МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ 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
Зам. директора по инф.
технологиям и безопасности
ГБУЗ “ЧОМИАЦ”

_____ А.С. Староверов

“____”_____ 2018 г.

ДОПУСТИТЬ К ЗАЩИТЕ

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

_____ Л.Б. Соколинский

“____”_____ 2018 г.

РАЗРАБОТКА ПАРАЛЛЕЛЬНОГО АЛГОРИТМА ПОИСКА САМОЙ ПОХОЖЕЙ ПОДПОСЛЕДОВАТЕЛЬНОСТИ ВРЕМЕННОГО РЯДА ДЛЯ МНОГОЯДЕРНЫХ ПРОЦЕССОРОВ INTEL XEON PHI (KNIGHTS LANDING)

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
ЮУрГУ – 02.03.02.2018.115-011.ВКР

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

_____ М.Л. Цымблер

Автор работы,
студент группы КЭ-401

_____ Я.А. Краева

Ученый секретарь
(нормоконтролер)

_____ О.Н. Иванова

“____”_____ 2018 г.

Челябинск-2018

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ 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

09.02.2018

ЗАДАНИЕ
на выполнение выпускной квалификационной работы бакалавра
студентке группы КЭ-401
Краевой Яне Александровне,
обучающейся по направлению
02.03.02 «Фундаментальная информатика и информационные технологии»

1. **Тема работы** (утверждена приказом ректора от 04.04.2018 № 580)
   Разработка параллельного алгоритма поиска самой похожей подпоследовательности временного ряда для многоядерных процессоров Intel Xeon Phi (Knights Landing)
2. **Срок сдачи студентом законченной работы:** 05.06.2018.
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. Rakthanmanon T., Campana B., Mueen A., Batista G., Westover B., Zhu Q., Zakaria J., Keogh E. Searching and Mining Trillions of Time Series Subsequences under Dynamic Time Warping // Proceedings of the 18th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Beijing, China, August 12–16, 2012. – ACM, 2012. – P. 262–270.
   - 3.4. Sart D., Mueen A., Najjar W., Keogh E., Niennattrakul V. Accelerating Dynamic Time Warping Subsequence Search with GPUs and FPGAs // Proceedings of the 10th IEEE International Conference on Data Mining, Sydney, NSW, Australia, December 13–17, 2010. – IEEE Computer Society, 2010. – P. 1001–1006.
   - 3.5. Ding H., Trajcevski G., Scheuermann P., Wang X., Keogh E. Querying and mining of time series data: experimental comparison of representations and distance measures // Proceedings of the VLDB Endowment, 2008. – Vol. 1. – No. 2. – P. 1542–1552.
4. **Перечень подлежащих разработке вопросов**
   - 4.1. Провести обзор параллельных алгоритмов поиска похожих подпоследовательностей временных рядов.
   - 4.2. Изучить аппаратную архитектуру и программную модель процессора Intel Xeon Phi (Knights Landing).

4.3. Спроектировать и реализовать параллельный алгоритм поиска самой похожей подпоследовательности временного ряда для многоядерных процессоров Intel Xeon Phi (Knights Landing).

4.4. Провести вычислительные эксперименты по анализу эффективности разработанного алгоритма.

5. **Дата выдачи задания:** 09.02.2018.

Научный руководитель

к.ф.-м.н., доцент М.Л. Цымблер

Задание принял к исполнению Я.А. Краева

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.

Поиск похожих подпоследовательностей во временном ряде часто используется как подпрограмма во многих задачах интеллектуального анализа временных рядов: обнаружении мотивов [12], классификации [1, 23], кластеризации [7]. Основной идеей задачи является нахождение k подпоследовательностей в заданном временном ряде, наиболее похожих на образец поиска.

Существуют области, в которых поиск похожих подпоследовательностей очень необходим, например, в мониторинге температуры, в медицине [13], в энтомологии [19], распознавании жестов и слов в рукописном тексте и др. Например, каждый день астрономы получают большое количество данных, измеренных с помощью антенной решетки. Поиск определенной последовательности в большой базе данных может помочь астрономам обнаружить типичные астрономические явления [4].

Схожесть двух подпоследовательностей определяет мера схожести. Среди десятков различных мер схожести, наилучшей мерой признана динамическая трансформация временной шкалы *(Dynamic Time Warping, DTW)* [5]. Однако классический расчет DTW трудоемок и сложен, из-за чего снижается эффективность алгоритмов, которые используют данную меру. Чтобы улучшить производительность алгоритмов на основе DTW, исследователями были предложены техники [3, 5, 9, 13, 15], которые ускоряют поиск самой похожей подпоследовательности. Однако вычисление расстояния DTW по-прежнему занимает слишком много времени, до 80 % всего времени поиска подобия [24]. Поэтому есть необходимость в распараллеливании DTW меры.

В связи с этим были предложены исследователями различные па-

раллельные реализации данного алгоритма на таких вычислительных системах, как кластерах [16, 20], многоядерных процессорах [18], GPU [24], FPGA [15, 22]. Однако перечисленные реализации не могут быть перенесены на Intel Xeon Phi. Поэтому объектом исследования данной работы является процессор Intel Xeon Phi.

Использование процессоров Intel Xeon Phi – новая тенденция в разработке аппаратных средств для суперкомпьютеров. 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) [17], отличается от первого поколения Phi, Knights Corner (KNC) [2], тем, что может выступать в качестве автономного процессора без центрального процессора. Данная работа является продолжением работ [11, 25], в которых представлена разработка поиска похожих подпоследовательностей в режиме offload, когда приложение запускается на хост-системе и некоторые вычислительно сложные участки программы выгружаются на сопроцессор Intel Xeon Phi (Knights Corner). Однако данная схема работы алгоритма слабо использует возможности векторизации Intel Xeon Phi и не применима для процессоров второго поколения Knights Landing, которые могут запускать программы только в native режиме.

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 энтомологии, распознавании речи и др.

Цель и задачи исследования

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

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

1) провести обзор параллельных алгоритмов поиска похожих подпоследовательностей временных рядов;

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

3) спроектировать и реализовать параллельный алгоритм поиска самой похожей подпоследовательности 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 31 страница, объем библиографии – 25 наименований.

Содержание работы

Первая глава, «Обзор работ по тематике исследования», содержит обзоры существующих параллельных алгоритмов поиска похожих подпоследовательностей 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 самой похожей подпоследовательности во временном ряде. Приведены техники ускорения поиска самой похожей подпоследовательности, и описан последовательный алгоритм UCR-DTW, на основе которого разработан параллельный алгоритм.

Третья глава, «Параллельный алгоритм phiBestMatch», содер-

жит детальное описание разработанного параллельного алгоритма 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 параллельные реализации поиска похожих подпоследовательностей на различных вычислительных системах.

Существуют работы, которые используют крупнозернистый параллелизм (coarse-grained). Авторы работы [20] распараллеливают вычисление расстояния DTW посредством технологии MPI. Один из процессов распределяет подпоследовательности между другими процессами. После чего каждый процесс вычисляет DTW расстояние с помощью SPRING алгоритма. Затем после завершения вычислений каждый CPU передает результат вычислений первому процессу, который находит самые похожие подпоследовательности. В другой работе [18] авторы распределяют вычисления DTW между нитями многоядерного процессора. Однако данные реализации не могут дать достаточного ускорения из-за большого количества передачи данных и тем более не ускоряют сам процесс вычисления DTW.

Сарт (Sart) и др. в своей работе [15] представили реализацию на GPU и FPGA для ускорения вычисления DTW. GPU реализация состоит из двух этапов: каждая нить параллельно нормализует подпоследовательность и вычисляет меру DTW. Далее, записывается результат вычисления в элемент массива, индекс которого равен номеру нити. После обработки всех подпоследовательностей массив копируется в оперативную память CPU для поиска минимального значения DTW. FPGA реализация генерируется технологией C-To-VHDL, называемой ROCCC. Для повышения эффективности используется параллелизм на уровне команд и повторно используются данные. Однако алгоритм недостаточно масштабируем. Кроме того, в алгоритме не используются какие-либо методы предварительной обработки данных, что не дает большего ускорения вычислений.

В работе [24] предложена реализация алгоритма на GPU, в которой использовался мелкозернистый параллелизм для вычисления меры DTW. Алгоритм сначала параллельно заполняет матрицу расстояний, т.е. каждый поток вычисляет элемент матрицы, равный расстоянию между двумя точками подпоследовательности и запроса. Однако на втором шаге вычисления DTW каждый поток одновременно для разных подпоследовательно-

стей формирует оптимальный путь транформации и подсчитывает его стоимость. Поскольку вычисление DTW разделено на два этапа, то необходимо хранить и перемещать матрицу расстояний, что оказывает нагрузку на всю систему.

В статье [22] предложили новый потоково-ориентированный фреймворк для поиска подобия подпоследовательностей и новую структуру PE-кольца (PE-ring) для вычисления DTW. Алгоритм использует технику ограничения оценки схожести снизу (lower bounding) для раннего отсечения подпоследовательностей и крупнозернистый параллелизм для поиска похожих подпоследовательностей. PE-кольцо использует мелкозернистый параллелизм для ускорения вычисления DTW меры.

Авторы статьи [4] предложили новую реализацию поиска похожей подпоследовательности на гетерогенной вычислительной платформе AMD, состоящей из CPU и GPU, используемого в качестве дополнительного вычислительного устройства. Данная работа является продолжением работы [22]. Данная реализация достигает большего ускорения по сравнению с предыдущей работой авторов и с другими существующими реализациями на GPU благодаря тому, что CPU выполняет нормализацию, а GPU параллельно вычисляет нижние оценки и меру DTW.

В работе [16] приведен параллельный алгоритм для кластера на основе фреймворка Apache Spark. Данный алгоритм использует оригинальный алгоритм UCR-DTW [13]. Spark API предоставляет широковещательные переменные в виде общих переменных, которые могут использоваться для распространения данных между рабочими узлами, но они не удовлетворяют, поскольку *bsf* должен быть перезаписан при каждом вычислении наилучшего результата. В итоге, это привело к уменьшению ускорения.

В работах [11, 25] представлена разработка алгоритма в режиме offload. Сопроцессор Intel Xeon Phi используется только для вычисления меры DTW. В то время как CPU выполняет технику lower bounding и заполняет очередь теми подпоследовательностями, которые не были отсечены техникой. После заполнения очереди кандидаты выгружаются на сопроцессор для вычисления меры DTW. Результаты экспериментов показали, что алгоритм превосходит параллельные алгоритмы для GPU и FPGA.

2. ФОРМАЛЬНЫЕ ОБОЗНАЧЕНИЯ И ПОСТАНОВКА ЗАДАЧИ

2.1. Формальные обозначения

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

Временной ряд (time series) T – хронологически упорядоченная последовательность вещественных значений $t_1, t_2, ..., t_m$, где m – длина последовательности.

Динамическая трансформация временной шкалы (Dynamic Time Warping, DTW) – мера схожести, которая подходит для сравнения таких двух временных рядов, которые сжаты, растянуты, смещены во времени (вдоль оси Ox) или имеющие разные длины. Мера DTW сопоставляет каждое значение одного временного ряда со всеми значениями другого временного ряда. Далее на протяжении всей работы будет рассматриваться случай, когда временные ряды имеют одинаковые длины m [13]. Это упрощает изложение и не ограничивает общности. Мера DTW между $X = (x_1, x_2, ..., x_m)$ и $Y = (y_1, y_2, ..., y_m)$ вычисляется следующим образом:

$$DTW(X, Y) = d(m, m);$$

$$d(i, j) = \|x_i - y_j\| + \min \begin{cases} d(i-1, j) \\ d(i, j-1) \\ d(i-1, j-1) \end{cases}; \qquad (1)$$

$$d(0, 0) = 0;\ d(i, 0) = d(0, j) = \infty;\ 1 \le i \le m;\ 1 \le j \le m,$$

где $\|\cdot\|$ – евклидова норма. Также в качестве функции расстояния может выступать норма ℓ_1 (манхэттенское расстояние).

Для вычисления DTW строится матрица трансформации, элементы которой вычисляются по формуле (1). Далее строится путь трансформации P.

Путь трансформации P (warping path) – последовательность матричных элементов, которая характеризует соответствие между Q и C, т.е. $P = (p_1, .., p_L)$, где $p_\ell = d(i, j)_\ell$, $n \le L \le 2 \cdot n - 1$, L – длина пути. Поскольку существует большое количество путей трансформации, то берется

тот путь, который минимизирует DTW расстояние (стоимость пути):

$$DTW(Q, C) = \sqrt{\sum_{\ell=1}^{L} p_\ell} \tag{2}$$

Подпоследовательностью (subsequence) $T_{i,n}$ временного ряда T называется непрерывное подмножество T длины n $(n \leq m)$, начинающееся с позиции i: $T_{i,n} = (t_i, t_{i+1}, ..., t_{i+n-1}),\ 1 \leq i \leq m - n + 1, n \ll m$.

Для ясности изложения алгоритма подпоследовательность $T_{i,n}$ будем называть кандидатом C, а временной ряд, сравнивающийся с кандидатами и имеющий такую же длину n, что и кандидаты, называть *запросом (query)* Q.

Опишем задачу *поиска самой похожей подпоследовательности во временном ряде (best-match search)*. Пусть дано множество кандидатов, составленных из временного ряда T, и запрос Q. Тогда самой похожей подпоследовательностью будет называться такая подпоследовательность $T_{i,n}$, которая максимально похожа на запрос Q, т.е.

$$\exists i\ \forall k\ DTW(Q, T_{i,n}) \leq DTW(Q, T_{k,n});\ 1 \leq i, k \leq m - n + 1. \tag{3}$$

2.2. Последовательный алгоритм UCR-DTW

Мера DTW имеет большую временную сложность $O(n^2)$. Обозначим за K – количество подпоследовательностей длины n временного ряда T, который имеет длину m. Тогда $K = m - n + 1$. Поэтому временная сложность алгоритма поиска самой похожей последовательности возрастет до $O(K \cdot n^2)$.

Для ускорения поиска самой похожей подпоследовательности были предложены техники, сложность которых меньше, чем $O(n^2)$. Одним из наиболее эффективных методов является техника *ограничения оценки схожести снизу (lower bounding)* [3, 5, 9], которая позволяет отбрасывать непохожие на образец подпоследовательности без вычисления меры DTW. В качестве оценок схожести выступают функции $LB(Q, T_{i,n})$, которые имеют меньшую вычислительную сложность, чем мера DTW. Обозначим за *bsf (best-so-far)* – лучшую текущую нижнюю оценку схожести. Идея *lower*

bounding заключается в том, что если нижняя оценка $LB(Q, T_{i,n})$ превышает *bsf*, тогда DTW тоже превысит bsf и подпоследовательность отбросится как непохожая на образец. Алгоритм UCR-DTW использует LB_{KimFL}, LB_{Keogh}, которые описаны ниже. Далее подробно описывается техника *lower bounding*.

Пусть дан временной ряд T и запрос Q. При последовательном переборе всех подпоследовательностей временного ряда T для текущей подпоследовательности $T_{i,n}$ вычисляется bsf_i, которая инициализируется $+\infty$:

$$bsf_i = \begin{cases} +\infty, & if\ LB(Q, T_{i,n}) > bsf_{local} \\ DTW(Q, T_{i,n}), & otherwise \end{cases}, \tag{4}$$

где bsf_{local} – локальный минимум нижней оценки схожести, который определяется следующим образом:

$$\exists local\ \forall k\ bsf_{local} \leq bsf_k;\ k \in \dot{U}(local, 0, i+1);\ 0 \leq i \leq m-n. \tag{5}$$

В формуле (5) под $\dot{U}(local, 0, i+1)$ понимается проколотая окрестность точки $local$:

$$\dot{U}(local, 0, i+1) = \{k : 0 < local - k < i+1\}. \tag{6}$$

bsf_i равна $DTW(Q, T_{i,n})$ в том случае, если подпоследовательность не была отброшена техникой *lower bounding*. После выбирается минимальное значение среди bsf_{local} и bsf_i:

$$bsf_{local} = \min(bsf_{local}, bsf_i) \tag{7}$$

Кроме этого, существуют и другие техники, такие как *ранний отказ (early abandoning)* [13], основанный на сравнении промежуточного значения $DTW(Q_{1,k}, C_{1,k})$, $1 \leqslant k \leqslant m-n+1$ с пороговым значением ε. Если промежуточное значение $DTW(Q_{1,k}, C_{1,k})$ превышает ε, то нет необходимости в досчитывании DTW, поскольку данная подпоследовательность не похожа на запрос.

Данная работа основана на алгоритме *UCR-DTW* [13], в котором используется перечисленные выше техники. До сравнения подпоследовательностей необходимо их преобразовать, выполнить *Z-нормализацию*, для

более точной работы алгоритма [21]. Z-нормализация необходима в том случае, если запрос и кандидат имеют практически одинаковые формы, но различны по амплитуде, т.е. $q_i = c_i + k$, k – константа. После нормализации среднее арифметическое подпоследовательности приблизительно равно 0, а среднеквадратичное отклонение близко к 1. Нормализованный временной ряд длины n обозначим как $\hat{C} = (\hat{c}_1, .., \hat{c}_n)$, значения которого вычисляются по формуле (9):

$$\hat{c}_i = \frac{c_i - \mu}{\sigma};\ 1 \leq i \leq n, \tag{8}$$

где: $\mu = \frac{1}{n}\sum_{i=1}^{n} c_i$ – среднее арифметическое; $\sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n} c_i^2 - \mu^2}$ – среднеквадратическое отклонение.

Идея алгоритма *UCR-DTW* состоит в использовании следующих техник для ускорения DTW.

Использование квадрата евклидова расстояния. В качестве функции расстояния между кандидатом и запросом используется квадрат евклидова расстояния вместо корня квадратного в формуле расчета DTW (2).

Использование нижних границ схожести lower bounding. Алгоритм *UCR-DTW* использует такие функции *LB*, как LB_{KimFL} [13], LB_{Keogh} [8]:

- LB_{KimFL} – оценка, полученная из расстояния между первой и последней парами точек Q и C. Вычислительная сложность составляет $O(1)$.

$$LB_{KimFL} = \|\hat{q}_1 - \hat{c}_1\| + \|\hat{q}_n - \hat{c}_n\| \tag{9}$$

- LB_{Keogh} – оценка, для вычисления которой необходимо найти оболочку E (envelope) для запроса Q. Оболочка E ограничивается верхней и нижней последовательностями, $U(Upper)$ и $L(Lower)$ соответственно которые вычисляются по формуле (10):

$$\begin{aligned} U_i &= \max(\hat{q}_{i-r}, .., \hat{q}_{i+r}); \\ L_i &= \min(\hat{q}_{i-r}, .., \hat{q}_{i+r}), \end{aligned} \tag{10}$$

где параметр $r > 0$ в технике *Sakoe-Chiba band* [14] ограничивает путь наименьшей стоимости, не позволяя уходить больше, чем на r элементов от диагонали матрицы трансформации. Оценка между кандидатом и оболочкой запроса вычисляется по формуле (11). Вычислительная сложность

техники LB_{Keogh} составляет $O(n)$.

$$LB_{Keogh}(\hat{Q},\hat{C}) = \sum_{i=1}^{n} \begin{cases} (\hat{c}_i - U_i)^2, & if\ \hat{c}_i > U_i \\ (\hat{c}_i - L_i)^2, & if\ \hat{c}_i < L_i \\ 0, & otherwise \end{cases} \tag{11}$$

Смена ролей запроса $\hat{Q}$ и кандидата $\hat{C}$ в LB_{Keogh}. Данная оценка вычисляется между запросом $\hat{Q}$ оболочкой кандидата $\hat{C}$ по формуле (11) , причем $LB_{Keogh}(\hat{Q},\hat{C}) \neq LB_{Keogh}(\hat{C},\hat{Q})$. Вычислительная сложность данной техники составляет $O(n)$.

Раннее прекращение вычисления LB_{Keogh}. Если промежуточная сумма (11) превышает bsf, то вычисление заканчивается, и кандидат отбрасывается. Временная сложность варьируется от $O(1)$ до $O(n)$.

Раннее прекращение вычисления DTW. Техника заключается в частичном вычислении $DTW(\hat{Q},\hat{C})$ и $LB_{Keogh}(\hat{Q},\hat{C})$. Если текущая сумма $DTW(\hat{Q}_{1,k},\hat{C}_{1,k}) + LB_{Keogh}(\hat{Q}_{k+1,n},\hat{C}_{k+1,n})$, где $k = 1, ..., n-1$, превышает bsf, то подпоследовательность отбрасывается. Временная сложность изменяется от $O(1)$ до $O(n^2)$.

Одновременное вычисление оценки LB_{Keogh} и Z-нормализации. Идея данного алгоритма заключается в том, что Z-нормализация считается одновременно с LB_{Keogh}. Если LB_{Keogh} превышает bsf, то вычисление Z-нормализации прекращается вместе с LB_{Keogh}.

Переупорядочивание абсолютных значений. Данная оптимизация переупорядочивает абсолютные значения в подпоследовательности в порядке их возрастания для выполнения меньшего количества итераций подсчета LB_{Keogh}. Временная сложность составляет $O(n)$.

Каскадное применение оценок. Каскадное применение оценок заключается в последовательном вычислении нижних оценок LB. Сначала вычисляется LB_{KimFL}, затем, если кандидат не был отброшен, то вычисляется $LB_{Keogh}(\hat{Q},\hat{C})$. Если $LB_{Keogh}(\hat{Q},\hat{C})$ не превысил bsf, то вычисляется $LB_{Keogh}(\hat{C},\hat{Q})$.

3. ПАРАЛЛЕЛЬНЫЙ АЛГОРИТМ PHIBESTMATCH

3.1. Краткое описание параллельного алгоритма phiBestMatch

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 [6].

Оригинальный алгоритм *UCR-DTW* [13] по построению не обеспечивает векторизацию и выравнивание данных. Алгоритм считывает из файла одну подпоследовательность и сравнивает ее с запросом. Так алгоритм обрабатывает все подпоследовательности, одну за другой. Данная схема не позволяет сделать код векторизованным. Кроме того, в алгоритме реализовано каскадное применение оценок, из-за чего используется множество условных конструкций *if-else* и существуют зависимости по данным. В соответсвии с этим в рамках данной работы был спроектирован новый параллельный алгоритм *phiBestMatch*, используемый иную схему организации вычислений и данных. Поскольку алгоритм *phiBestMatch* связан с параллельной обработкой матриц, то матрицы подтвергаются сегментации для равномерного распределения вычислительной нагрузки между нитями. В данной работе рассматривается тот случай, когда количество сегментов s равно количеству нитей p. Введем обозначение ширины сегмента $w_{segm} = [\frac{K}{s}]$ и массива pos_{segm} длины s, предназначенного для хранения индексов первой необработанной подпоследовательности в каждом сегменте.

На рис. 1 представлен превдокод разработанного параллельного алгоритма *phiBestMatch*.

Algorithm 1 phiBestMatch(in **Q**, **T**, **r**, **k**, **p**; out $\mathbf{bsf_{local}}$, **index**)

1: Block'n'Pad(T)
2: Znormalize(Q)
3: CalcEnvelope($\hat{Q}$, r)
4: CalculateLB($MatrixC$, $\hat{Q}$, r)
5: bsf_{local} ←DTW($\hat{Q}$, $\hat{T}_{1,n}$, r, $+\infty$)
6: **while** *not handle all Candidates* **do**
7: LowerBounding($lb1$, $lb2$, $lb3$, BM)
8: FillMatrixDTW($\widehat{MatrixC}$, BM, $MDTW$, $Index_C$)
9: CalculateDTW($MDTW$, $Index_C$, bsf_{local}, $index$)

Рис. 1. Псевдокод параллельного алгоритма *phiBestMatch*

Алгоритм *phiBestMatch* включает следующие основные шаги. *Block'n'Pad* считывает из текстового файла временной ряд и создает структуру данных для хранения подпоследовательностей временного ряда в виде матрицы.

Далее считанный из файла запрос Q нормализуется (строка 2), и строится для него оболочка (строка 3). Оболочка запроса строится один раз для всех подпоследовательностей, используемая в оценке $LB_{Keogh}(Q, C)$.

CalculateLB предназначен для Z-нормализации всех подпоследовательностей, хранящихся в матрице, и для раннего вычисления оценок LB_{KimFL}, $LB_{Keogh}(\hat{Q}, \hat{C})$, $LB_{Keogh}(\hat{C}, \hat{Q})$.

Строка 5 в алгоритме 1 обозначает инициализацию bsf_{local} значением DTW меры между первой нормализованной подпоследовательностью $\hat{T}_{1,n}$ и нормализованным запросом $\hat{Q}$.

Модуль *LowerBounding* на основании ранее подсчитанных оценок отбрасывает те подпоследовательности, оценки которых больше, чем bsf_{local}.

FillMatrixDTW заполняет матрицу подпоследовательностями, заведомо похожими на образец поиска, для каждой из которой вычисляется мера DTW в модуле *CalculateDTW*, и матрицу, элементами которой являются индексы этих подпоследовательностей. Если DTW какой-либо подпоследовательности меньше, чем bsf_{local}, то bsf_{local} обновляется и индекс лучшей на данный момент подпоследовательности запоминается.

Модули *LowerBounding, FillMatrixDTW, CalculateDTW* выполняются до тех пор, пока все подпоследовательности не будут обработаны. В результате алгоритм выводит индекс самой лучшей подпоследовательности временного ряда и bsf_{local}.

Далее детально описываются модули алгоритма *phiBestMatch*.

3.2. Модуль Block'n'Pad

На первом шаге алгоритма *phiBestMatch* считывается временной ряд T длины m из текстового файла в матрицу подпоследовательностей $MatrixC_{K \times W}$. Матрица $MatrixC$ имеет $K = m - n + 1$ строк, равное количеству подпоследовательностей, и W столбцов. Количество столбцов должно быть кратно 16, чтобы использовать всю ширину векторных регистров w_{VPU} для ускорения. Если длина подпоследовательностей n не кратно 16, то матрица подпоследовательностей дополняется незначащими нулями в количестве $w_{VPU} - (n \; mod \; w_{VPU})$. Матрица $MatrixC$ хранит данные с плавающей точкой одинарной точности (float) и определяется следующим образом:

$$\mathbb{R}^m \times \mathbb{N}^2 \to \mathbb{R}^{K \times W},$$

$$W = \begin{cases} n, & if \; n \vdots w_{VPU}; \\ n + w_{VPU} - (n \; mod \; w_{VPU}), & otherwise. \end{cases} \tag{12}$$

3.3. Модуль CalculateLB

Данный модуль предназначен для раннего вычисления оценок LB_{KimFL}, $LB_{Keogh}(\hat{Q}, \hat{C})$, $LB_{Keogh}(\hat{C}, \hat{Q})$. Псевдокод *CalculateLB* приведен в алгоритме 2 (рис. 2). Строка 2 обозначает цикл, итерации которого распределены между p нитями. На каждой i итерации для i подпоследовательности данный цикл нормализует кандидат (строка 3), вычисляет оценки LB_{KimFL}, $LB_{Keogh}(\hat{Q}, \hat{C})$, $LB_{Keogh}(\hat{C}, \hat{Q})$ (строки 4, 5, 7) и заполняет оценками массивы $lb1$, $lb2$, $lb3$ длины K соответственно. В строке 6 вычисляется оболочка кандидата, которая используется в подсчете оценки $LB_{Keogh}(\hat{C}, \hat{Q})$. По сравнению с последовательным алгоритмом, где оцен-

ки вычисляются каскадно, в разработанном параллельном алгоритме оценки вычисляются все сразу однократно. Распараллеливание цикла компенсирует избыточность вычислений всех нижних оценок схожести.

Algorithm 2 CalculateLB(in **MatrixC**, **Q**; out **lb1**, **lb2**, **lb3**)

1: #pragma omp parallel for num_threads(p)
2: **for** i **from** 1 **to** K **do**
3: $Znormalize(MatrixC(i,\cdot))$
4: $lb1(i) \leftarrow LB_{KimFL}(\hat{Q}, \widehat{MatrixC}(i,\cdot))$
5: $lb2(i) \leftarrow LB_{Keogh}(\hat{Q}, \widehat{MatrixC}(i,\cdot))$
6: $CalcEnvelope(\widehat{MatrixC}(i,\cdot), r)$
7: $lb3(i) \leftarrow LB_{Keogh}(\hat{C}, \hat{Q})$

Рис. 2. Псевдокод *CalculateLB*

Реализация процедуры *CalcEnvelope* была изменена, поскольку в базовом алгоритме *UCR-DTW* оболочка E вычисляется с помощью алгоритма *Lemire* [10], который не векторизуется из-за большого количества циклов while, использования неподходящих структур данных и зависимостей по данным. В алгоритме *phiBestMatch* вычисляется по формуле (10). Все процедуры и функции в модуле *CalculateLB* векторизуются.

3.4. Модуль LowerBounding

Алгоритм 3 (рис. 3) представляет модифицированную технику отбрасывания заведомо непохожих подпоследовательностей на поисковый запрос *Lower Bounding*, идея которой состоит в следующем.

Введем матрицу $BM_{K\times(L+1)}$, где $BM(i,j) \in B^{K\times(L+1)}$, L – количество ранее подсчитанных оценок в предыдущем модуле $CalculateLB$. Значения матрицы BM вычисляются путем сравнения каждой оценки схожести с bsf_{local}, т.е. первый столбец матрицы BM заполняется следующим образом:

$$BM(i,1) = \begin{cases} 0, & if\ lb1(i) \geq bsf_{local} \\ 1, & if\ lb1(i) < bsf_{local} \end{cases}, \text{ где } 1 \leq i \leq K. \qquad (13)$$

Для оставшихся оценок столбцы заполняются аналогичным образом. $(L+1)$ столбец содержит результат от операции конъюнкции L первых столбцов. Это означает, что если хотя бы одна из оценок превышает bsf_{local}, следовательно, такая подпоследовательность отбрасывается как непохожая на образец поиска и не учитывается в последующих вычислениях меры DTW.

$$BM(i,(L+1)) = \bigwedge_{j=1}^{L} BM(i,j), \text{ где } 1 \le i \le K. \tag{14}$$

Цикл в строке 6 автоматически векторизуется и распараллеливается. Для эффективной векторизации цикла каждый массив $lb1$, $lb2$, $lb3$ со значениями вычисленных оценок схожести выравнен в оперативной памяти (строки 3–5). В течение всего алгоритма поиска самой похожей подпоследовательности матрица BM сканируется параллельно несколько раз до тех пор, пока не отбросятся все подпоследовательности, кроме одной. Каждая нить обрабатывает свой сегмент. Для того чтобы не сканировать сегмент с начала до конца, каждый сегмент обрабатывается с той подпоследовательности, на которой остановились на предыдущей итерации. Таким образом, *Lower Bounding* не выполняется для тех подпоследовательностей, которые были отброшены или для которых уже была вычислена мера DTW.

Algorithm 3 LowerBounding(in **lb1**, **lb2**, **lb3**; out **BM**)

```
1:  #pragma omp parallel num_threads(p)
2:  num_thread ←omp_get_thread_num()
3:  __assume_aligned(lb1, 64)
4:  __assume_aligned(lb2, 64)
5:  __assume_aligned(lb3, 64)
6:  for i from pos_s[num_thread] to num_thread · w_s do
7:      BM(i,1) ← (lb1(i) < bsf)
8:      BM(i,2) ← (lb2(i) < bsf)
9:      BM(i,3) ← (lb3(i) < bsf)
10:     BM(i,4) ← BM(i,1) ∧ BM(i,2) ∧ BM(i,3)
```

Рис. 3. Псевдокод *LowerBounding*

3.5. Модуль FillMatrixDTW

После проверки всех подпоследовательностей на схожесть с запросом с помощью алгоритма *Lower Bounding* заполняется матрица подпоследовательностей-кандидатов $MDTW \in \mathbb{R}^{(k \cdot p) \times n}$, где $k \in \mathbb{Z}$ – параметр алгоритма, до тех пор, пока не заполнятся все строки матрицы теми подпоследовательностями, которые не были отброшены в модуле *LowerBounding*. Для заполнения матрицы $MDTW$ необходимо циклически обойти $MatrixC$ и BM. Данный обход напоминает принцип Round-robin: берется первая подпоследовательность из первого сегмента и проверяется, была ли она отброшена, т.е. $BM(i, L+1) = 0$. Если подпоследовательность не была отброшена, тогда эта подпоследовательность добавляется в матрицу $MDTW$ как заведомо похожая на образец поиска и ее индекс записывается в матрицу $Index_C$. Несмотря на то, была ли подпоследовательность добавлена в матрицу $MDTW$ или нет, переходим ко второму сегменту и проверяем на схожесть первую подпоследовательность второго сегмента. Таким образом обрабываются первые подпоследовательности всех сегментов. Как только обработали первые подпоследовательности всех сегментов вовращаемся к первому сегменту и проверяем на схожесть вторую подпоследовательность. Данный процесс продолжается до тех пор, пока не заполнится матрица $MDTW$. При этом каждый раз обновляется текущая позиция в num_{thread} сегменте, т.е. $pos_s(num_{thread}) = pos_s(num_{thread}) + 1$, чтобы в следующий раз, при повторном обходе сегмента, проверять на схожесть ту подпоследовательность, на которой остановились на предыдущей итерации. После заполнения матрицы $MDTW$ вызывается модуль *CalculateDTW*.

Данный модуль невозможно распараллелить и векторизовать из-за неизвестного и неравномерного количества в сегментах похожих на образец поиска подпоследовательностей. В результате одни нити могут простаивать из-за того, что все подпоследовательности уже отброшены или для них уже вычислена мера DTW, а другие продолжать выполнять поиск тех подпоследовательностей, у которых все оценки не превышают локальный минимум в своем сегменте. Поэтому заполнение матрицы $MDTW$ подпоследовательностями-кандидатами выполняется последовательно.

3.6. Модуль CalculateDTW

В модуле *CalculateDTW* (рис. 4) вычисляется DTW мера для кандидатов из матрицы $MDTW$ и запроса (строка 3). Вычисления DTW распределяются статически между нитями для сбалансированной нагрузки нитей. Каждая нить вычисляет DTW для k подпоследовательностей. Если bsf_i меньше, чем bsf_{local}, то bsf_{local} обновляется на bsf_i (строки 4–7) и индекс локально похожей подпоследовательности запоминается.

Algorithm 4 CalculateDTW(in **MDTW**, **Index$_C$**, **Q**, **r**; out **bsf$_{local}$**, **index**)

1: #pragma omp parallel for shared(bsf_{local}, index) private(bsf_i)
2: **for** i **from** 1 **to** $k \cdot num_{threads}$ **do**
3: $\quad bsf_i \leftarrow DTW(Q, MDTW(i, \cdot), r, bsf_{local})$
4: $\quad$ #pragma omp critical
5: $\quad$ **if** $bsf_{local} > bsf_i$ **then**
6: $\quad\quad bsf_{local} \leftarrow bsf_i$
7: $\quad\quad index \leftarrow Index_C(i)$

Рис. 4. Псевдокод *CalculateDTW*

В работах [11, 25] представлена модификация вычисления DTW. Данная модификация убыстряется вычисление меры DTW благодаря векторизации. Авторы разбивают цикл, в котором строится матрица трансформации, выбирается оптимальный пути трансформации P и подсчитывается стоимость этого пути, на два цикла. Поскольку первый цикл не содержит зависимостей по данным, следовательно, он векторизуется.

Репозиторий с кодом программы *phiBestMatch* расположен по адресу: `https://bitbucket.org/YanaKraeva/phibestmatch`.

4. ВЫЧИСЛИТЕЛЬНЫЕ ЭКСПЕРИМЕНТЫ

В данном разделе представлены результаты экспериментов по исследованию эффективности разработанного параллельного алгоритма *phiBestMatch* поиска самой похожей подпоследовательности, описанного в главе 3.

4.1. Цели экспериментов

Вычислительные эксперименты для оценки эффективности параллельного алгоритма *phiBestMatch* поиска самой похожей подпоследовательности временного ряда проводились следующим образом. 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 вычислительных экспериментах были проведены исследования ускорения и параллельной эффективности алгоритма *phiBestMatch*, а также проводились эксперименты для исследования зависимости показателей масштабируемости (ускорения и эффективности) от различных значений параметров k и r.

Ускорение и эффективность параллельного алгоритма, выполняемого на k нитях, рассчитываются по формулам (15) и (16) соответственно следующим образом:

$$a(k) = \frac{t_1}{t_k}, \tag{15}$$

$$e(k) = \frac{a(k)}{k}, \tag{16}$$

где: t_1 – время выполнения последовательного алгоритма; t_k – время выполнения параллельного алгоритма на k нитях; k – количество нитей, на которых выполняется параллельный алгоритм.

Все вычислительные эксперименты проводились с использованием многоядерного процессора Intel Xeon Phi суперкомпьютера «Торнадо ЮУрГУ». Технические характеристики процессора Intel Xeon Phi представлены в табл. 1.

Табл. 1. Технические характеристики процессора Intel Xeon Phi суперкомпьютера «Торнадо ЮУрГУ»

Процессор	Intel Xeon Phi SE10X
Количество физических ядер	61
Количество нитей на ядро	4
Количество логических ядер	244
Тактовая частота, ГГц	1.1
Производительность, TFLOPS	1.076
Ширина векторных регистров, бит	512

4.2. Наборы данных

Эксперименты проводились над синтетическими и реальными данными. Из-за ограниченности оперативной памяти процессора Intel Xeon Phi (8 Гб) не рассматривались очень большие временные ряды длиной выше 10^7. Поэтому брались только такие наборы данных, которые полностью помещались в оперативную память процессора Intel Xeon Phi. Максимальная длина временного ряда составляла 10^6 точек. Такие ряды, которые имеют длину не больше 10^6 точек, используются в различных областях человеческой деятельности.

Синтетические данные были взяты из оригинального алгоритма UCR-DTW [13]. Временные ряды были сгенерированы с помощью математической модели случайного блуждания *Random walk*, где каждое следующее значение временного ряда изменяется на некоторую величину с одинаковой вероятностью. Данный временной ряд состоит из 10^6 точек, длина запроса равна 128.

В качестве реальных данных брался временной ряд *EPG* из области энтомологии [15]. Данные содержат колебательное напряжение, возникающее при взаимодействии насекомого с растением. Таким образом, благодаря показаниям EPG биологи изучают основы передачи вируса растениям, тем самым они борются с вирусами, распространяемыми насекомыми, и защищают растения от их вредного влияния. Временной ряд состоит из $2.5 \cdot 10^5$ значений, а запрос из 360 значений.

4.3. Результаты экспериментов

Масштабируемость

На рис. 5 и рис. 6 представлены графики ускорения и эффективности параллельного алгоритма *phiBestMatch* для наборов данных Random walk и EPG. По графикам можно сделать вывод, что ускорение близко к линейному и параллельная эффективность больше 50 % тогда, когда алгоритм запущен на таком количестве нитях, которое соответствует количеству физических ядер процессора Intel Xeon Phi. Далее наблюдается загоризонталивание графика ускорения и падение эффективности до 30 %. Это возникает по следующим двум причинам. Во-первых, алгоритм не полностью параллельный, поскольку заполнение матрицы подпоследовательностями-кандидатами выполняется последовательно. Во-вторых, вычисления DTW меры слабо векторизуются, поскольку DTW мера является задачей динамического программирования.

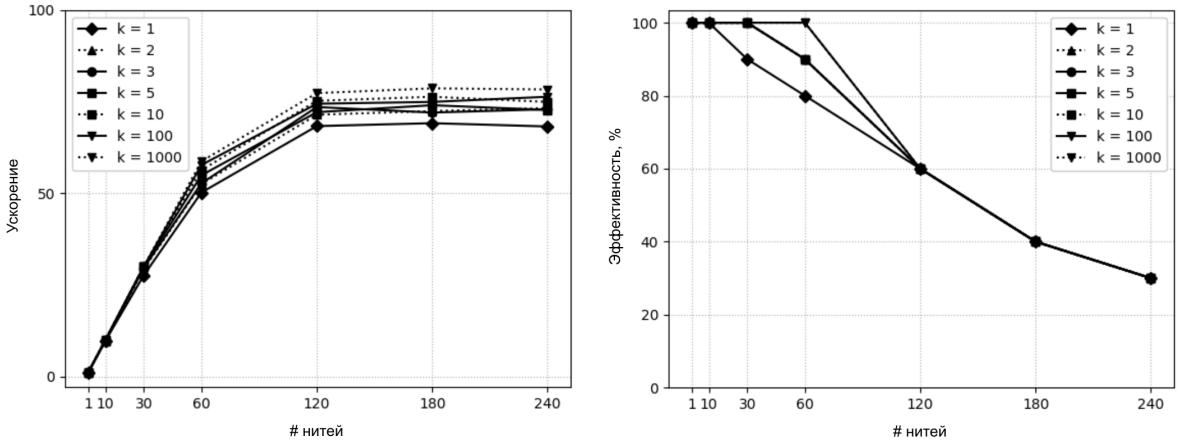
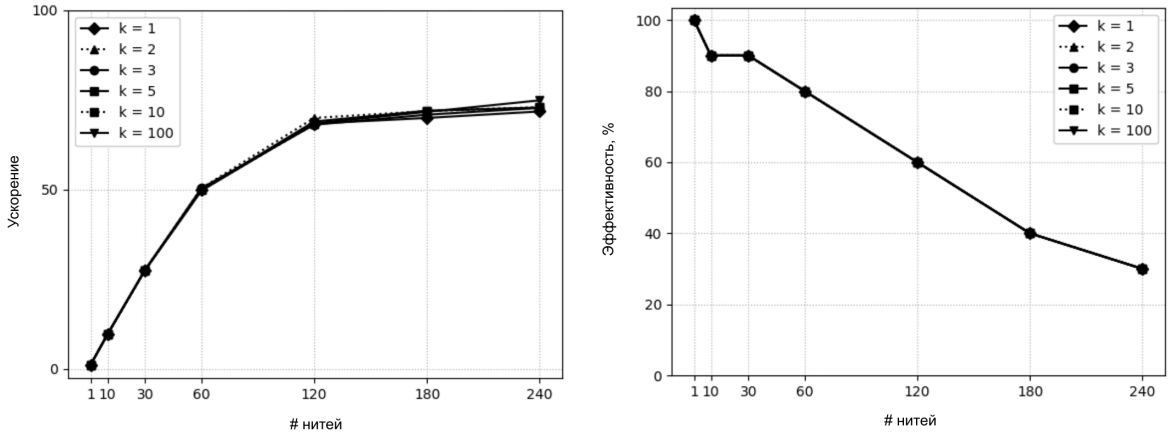


Рис. 5. Исследование ускорения и эффективности на наборе данных Random walk

Исследование зависимости ускорения и эффективности от параметра *k*

На рис. 5 и рис. 6 представлены результаты исследования влияния параметра *k* на ускорение и эффективность параллельного алгоритма. Как показывают графики, при увеличении параметра *k* ускорение увеличивается на незначительную величину. Такой маленький прирост ускорения можно объяснить следующим образом. Поскольку размер матрицы *MDTW* увели-



Ускорение Эффективность

Рис. 6. Исследование ускорения и эффективности на наборе данных EPG

чивается в *k* раз, то каждая нить параллельно вычисляет DTW у *k* подпоследовательностей. Несмотря на большое количество одновременных подсчетов DTW, обсчитывается DTW мера у таких подпоследовательностей, DTW которых не улучшает значение лучшей текущей оценки bsf_{local}. Таким образом, вычисляется лишнее DTW.

Исследование зависимости ускорения от параметра *r*

На рис. 7 представлены результаты экспериментов по исследованию зависимости ускорения от параметра r. Эксперименты проводились над синтетическими данными *Random walk*, где временной ряд состоит из 10^6 точек, а запрос из 128 точек. Параметр *k* был равен 1000. Эксперименты показали, что параллельный алгоритм более эффективен при большем значении *r*. Это говорит о том, что чем больше нагружать сложными и ресурсоемкими вычислениями процессор Intel Xeon Phi, тем быстрее выполняется поиск самой лучшей подпоследовательности во временном ряде.

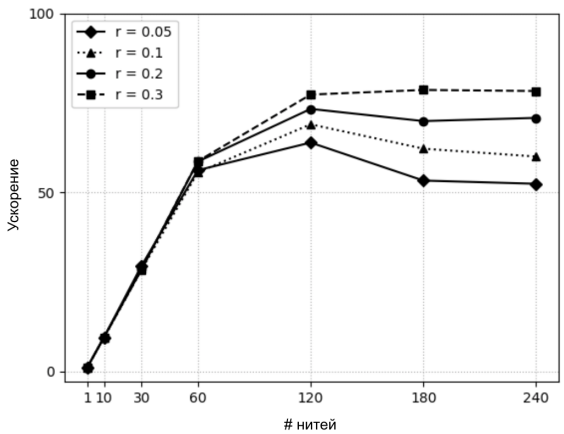


Рис. 7. Зависимость ускорения от параметра *r*

ЗАКЛЮЧЕНИЕ

Данная выпускная квалификационная работа была посвящена разработке параллельного алгоритма поиска самой похожей подпоследовательности временного ряда для многоядерных процессоров Intel Xeon Phi (Knights Landing).

В ходе выполнения работы были получены следующие основные результаты:

1) проведен обзор параллельных алгоритмов поиска похожих подпоследовательностей 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 самой похожей подпоследовательности временного ряда для многоядерных процессоров Intel Xeon Phi (Knights Landing);

4) проведены вычислительные эксперименты на реальных и синтетических данных, показавшие хорошую масштабируемость разработанного алгоритма.

В ходе выпускной квалификационной работы были сделаны доклады на 71-ой студенческой научной конференции ЮУрГУ и XIII Уральской выставке НТТМ «Евразийские ворота России – Шаг в будущее».

В рамках выпускной работы была подана статья на XX международную конференцию «Data Analytics and Management in Data Intensive Domains 2018».

ЛИТЕРАТУРА

1. Athitsos V. Approximate embedding-based subsequence matching of time series. / V. Athitsos, P. Papapetrou, M. Potamias, G. Kollios, D. Gunopulos. // Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2008, Vancouver, BC, Canada, June 10-12, 2008. – 2008. – P. 365–378. – URL: http://doi.acm.org/10.1145/1376616.1376656.

2. 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.

3. Fu A.W. Scaling and time warping in time series querying. / A.W. Fu, E.J. Keogh, L.Y.H. Lau, C.A. Ratanamahatana, R.C. Wong. // VLDB J. – 2008. – Vol. 17. – No. 4. – P. 899–921. – URL: https://doi.org/10.1007/s00778-006-0040-z.

4. Huang S. DTW-Based Subsequence Similarity Search on AMD Heterogeneous Computing Platform. / S. Huang, G. Dai, Y. Sun, Z. Wang, Y. Wang, et al. // 10th IEEE International Conference on High Performance Computing and Communications & 2013 IEEE International Conference on Embedded and Ubiquitous Computing, HPCC/EUC 2013, Zhangjiajie, China, November 13-15, 2013. – 2013. – P. 1054–1063. – URL: https://doi.org/10.1109/HPCC.and.EUC.2013.149.

5. Hui D. Querying and Mining of Time Series Data: Experimental Comparison of Representations and Distance Measures. / D. Hui, T. Goce, S. Peter, W. Xiaoyue, K. Eamonn. // Proceedings of the VLDB Endowment. – 2008. – Vol. 1. – No. 2. – P. 1542–1552. – URL: http://dx.doi.org/10.14778/1454159.1454226.

6. Jeffers J., Reinders J. Intel Xeon Phi Coprocessor High Performance Programming. 1st ed. – San Francisco, CA, USA : Morgan Kaufmann Publishers Inc., 2013.

7. Keogh E., Lin J. Clustering of time-series subsequences is meaningless: implications for previous and future research. // Knowledge and Information Systems. – 2005. – Aug. – Vol. 8. – No. 2. – P. 154–177. – URL: https://doi.org/10.1007/s10115-004-0172-7.

8. Keogh E.J., Ratanamahatana C.A. Exact indexing of dynamic time warping. // Knowl. Inf. Syst. – 2005. – Vol. 7. – No. 3. – P. 358–386. – URL: http://www.springerlink.com/index/10.1007/s10115-004-0154-9.

9. Kim S., Park S., Chu W.W. An Index-Based Approach for Similarity Search Supporting Time Warping in Large Sequence Databases. // Proceedings of the 17th International Conference on Data Engineering, April 2-6, 2001, Heidelberg, Germany. – 2001. – P. 607–614. – URL: https://doi.org/10.1109/ICDE.2001.914875.

10. Lemire D. Faster Retrieval with a Two-pass Dynamic-time-warping Lower Bound. // Pattern Recogn. – 2009. – Vol. 42. – No. 9. – P. 2169–2180. – URL: http://dx.doi.org/10.1016/j.patcog.2008.11.030.

11. Movchan A., Zymbler M.L. Time Series Subsequence Similarity Search Under Dynamic Time Warping Distance on the Intel Many-core Accelerators. // Similarity Search and Applications - 8th International Conference, SISAP 2015, Glasgow, UK, October 12-14, 2015, Proceedings. – 2015. – P. 295–306. – URL: https://doi.org/10.1007/978-3-319-25087-8_28.

12. 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, 6-9 December 2009. – 2009. – P. 367–376. – URL: https://doi.org/10.1109/ICDM.2009.15.

13. 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: http://doi.acm.org/10.1145/2339530.2339576.

14. Sakoe H., Chiba S. Readings in Speech Recognition. / Ed. by A. Waibel, K.F. Lee. – San Francisco, CA, USA : Morgan Kaufmann Publishers Inc., 1990. – P. 159–165. – URL: http://dl.acm.org/citation.cfm?id=108235.108244.

15. Sart D. Accelerating Dynamic Time Warping Subsequence Search with GPUs and FPGAs. / D. Sart, A. Mueen, W.A. Najjar, E.J. Keogh, V. Niennattrakul. // ICDM 2010, The 10th IEEE International Conference on

Data Mining, Sydney, Australia, 14-17 December 2010. – 2010. – P. 1001–1006. – URL: https://doi.org/10.1109/ICDM.2010.21.

16. Shabib A. Parallelization of searching and mining time series data using Dynamic Time Warping. / A. Shabib, A. Narang, C.P. Niddodi, M. Das, R. Pradeep, et al. // 2015 International Conference on Advances in Computing, Communications and Informatics, ICACCI 2015, Kochi, India, August 10-13, 2015. – 2015. – P. 343–348. – URL: https://doi.org/10.1109/ICACCI.2015.7275633.

17. 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.

18. Srikanthan S., Kumar A., Gupta R. Implementing the dynamic time warping algorithm in multithreaded environments for real time and unsupervised pattern discovery. // 2011 2nd International Conference on Computer and Communication Technology, ICCCT 2011, Allahabad, India, September 15-17, 2011. – 2011. – P. 394–398. – URL: https://doi.org/10.1109/ICCCT.2011.6075111.

19. Stafford C.A., Walker G.P. Characterization and correlation of DC electrical penetration graph waveforms with feeding behavior of beet leafhopper, Circulifer tenellus. // Entomologia Experimentalis et Applicata. – Vol. 130. – No. 2. – P. 113–129. – URL: https://doi.org/10.1111/j.1570-7458.2008.00812.x.

20. Takahashi N. A Parallelized Data Stream Processing System Using Dynamic Time Warping Distance. / N. Takahashi, T. Yoshihisa, Y. Sakurai, M. Kanazawa. // 2009 International Conference on Complex, Intelligent and Software Intensive Systems, CISIS 2009, Fukuoka, Japan, March 16-19, 2009. – 2009. – P. 1100–1105. – URL: https://doi.org/10.1109/CISIS.2009.77.

21. Tarango J., Keogh E.J., Brisk P. Instruction set extensions for Dynamic Time Warping. // Proceedings of the International Conference on Hardware/Software Codesign and System Synthesis, CODES+ISSS 2013, Montreal, QC, Canada, September 29 - October 4, 2013. – 2013. – P. 18:1–18:10. – URL: https://doi.org/10.1109/CODES-ISSS.2013.6659005.

22. Wang Z. Accelerating subsequence similarity search based on

dynamic time warping distance with FPGA. / Z. Wang, S. Huang, L. Wang, H. Li, Y. Wang, et al. // The 2013 ACM/SIGDA International Symposium on Field Programmable Gate Arrays, FPGA '13, Monterey, CA, USA, February 11-13, 2013. – 2013. – P. 53–62. – URL: http://doi.acm.org/10.1145/2435264.2435277.

23. Xi X. Fast time series classification using numerosity reduction. / X. Xi, E.J. Keogh, C.R. Shelton, L. Wei, C.A. Ratanamahatana. // Machine Learning, Proceedings of the Twenty-Third International Conference (ICML 2006), Pittsburgh, Pennsylvania, USA, June 25-29, 2006. – 2006. – P. 1033–1040. – URL: http://doi.acm.org/10.1145/1143844.1143974.

24. Zhang Y., Adl K., Glass J.R. Fast spoken query detection using lower-bound Dynamic Time Warping on Graphical Processing Units. // 2012 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2012, Kyoto, Japan, March 25-30, 2012. – 2012. – P. 5173–5176. – URL: https://doi.org/10.1109/ICASSP.2012.6289085.

25. Zymbler M.L. Best-Match Time Series Subsequence Search on the Intel Many Integrated Core Architecture. // Advances in Databases and Information Systems - 19th East European Conference, ADBIS 2015, Poitiers, France, September 8-11, 2015, Proceedings. – 2015. – P. 275–286. – URL: https://doi.org/10.1007/978-3-319-23135-8_19.