n(i, j) := \tilde{t}_{i+j-1}. \tag{6}$$

Предлагаемый алгоритм использует следующие основные структуры для хранения данных в оперативной памяти.

Матрица подпоследовательностей $S_T^n \in \mathbb{R}^{N \times n}$ определяется в соответствии с 6 и хранит все подпоследовательности исходного временного ряда, выровненные в соответствии с 5.

Матрица $PAA_T^{n,w} \in \mathbb{R}^{N \times w}$ предназначена для хранения *PAA-кодов* подпоследовательностей из S_T^n, полученных в соответствии с формулой (3).

Матрица SAX-кодов, $SAX_T^{n,\mathcal{A}} \in \mathbb{N}^{N \times w}$, хранит символьные подпоследовательности в алфавите $\mathcal{A}$, полученные из PAA-кодов в соответствии с формулой (4).

Индекс потенциальных диссонансов представляет собой упорядоченный по возрастанию массив $Cand \in \mathbb{N}^N$ с номерами тех подпоследовательностей в матрице подпоследовательностей S_T^n, чьи SAX-коды наиболее редко встречаются в матрице $SAX_T^{n,\mathcal{A}}$:

$$\begin{gathered} Cand(i) = k \Leftrightarrow F_{SAX}(k) = \min_{1 \leqslant j \leqslant N} F_{SAX}(j) \wedge \\ \forall i < j \;\; Cand(i) < Cand(j). \end{gathered} \tag{7}$$

Здесь $F_{SAX} \in \mathbb{N}^N$ представляет собой *частотный индекс SAX-кодов* — массив, используемый для хранения частот слов из матрицы SAX-кодов:

$$F_{SAX}(i) = k \Leftrightarrow |\{j \mid SAX_T^{n,\mathcal{A}}(j,\cdot) = SAX_T^{n,\mathcal{A}}(i,\cdot)\}| = k. \tag{8}$$

Индекс потенциальных диссонансов создается на стадии подготовки данных и далее на стадии поиска задает порядок перебора подпоследовательностей, которые могут являться диссонансами.

Словарь — матрица $W_{\mathcal{A}} \in \mathbb{N}^{dict_{size} \times w}$, предназначенная для хранения всех возможных слов длины w, составленных из символов алфавита $\mathcal{A}$. Словарь заполняется в соответствии с известным алгоритмом, описанным у Д. Кнута [12]. Словарь организуется таким образом, чтобы все символы каждого слова (элементы в одной строке матрицы) и все слова (строки матрицы) были упорядочены по возрастанию. Мощность словаря $dict_{size}$ вычисляется как число размещений символов алфавита $\mathcal{A}$ по w символов с повторениями:

$$dict_{size} = \bar{A}_{|\mathcal{A}|}^{w} = |\mathcal{A}|^w. \tag{9}$$

В качестве символов алфавита $\mathcal{A}$ будем рассматривать упорядоченный набор натуральных чисел $1, 2, \ldots, |\mathcal{A}|$. Для доступа к элементам словаря вводится хэш-функция $h : \mathbb{N}^w \to \{1, 2, \ldots, dict_{size}\}$, определяемая следующим образом:

$$h(a_1, a_2, \ldots, a_w) = \sum_{j=1}^{w+1} a_j \cdot w^{w-j-1}. \tag{10}$$

Значения длины слова и мощности алфавита, $w = 4$ и $|\mathcal{A}| = 4$, соответственно, как показывают эксперименты [10, 11], хорошо подходят при поиске диссонансов во временных рядах из различных предметных обла-

стей. В силу этого словарь ($4^4 \times 4 = 256$ элементов) может быть размещен в оперативной памяти.

Словарный индекс предназначен для хранения индексов слов алфавита $\mathcal{A}$ в матрице SAX-кодов и представляет собой матрицу $I_W \in \mathbb{N}^{dict_{size} \times N}$:

$$I_W(i,j) = k \Leftrightarrow W_{\mathcal{A}}(i,\cdot) = SAX_T^{n,\mathcal{A}}(k,\cdot). \tag{11}$$

Словарный индекс создается на стадии подготовки данных и далее на стадии поиска используется для задания порядка перебора тех подпоследовательностей, которые не пересекаются с данной.

Частотный индекс словаря $F_W \in \mathbb{N}^{|\mathcal{A}|^w}$ для каждого слова в словаре хранит частоту появления этого слова в матрице SAX-кодов:

$$F_W(i) = k \Leftrightarrow k = |\{j \mid W_{\mathcal{A}}(i,\cdot) = SAX_T^{n,\mathcal{A}}(j,\cdot)\}|. \tag{12}$$

Матричное представление перечисленных выше структур данных представлено в рис. 1.

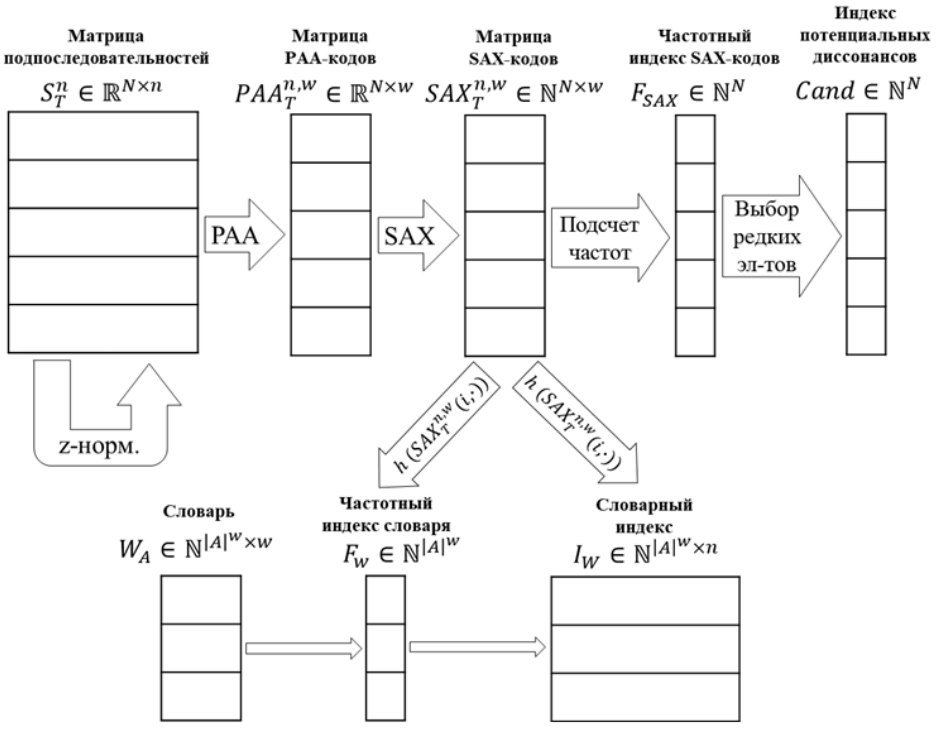


Рис. 1. Матричное представление данных

3.3. Реализация алгоритма

Реализация параллельного поиска диссонансов временного ряда представлена в алг. 3. На стадии подготовки алгоритм выполняет построе-

ние структур данных, рассмотренных выше в разделе 3.2. Далее на стадии поиска алгоритм осуществляет нахождение диссонанса с помощью указанных структур.

Алгоритм 3. PhiDD(in T, n, w; out $\{pos_{bsf}, dist_{bsf}\}$)

▷ Стадия подготовки

1: $S_T^n \leftarrow$ Znormalize(S_T^n)
2: $W_{\mathcal{A}} \leftarrow$ MakeWordMatrix$(\mathcal{A}, w)$
3: $PAA_T^{n,w} \leftarrow$ PAA(S_T^n, w)
4: $SAX_T^{n,\mathcal{A}} \leftarrow$ SAX$(PAA_T^{n,w}, \mathcal{A})$
5: $Cand \leftarrow$ MakeCandidates$(SAX_T^{n,\mathcal{A}})$
6: $I_W \leftarrow$ MakeIndexWord$(SAX_T^{n,\mathcal{A}})$

▷ Стадия поиска

7: $pos_{bsf} \leftarrow 0$; $dist_{bsf} \leftarrow 0$
8: $\{pos_{bsf}, dist_{bsf}\} \leftarrow$ PotentialDiscord$(I_W, Cand, pos_{bsf}, dist_{bsf})$
9: $\{pos_{bsf}, dist_{bsf}\} \leftarrow$ RefineDiscord$(I_W, Cand, pos_{bsf}, dist_{bsf})$
10: **return** $\{pos_{bsf}, dist_{bsf}\}$

На *стадии подготовки* алгоритм действует следующим образом (см. рис. 1). Сначала осуществляется построение матрицы подпоследовательностей исходного временного ряда. Далее выполняется генерация матрицы слов, строками которой будут все возможные слова длины w, составленные из символов алфавита $\mathcal{A}$.

После этого происходит формирование PAA-кода каждой подпоследовательности в матрице подпоследовательностей в соответствии с формулой (3). Соответствующий цикл обработки строк матрицы подпоследовательностей распараллеливается с помощью стандартной директивы компилятора `#pragma omp parallel for` OpenMP, обеспечивающей статическое разбиение итераций цикла между нитями.

Затем выполняется формирование SAX-кода подпоследовательности для каждого PAA-кода, полученного на предыдущем шаге. Параллельная обработка строк матрицы PAA-представлений, так же как и на предыдущем шаге, обеспечивается использованием директивы компилятора `#pragma omp parallel for`.

Далее на основе полученной матрицы SAX-кодов подпоследовательностей осуществляется формирование индекса потенциальных диссонансов $Cand$ (см. формулу 7). Для этого вычисляется частотный индекс F_{SAX} (см. формулу 8) и в $Cand$ добавляются номера строк матрицы SAX-кодов, которые имеют минимальную частоту (встречаются в матрице SAX-кодов наиболее редко). На данном шаге нахождение минимального элемента массива F_{SAX} также распараллеливается с помощью директивы компилятора `#pragma omp parallel for` с использованием параметра `reduction`, который обеспечивает свертку операции поиска минимума.

Следующим шагом алгоритма является построение словарного индекса I_W, выполняемое следующим образом. Осуществляется сканирование матрицы SAX-кодов. Для каждой строки этой матрицы вычисляется хэш-функция, дающая номер соответствующего элемента в матрице слов (см. формулу 10). Далее в строку словарного индекса с номером, совпадающим со значением хэш-функции, записывается номер слова в матрице SAX-кодов. В данном шаге цикл обработки строк матрицы SAX-кодов распараллеливается с помощью стандартной директивы компилятора `#pragma omp parallel for` OpenMP, обеспечивающей статическое разбиение итераций цикла между нитями.

Стадия поиска состоит из двух следующих шагов. На первом шаге (см. алг. 4) выполняется поиск диссонансов среди подпоследовательностей, входящих в индекс потенциальных диссонансов, построенный на предыдущей стадии, в порядке, задаваемом указанным индексом. Результатом данного шага позиция предполагаемого диссонанса в исходном временном ряде и расстояние (степень схожести) предполагаемого диссонанса с его ближайшим соседом. На втором шаге (см. алг. 5) выполняется уточнение найденной ранее позиции предполагаемого диссонанса среди тех подпоследовательностей, которые не входят в в индекс потенциальных диссонансов.

Шаг поиска диссонансов (см. алг. 4 и рис. 2) заключается в следующем. Для каждой подпоследовательности индекса потенциальных диссонансов $Cand$ выполняется нахождение ближайших соседей среди всех подпоследовательностей временного ряда, в порядке, задаваемом этим ин-

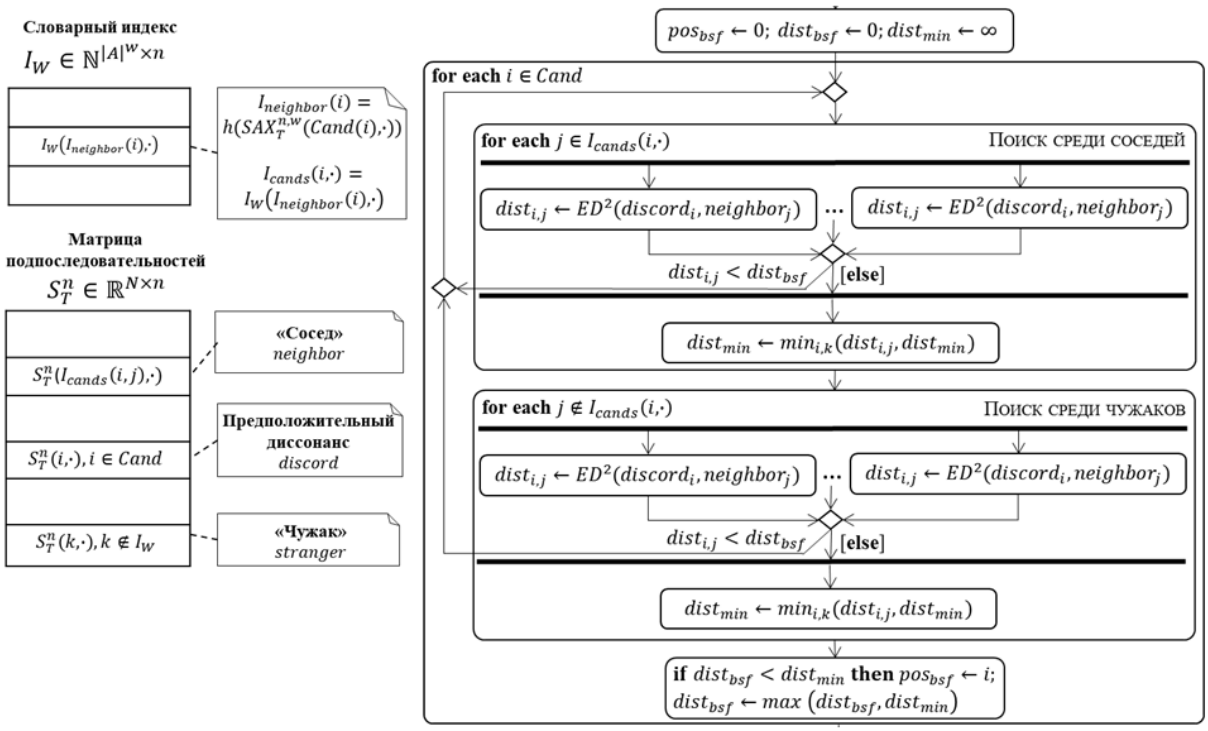


Рис. 2. Стадия поиска потенциального диссонанса

дексом, по формуле (7). Из найденных ближайших соседей выбирается сосед, имеющий наименьшую степень схожести с другими подпоследовательностями (в смысле Евклидовой метрики). При этом в вычислениях используется квадрат Евклидова расстояния (вместо собственно значения расстояния) для ускорения вычислений.

Поиск ближайшего соседа подпоследовательности осуществляется следующим образом. Сначала в качестве соседей рассматриваются подпоследовательности строки словарного индекса I_W, соответствующей данной обрабатываемой подпоследовательности, которые имеют одинаковый с ней SAX-код. Перебор соседей осуществляется в порядке, задаваемом словарным индексом.

Затем рассматриваются подпоследовательности, не вошедшие в словарный индекс, в порядке, задаваемом матрицей подпоследовательностей. При этом расстояние вычисляется только для подпоследовательностей, которые не являются пересекающимися. Если найденное расстояние между подпоследовательностями меньше найденного на предыдущих итерациях цикла расстояния до ближайшего соседа $dist_{bsf}$, то текущая подпоследовательность исключается из рассмотрения. Такое отбрасывание позволяет

Алгоритм 4. PotentialDiscord(in $T,\ n$; out $\{pos_{bsf}, dist_{bsf}\}$)

```
1:  for all C_i ∈ Cand do
2:      dist_min ← ∞
3:      for all C_j ∈ I_W(SAX_T^{n,A}(C_i)) and |i − j| ⩾ n do
4:          dist ← ED^2(C_i, C_j)
5:          if dist < dist_bsf then
6:              break
7:          end if
8:          if dist < dist_min then
9:              dist_min ← dist
10:         end if
11:     end for
12:     #pragma omp parallel for schedule(dynamic)
13:     for all C_j ∉ I_W(SAX_T^{n,A}(C_i)) and |i − j| ⩾ n do
14:         dist ← ED^2(C_i, C_j)
15:         if dist < dist_bsf then
16:             break
17:         end if
18:     end for
19:     if dist_min > dist_bsf then
20:         dist_bsf ← dist_min
21:         pos_bsf ← i
22:     end if
23: end for
24: return {pos_bsf, dist_bsf}
```

сократить объем вычислений и возможно, поскольку найдена подпоследовательность, степень схожести которой с текущей подпоследовательностью больше, чем между двумя другими подпоследовательностями, рассмотренными на предыдущей итерации цикла.

После того, как найден ближайший сосед для данной подпоследовательности, производится сравнение его степени схожести со степенью схожести найденного на предыдущих итерациях самого необычного из бли-

жайших соседей. Полученный самый необычный ближайший сосед является предположительным диссонансом, и алгоритм выдает положение диссонанса во временном ряде.

На втором шаге (см. алг. 5 и рис. 3) выполняется уточнение найденной ранее позиции предполагаемого диссонанса среди тех подпоследовательностей, которые не входят в индекс потенциальных диссонансов. Поиск ближайших соседей каждой из перебираемых подпоследовательностей выполняется аналогично поиску на первом шаге (нахождение предположительного диссонанса, см. алг. 4). Среди всех найденных ближайших соседей осуществляется поиск такого соседа, который имеет наименьшую степень схожести со всеми подпоследовательностями. Результатом данного шага является уточненное положение диссонанса временного ряда и расстояние (степень схожести) от данного диссонанса до его ближайшим соседом.

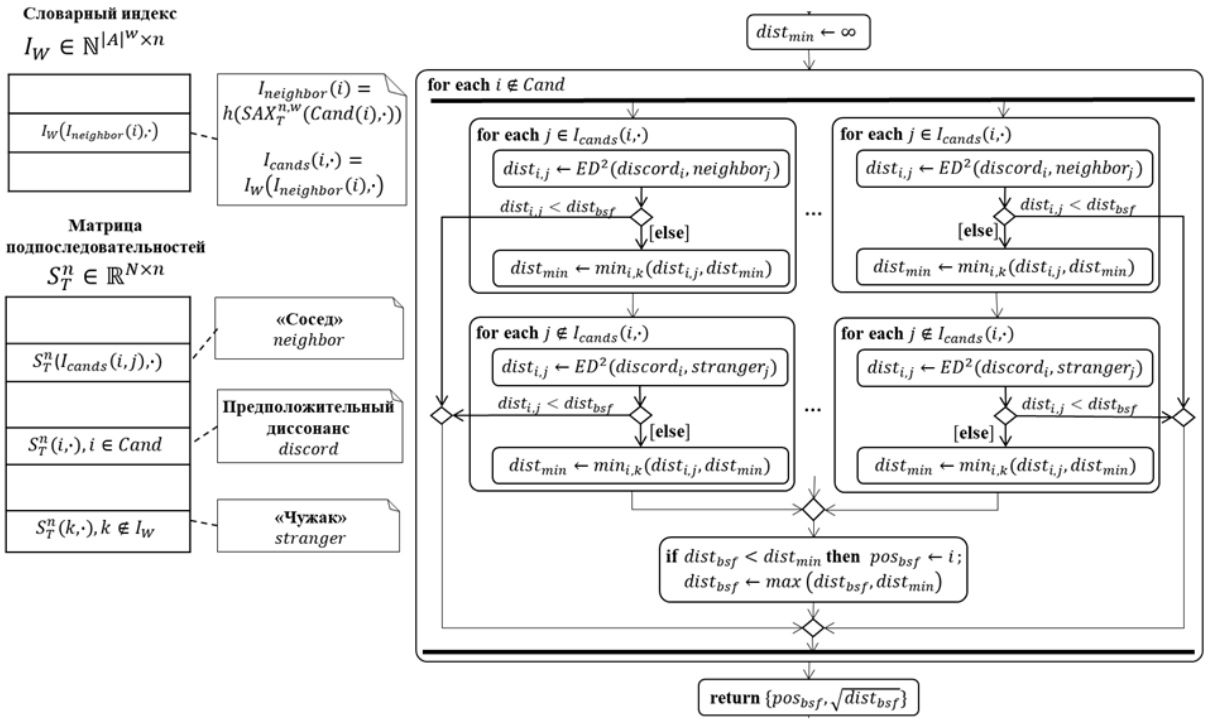


Рис. 3. Уточнение диссонанса

В обоих описанных выше шагах стадии поиска перебор ближайших соседей распараллеливается с помощью стандартной директивы компилятора OpenMP `#pragma omp parallel for`. Поскольку отбрасывание неперспективных подпоследовательностей приводит к неравномерной вы-

Алгоритм 5. RefineDiscord(in $T,\ n$; out $\{pos_{bsf}, dist_{bsf}\}$)

```
#pragma omp parallel for schedule(dynamic)
for all C_i ∉ Cand do
    dist_min ← ∞
    for all C_j ∈ I_W(SAX_T^{n,A}(C_i)) and |i − j| ⩾ n do
        dist ← ED²(C_i, C_j)
        if dist < dist_bsf then
            break
        end if
        if dist < dist_min then
            dist_min ← dist
        end if
    end for
    for all C_j ∉ I_W(SAX_T^{n,A}(C_i)) and |i − j| ⩾ n do
        dist ← ED²(C_i, C_j)
        if dist < dist_bsf then
            break
        end if
        if dist < dist_min then
            dist_min ← dist
        end if
    end for
    if dist_min > dist_bsf then
        dist_bsf ← dist_min
        pos_bsf ← i
    end if
end for
return {pos_bsf, √dist_bsf}
```

числительной загрузке нитей, в указанной директиве используется параметр `schedule (dynamic)`, обеспечивающий динамическое распределение итераций цикла между нитями. Операторы тела цикла, выполняю-

щего вычисление квадратов Евклидовых расстояний, векторизуются компилятором.

Репозиторий с кодом программы *phiDD* расположен по адресу: `https://github.com/AVPGenium/discords_discovery`.

4. ВЫЧИСЛИТЕЛЬНЫЕ ЭКСПЕРИМЕНТЫ

В данном разделе представлены результаты экспериментов по исследованию эффективности разработанного параллельного алгоритма *phiDD* поиска диссонансов временного ряда, описанного в главе 3.

4.1. Цели, аппаратная платформа и наборы данных экспериментов

Для исследования эффективности разработанного параллельного алгоритма *phiDD* поиска диссонансов временного ряда были проведены вычислительные эксперименты. Был взят разработанный параллельный алгоритм, который был запущен для разных наборов входных данных. После чего было измерено время выполнения алгоритма. При подсчете времени выполнения алгоритма не учитывается считывание временного ряда и запроса из файлов и загрузка данных в память, а также вывод результатов выполнения алгоритма. В качестве аппаратной платформы экспериментов использованы вычислительные узлы суперкомпьютеров «Торнадо ЮУрГУ» [29] и Сибирского Суперкомпьютерного Центра ИВМиМГ СО РАН [30], характеристики которых приведены в табл. 1.

Табл. 1. Аппаратная платформа экспериментов

Характеристика	Процессор Intel Xeon		Ускоритель Intel Xeon Phi	
Модель	E5-2630v4	E5-2697v4	SE10X (KNC)	7290 (KNL)
К-во физ. ядер	2×10	2×16	60	72
Гиперпоточность	2×	2×	4×	4×
К-во лог. ядер	40	64	240	288
Частота, ГГц	2.2	2.6	1.1	1.5
Размер VPU, бит	256	256	512	512
Пик. пр-ть, TFLOPS	0.390	0.600	1.076	3.456

Исследование производилось на наборах данных, которые представлены в табл. 2. Указанные наборы использовались в экспериментах по исследованию эффективности работы алгоритма поиска диссонансов, предложенного в работе [8].

В экспериментах исследовались производительность и масштабируемость алгоритма *PhiDD*. Под производительностью понимается время работы алгоритма без учета времени загрузки данных в память и выдачи ре-

Табл. 2. Наборы данных для экспериментов

Набор данных	Вид	$\|T\| = m$	n
SCD-1M	Синтетический	10^6	64, 128, 512, 1024, 2048
SCD-10M	Синтетический	10^7	64, 128, 512, 1024, 2048

зультата. 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 [27].

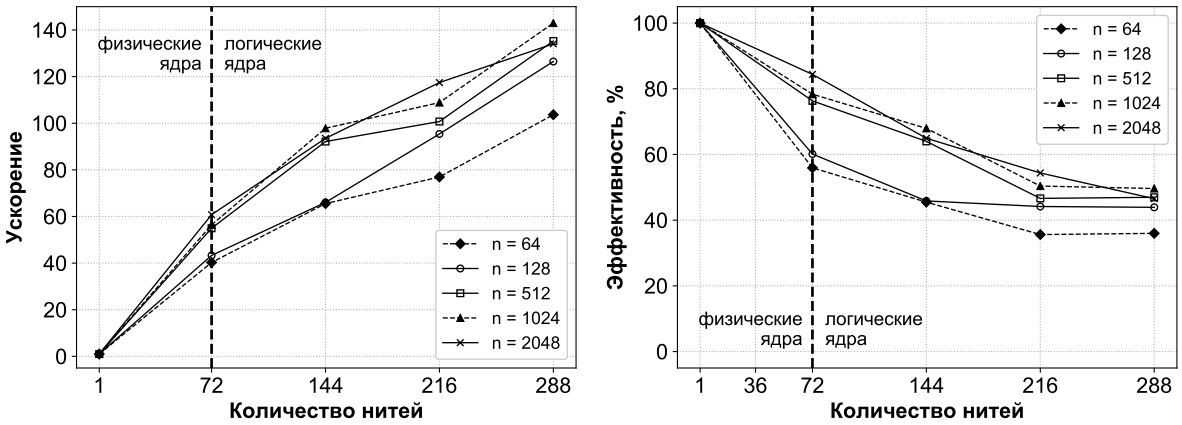
Ускорение и *параллельная эффективность* параллельного алгоритма, запускаемого на k нитях, вычисляются как $s(k) = \frac{t_1}{t_k}$ и $e(k) = \frac{s(k)}{k}$ соответственно, где t_1 и t_k — время работы алгоритма на одной и k нитях соответственно.

В экспериментах рассматривались вышеприведенные показатели в зависимости от изменения параметра n (длина искомого диссонанса). Время работы алгоритма *PhiDD* сравнивалось с временем работы оригинального последовательного алгоритма поиска диссонансов *HOTSAX* [11] и распределенным алгоритмом *PDD* [8, 22].

4.2. Результаты экспериментов

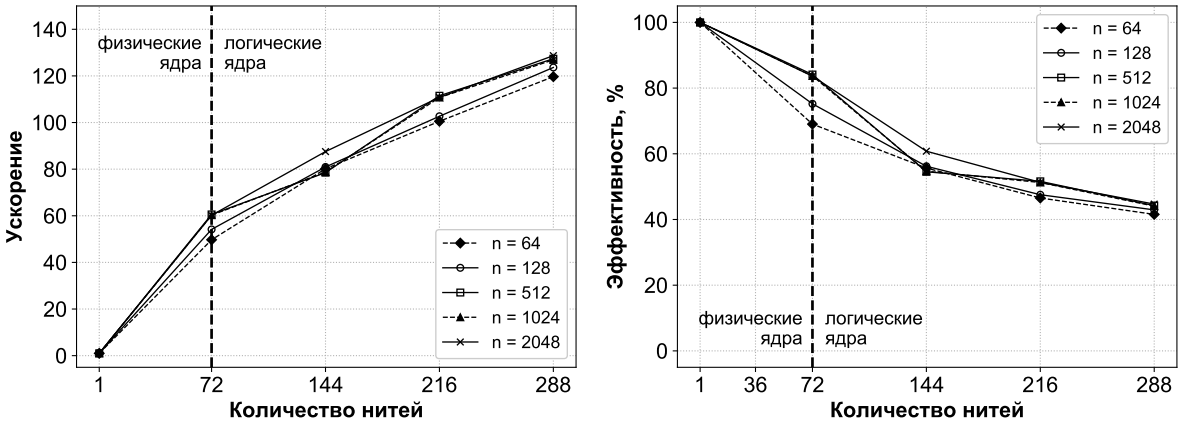
Результаты экспериментов ***по исследованию масштабируемости*** алгоритма на системе Intel Xeon Phi Knights Landing представлены на рис. 4 и 5. Результаты экспериментов показывают, что *PhiDD* демонстрирует ускорение от 40 до 60 и параллельную эффективность от 50 до 90% (в зависимости от длины искомого диссонанса), если количество нитей, на которых запущен алгоритм, совпадает с количеством физических ядер системы Intel Xeon Phi.

При увеличении количества нитей, запускаемых на одном физическом ядре системы, ускорение является сублинейным, равно как наблюдается и падение параллельной эффективности. При этом наилучшие показатели ускорения и параллельной эффективности ожидаемо наблюдаются при бóльших значениях параметров m и n (длина исходного временного



Ускорение Параллельная эффективность

Рис. 4. Масштабируемость *PhiDD* при обработке ряда SCD-1M



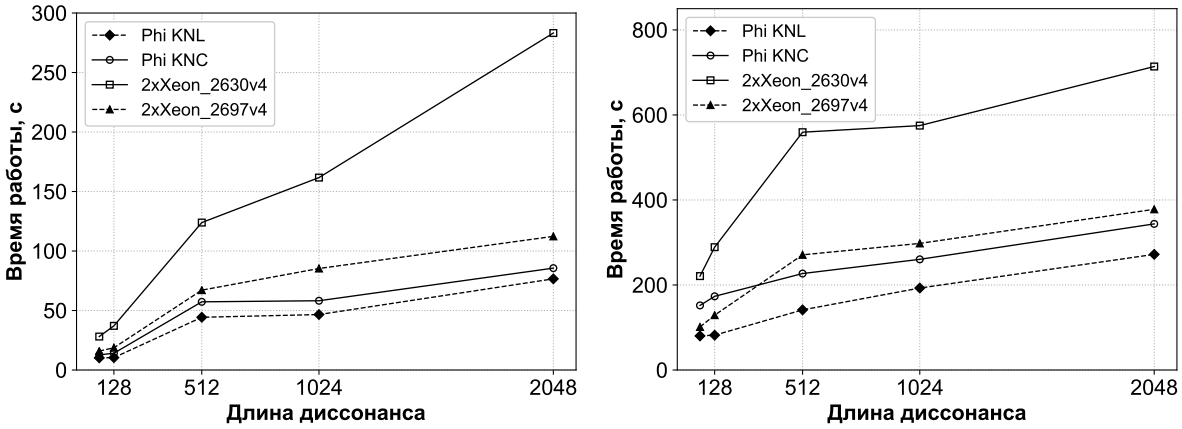
Ускорение Параллельная эффективность

Рис. 5. Масштабируемость *PhiDD* при обработке ряда SCD-10M

ряда и длина искомого диссонанса соответственно), обеспечивающих алгоритму наибольшую вычислительную нагрузку. Например, при обработке ряда длиной 1 млн. 288 нитями наблюдаются ускорение 100 и параллельная эффективность 40% при $n = 64$, тогда как при $n = 2048$ эти показатели равны 140 и 50% соответственно.

Полученные результаты позволяют сделать заключение о хорошей масштабируемости разработанного алгоритма и эффективном использовании им возможностей векторизации вычислений на многоядерной системе Intel Xeon Phi.

Результаты экспериментов ***по исследованию производительности*** алгоритма представлены в рис. 6. Результаты экспериментов показывают, что алгоритм *PhiDD* работает, как правило, быстрее на платформе многопроцессорной системы Intel Xeon Phi, чем на платформе двухпроцессорно-



Набор данных SCD-1M Набор данных SCD-10M

Рис. 6. Производительность алгоритма *PhiDD*

го узла Intel Xeon. Данный факт всегда имеет место для более производительного устройства поколения Knights Landing; менее производительное устройство поколения Knights Corner уступает 64-ядерному двухпроцессорному узлу Intel Xeon при поиске диссонансов длины $n \leqslant 128$ в ряде из 10 млн. точек. В целом на платформе Intel Xeon Phi KNL алгоритм опережает себя на платформе 64-ядерного двухпроцессорного узла Intel Xeon в 1.5–2 раза.

Результаты экспериментов ***по сравнению алгоритма с аналогами*** представлены в табл. 3. Данные о быстродействии алгоритма *PDD* взяты из работы [8]. Результаты быстродействия алгоритма *HOTSAX* получены на платформе рабочей станции с процессором Intel Core i5 2.6 ГГц и использованы для сравнения ускорения, показываемого алгоритмами *PDD* и *PhiDD* относительно *HOTSAX*. Быстродействие *PhiDD* дополнительно показано на 10 ядрах, поскольку в этом случае аппаратная платформа экспериментов имеет примерно равную пиковую производительность с платформой, на которой получены экспериментальные результаты быстродействия алгоритма *PDD*. Результаты экспериментов показывают, что алгоритм *PhiDD* опережает аналоги.

4.3. Анализ пространственной сложности алгоритма

Экспериментальные исследования показывают (см. раздел 4.2), что *PhiDD* демонстрирует хорошую масштабируемость, однако для этого

необходимо, чтобы исходные данные обеспечивали алгоритму надлежащий большой объем вычислений (например, при длине искомого диссонанса $n \geqslant 128$ — см. рис. 4 и 5).

Дополнительно следует указать, что платой за масштабируемость алгоритма также является его повышенные, в отличие от аналогов, требования к оперативной памяти. С учетом предложенных в формулах (5)–(12) вспомогательных структур данных алгоритма можно сформулировать следующее утверждение.

Табл. 3. Сравнение производительности алгоритма *PhiDD* с аналогами

Набор	Время работы алгоритмов на указанных платформах (при $n = 128$), с				Ускорение алгоритмов относительно *HOTSAX*		
	PDD	*PhiDD*		*HOTSAX*	*PDD*	*PhiDD*	
	10 CPU	Intel Xeon Phi 7290		Intel Core i5			
	1.2 ГГц	10 нитей	288 нитей	2.6 ГГц	10 CPU	10 нитей	288 нитей
SCD-1M	36 720	109.2	10.4	147 600	$4\times$	$1.3 \cdot 10^3\times$	$1.4 \cdot 10^4\times$
SCD-10M	399 600	833.3	81.9	604 800[1]	1.5	$7.25 \cdot 10^2\times$	$7.4 \cdot 10^3\times$

Утверждение. Алгоритм *PhiDD* имеет пространственную сложность $O(mn)$, где m — длина временного ряда, n — длина искомого диссонанса.

Доказательство. По построению алгоритм использует следующие вспомогательные структуры данных: матрица выровненных подпоследовательностей $S_T^n \in \mathbb{R}^{N\times(n+pad)}$, матрица PAA-кодов $PAA_T^{n,w} \in \mathbb{R}^{N\times w}$, матрица SAX-кодов $SAX_T^{n,\mathcal{A}} \in \mathbb{N}^{N\times w}$, индекс потенциальных диссонансов $Cand \in \mathbb{N}^N$, частотный индекс SAX-кодов $F_{SAX} \in \mathbb{N}^N$, словарь $W_\mathcal{A} \in \mathbb{N}^{|\mathcal{A}|^w\times w}$, словарный индекс $I_W \in \mathbb{N}^{|\mathcal{A}|^w\times N}$ и частотный индекс словаря $F_W \in \mathbb{N}^{|\mathcal{A}|^w}$. Суммирование размерностей указанных структур данных дает пространственную сложность

$$O(N \cdot (n + pad + 2 \cdot (w + 1) + |\mathcal{A}|^w) + |\mathcal{A}|^w \cdot (w + 1)).$$

В данном выражении количество фиктивных элементов для выравнивания данных pad меньше длины искомого диссонанса n: $pad < n$. В свою очередь, длина диссонанса много меньше длины исходного временного ряда: $n \ll m$. Рекомендуемые для использования на практике значе-

[1] Остановлено после 7 суток безрезультатной работы.

ния длины слова и мощности алфавита, $w = 4$ и $|\mathcal{A}| = 4$ [10, 11]. Таким образом пространственная сложность алгоритма составляет $O(Nn)$, где N, количество подпоследовательностей длины n во временном ряде длины m, есть $N = m - n + 1$, что равносильно $O(mn)$.

ЗАКЛЮЧЕНИЕ

Данная выпускная квалификационная работа была посвящена разработке параллельного алгоритма поиска диссонансов временного ряда для многоядерных процессоров Intel Xeon Phi.

В ходе выполнения работы были получены следующие основные результаты:

1) проведен обзор последовательных и параллельных алгоритмов поиска диссонансов временных рядов на различных вычислительных системах;

2) изучена аппаратная архитектура и программная модель процессора Intel Xeon Phi (Knights Landing);

3) спроектирован и реализован параллельный алгоритм поиска диссонансов временного ряда для многоядерных процессоров Intel Xeon Phi (Knights Landing);

4) проведены вычислительные эксперименты на реальных и синтетических данных, результаты экспериментов показали хорошую масштабируемость алгоритма *PhiDD* и превосходство в производительности над алгоритмами-аналогами.

По результатам исследования имеется принятая к публикации статья в серии Communications in Computer and Information Science издательства Springer (индексируется в Scopus). В ходе выпускной квалификационной работы были сделаны доклады на XIII международной научной конференции «Параллельные вычислительные технологии (ПаВТ) 2019» (2–4 апреля 2019 г., Калининград) и 72 й научной конференции студентов ЮУрГУ (Челябинск, 24 мая 2019 г.).

ЛИТЕРАТУРА

1. Agarwal B., Mittal N. Hybrid Approach for Detection of Anomaly Network Traffic using Data Mining Techniques. // Procedia Technology. – 2012. – 12. – Vol. 6.

2. Chandola V., Banerjee A., Kumar V. Anomaly Detection: A Survey. // ACM Comput. Surv. – 2009. – Vol. 41. – No. 3. – P. 15:1–15:58. – URL: http://doi.acm.org/10.1145/1541880.1541882.

3. Chauhan A., Mishra G., Kumar Ahuja D.G. Survey on Data Mining Techniques in Intrusion Detection. // International Journal of Scientific & Engineering Research Volume. – 2011. – 08. – Vol. 2.

4. Chen X.y., Zhan Y.y. Multi-scale Anomaly Detection Algorithm Based on Infrequent Pattern of Time Series. // J. Comput. Appl. Math. – 2008. – Vol. 214. – No. 1. – P. 227–237. – URL: http://dx.doi.org/10.1016/j.cam.2007.02.027.

5. Chrysos G. Intel® Xeon Phi coprocessor (codename Knights Corner). // 2012 IEEE Hot Chips 24th Symposium (HCS), Cupertino, CA, USA, August 27–29, 2012. – 2012. – P. 1–31.

6. Esling P., Agon C. Time-series Data Mining. // ACM Comput. Surv. – 2012. – Vol. 45. – No. 1. – P. 12:1–12:34. – URL: https://doi.org/10.1145/2379776.2379788.

7. Fu T.C. A review on time series data mining. // Eng. Appl. of AI. – 2011. – Vol. 24. – No. 1. – P. 164–181. – URL: https://doi.org/10.1016/j.engappai.2010.09.007.

8. Huang T. Parallel Discord Discovery. / T. Huang, Y. Zhu, Y. Mao, X. Li, M. Liu, et al. // Advances in Knowledge Discovery and Data Mining – 20th Pacific-Asia Conference, PAKDD 2016, Auckland, New Zealand, April 19–22, 2016, Proceedings, Part II. – 2016. – P. 233–244. – URL: https://doi.org/10.1007/978-3-319-31750-2_19.

9. Jeffers J., Reinders J. Intel Xeon Phi Coprocessor High Performance Programming. – Morgan Kaufmann Publishers Inc., 2013. – URL: https://doi.org/10.1016/C2011-0-06997-1.

10. Keogh E., Lin J., Fu A. HOT SAX: efficiently finding the most unusual time series subsequence. // Proceedings of the 5th IEEE International

Conference on Data Mining, ICDM'05, Houston, Texas, November 27–30, 2005. – 2005. – P. 8. – URL: https://doi.org/10.1109/ICDM.2005.79.

11. Keogh E.J. Finding the most unusual time series subsequence: algorithms and applications. / E.J. Keogh, J. Lin, S. Lee, H.V. Herle. // Knowl. Inf. Syst. – 2007. – Vol. 11. – No. 1. – P. 1–27. – URL: https://doi.org/10.1007/s10115-006-0034-6.

12. Knuth D.E. The Art of Computer Programming, Volume 4, Fascicle 3: Generating All Combinations and Partitions. – Addison-Wesley Professional, 2005.

13. Lee W., Stolfo S.J. Data Mining Approaches for Intrusion Detection. // Proceedings of the 7th Conference on USENIX Security Symposium - Volume 7. – SSYM'98. – Berkeley, CA, USA : USENIX Association, 1998. – P. 6–6. – URL: http://dl.acm.org/citation.cfm?id=1267549.1267555.

14. 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.

15. Padhy N., Pragnyaban Mishra D., Panigrahi R. The Survey of Data Mining Applications And Feature Scope. // International Journal of Computer Science, Engineering and Information Technology (IJCSEIT). – 2012. – 11. – Vol. 2.

16. Rakthanmanon T. Searching and Mining Trillions of Time Series Subsequences under Dynamic Time Warping. / T. Rakthanmanon, B.J.L. Campana, A. Mueen, G.E.A.P.A. Batista, M.B. Westover, et al. // The 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD'12, Beijing, China, August 12–16, 2012. – 2012. – P. 262–270. – URL: https://doi.org/10.1145/2339530.2339576.

17. Senin P. GrammarViz 3.0: Interactive Discovery of Variable-Length Time Series Patterns. / P. Senin, J. Lin, X. Wang, T. Oates, S. Gandhi, et al. //

ACM Trans. Knowl. Discov. Data. – 2018. – Vol. 12. – No. 1. – P. 10:1–10:28. – URL: http://doi.acm.org/10.1145/3051126.

18. Sodani A. Knights Landing (KNL): 2nd Generation Intel® Xeon Phi processor. // 2015 IEEE Hot Chips 27th Symposium (HCS), Cupertino, CA, USA, August 22–25, 2015. – 2015. – P. 1–24.

19. Syarif I., Prugel-Bennett A., Wills G. Data mining approaches for network intrusion detection from dimensionality reduction to misuse and anomaly detection. // Journal of Information Technology Review. – 2012. – 05. – Vol. 3. – P. 70–83.

20. Wei L., Keogh E., Xi A. SAXually explicit images: Finding unusual shapes. // Proceedings - IEEE International Conference on Data Mining, ICDM. – 2006.

21. Woodbridge D.M. Time series discord detection in medical data using a parallel relational database. / D.M. Woodbridge, A.T. Wilson, M.D. Rintoul, R.H. Goldstein. // 2015 IEEE International Conference on Bioinformatics and Biomedicine, BIBM 2015, Washington, DC, USA, November 9–12, 2015. – 2015. – P. 1420–1426. – URL: https://doi.org/10.1109/BIBM.2015.7359885.

22. Wu Y. Distributed Discord Discovery: Spark Based Anomaly Detection in Time Series. / Y. Wu, Y. Zhu, T. Huang, X. Li, X. Liu, et al. // 17th IEEE International Conference on High Performance Computing and Communications, HPCC 2015, 7th IEEE International Symposium on Cyberspace Safety and Security, CSS 2015, and 12th IEEE International Conference on Embedded Software and Systems, ICESS 2015, New York, NY, USA, August 24–26, 2015. – 2015. – P. 154–159. – URL: https://doi.org/10.1109/HPCC-CSS-ICESS.2015.228.

23. Xu L. A Hierarchical Framework Using Approximated Local Outlier Factor for Efficient Anomaly Detection. / L. Xu, Y.R. Yeh, Y.J. Lee, J. Li. // Procedia Computer Science. – 2013. – Vol. 19. – P. 1174 – 1181. – The 4th International Conference on Ambient Systems, Networks and Technologies (ANT 2013), the 3rd International Conference on Sustainable Energy Information Technology (SEIT-2013). URL: http://www.sciencedirect.com/science/article/pii/S187705091300776X.

24. Yankov D., Keogh E.J., Rebbapragada U. Disk Aware Discord Discovery: Finding Unusual Time Series in Terabyte Sized Datasets. // Proceedings of the 7th IEEE International Conference on Data Mining (ICDM 2007), October 28–31, 2007, Omaha, Nebraska, USA. – 2007. – P. 381–390. – URL: https://doi.org/10.1109/ICDM.2007.61.

25. Yankov D., Keogh E.J., Rebbapragada U. Disk aware discord discovery: finding unusual time series in terabyte sized datasets. // Knowl. Inf. Syst. – 2008. – Vol. 17. – No. 2. – P. 241–262. – URL: https://doi.org/10.1007/s10115-008-0131-9.

26. Абдуллаев С.М. Алгоритмы краткосрочного прогноза с использованием радиолокационных данных: оценка трасляции и композиционный дисплей жизненного цикла. / С.М. Абдуллаев, О.Ю. Ленская, А.О. Гаязова, О.Н. Иванова, А.А. Носков, и др.. // Вестник ЮУрГУ. Серия: Вычислительная математика и информатика. – 2014. – Т. 3. – № 1. – С. 17–32. – URL: https://doi.org/10.14529/cmse140102.

27. Воеводин В.В., Воеводин Вл.В. Параллельные вычисления. – СПб.: БХВ-Петербург, 2002. – 608 с.

28. Дышаев М.М., Соколинская И.М. Представление торговых сигналов на основе адаптивной скользящей средней Кауфмана в виде системы линейных неравенств. // Вестник ЮУрГУ. Серия: Вычислительная математика и информатика. – 2013. – Т. 2. – № 4. – С. 103–108. – URL: https://doi.org/10.14529/cmse130408.

29. Костенецкий П.С., Сафонов А.Ю. Суперкомпьютерный комплекс ЮУрГУ. // Параллельные вычислительные технологии (ПаВТ’2016): труды международной научной конференции (28 марта – 1 апреля 2016 г., г. Архангельск). – Издательский центр ЮУрГУ, 2016. – С. 561–573. – URL: http://omega.sp.susu.ru/PaVT2016/short/119.pdf.

30. Перечень оборудования Центра коллективного пользования Сибирского Суперкомпьютерного Центра ИВМиМГ СО РАН.. – Дата обращения: 03.11.2018. – [Электронный ресурс] URL: http://www.sscc.icmmg.nsc.ru/hardware.html.