МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«Южно-Уральский государственный университет»
(национальный исследовательский университет)
Факультет Вычислительной математики и информатики
Кафедра системного программирования

РАБОТА ПРОВЕРЕНА

Рецензент
к.ф.-м.н., профессор

_____ В.И. Заляпин

“___”_____ 2014 г.

ДОПУСТИТЬ К ЗАЩИТЕ

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

_____ Л.Б. Соколинский

“___”_____ 2014 г.

Разработка параллельного алгоритма поиска похожих подпоследовательностей временного ряда для сопроцессора Intel Xeon Phi

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
ЮУрГУ – 010400.62.2014.10-018-1887.ВКР

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

_____М.Л. Цымблер

Автор работы,
студент группы ВМИ-456

_____А.В. Мовчан

Ученый секретарь
(нормоконтролер)

_____ О.Н. Иванова

“___”_____ 2014 г.

Челябинск-2014

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«Южно-Уральский государственный университет»
(национальный исследовательский университет)
Факультет Вычислительной математики и информатики
Кафедра системного программирования

УТВЕРЖДАЮ
Зав. кафедрой СП
_____ Л.Б. Соколинский
08.02.2014

ЗАДАНИЕ
на выполнение выпускной квалификационной работы бакалавра
студенту группы ВМИ-456 Мовчану Александру Вячеславовичу,
обучающемуся по направлению 010400.62
«Информационные технологии»

1. Тема работы (утверждена приказом ректора от 12.03.2014 № 368)

Разработка параллельного алгоритма поиска похожих подпоследовательностей временного ряда для сопроцессора Intel Xeon Phi

2. Срок сдачи студентом законченной работы: 04.06.2014 г.

3. Исходные данные к работе

3.1. Fu T.-C. A Review on Time Series Data Mining // Engineering Applications of Artificial Intelligence, 2011. – Vol. 24. – No. 1. – P. 164 – 181.
3.2. Jeffers J., Reinders J. Intel Xeon Phi Coprocessor High Performance Programming. – USA: Morgan Kaufmann, 2013. – 409 p.
3.3. Rakthanmanon T. Searching and Mining Trillions of Time Series Subsequences under Dynamic Time Warping / T. Rakthanmanon, B. Campana, A. Mueen, G. Batista, B. Westover, et al. // The 18th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Beijing, China, 12-16 August, 2012. – ACM, 2012. – P. 262–270.

4. Перечень подлежащих разработке вопросов

4.1. Изучить существующие алгоритмы поиска похожих подпоследовательностей временного ряда.
4.2. Изучить методы разработки параллельных алгоритмов для сопроцессора Intel Xeon Phi.
4.3. Спроектировать и реализовать параллельный алгоритм поиска похожих

подпоследовательностей временного ряда для сопроцессора Intel Xeon Phi.
4.4. Провести эксперименты по анализу эффективности разработанного алгоритма.

5. Дата выдачи задания: 08.02.2014.

Научный руководитель

Доцент кафедры СП,
к.ф.-м.н. М.Л. Цымблер

Задание принял к исполнению А.В. Мовчан

Оглавление

Введение

Актуальность

Временной ряд (time series) представляет собой последовательность значений, измеряемых через равные промежутки времени [6]. Эти значения обычно являются действительными числами.

Временные ряды широко распространены во многих сферах человеческой деятельности: медицине [7], промышленности [11], сфере финансов [16], метеорологии [3] и др. Например, в медицине создается большое количество временных рядов, таких как электрокардиограммы, электроэнцефалограммы, данные по экспрессии генов и др. К интеллектуальному анализу временных рядов относятся такие задачи как индексация, кластеризация, классификация, прогнозирование, поиск аномалий, сегментация. Многие алгоритмы интеллектуального анализа временных рядов используют поиск похожих подпоследовательностей в качестве основной подпрограммы.

Поиск похожих подпоследовательностей (subsequence matching) [5]. Пусть данные представляют собой временной ряд $T = t_1, t_2, \ldots, t_M$, а запрос представлен временным рядом $Q = q_1, q_2, \ldots, q_N$. Необходимо найти K подпоследовательностей $T_{ij} = t_i, \ldots, t_j$ длины N, для которых минимально значение функции схожести $D(T_{ij}, Q)$. Поиск похожих подпоследовательностей проиллюстрирован на рис. 1.

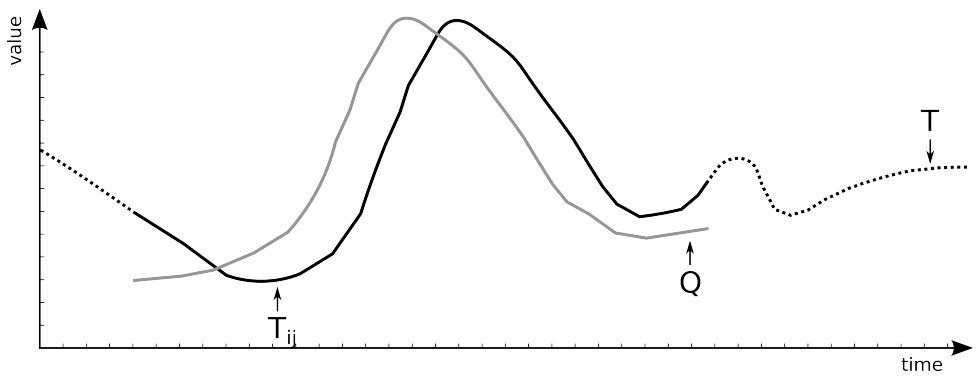


Рис. 1. Поиск похожих подпоследовательностей

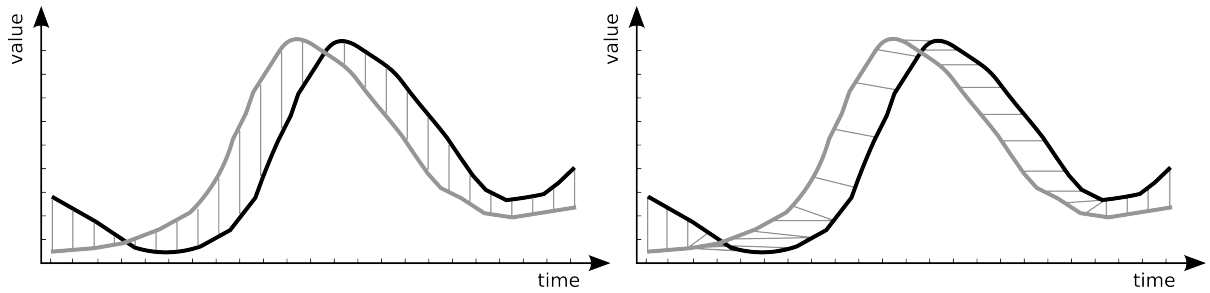
Схожесть временных рядов может быть определена с использованием различных методов [2]. Самым простым методом является использование *евклидовой метрики*. Схожесть с использованием евклидовой метрики вычисляется по формуле (1).

$$D(p,d)=\sqrt{\sum_{k=1}^{n}(p_k-q_k)^2}, \quad (1)$$

где p, d – временные ряды, длины n.

Соотнесение точек данных временных рядов с помощью евклидовой метрики приведено на рис. 2а.

Евклидову метрику, которая ставит в соответствие каждому элементу одного временного ряда один из элементов другого ряда, можно рассматривать как частный случай *алгоритма динамической трансформации шкалы времени (Dynamic Time Warping, DTW)* [14], который позволяет сопоставить один элемент первого ряда нескольким элементам другого временного ряда. DTW вычисляет оптимальную последовательность трансформации времени между двумя временными рядами, что позволяет сравнивать временные ряды, которые различаются скоростью изменения данных. Соотнесение точек данных временных рядов с помощью динамической трансформации шкалы времени приведено на рис. 2б.



а) с помощью евклидова расстояния б) с помощью динамической трансформации шкалы времени

Рис. 2. Соотнесение точек временных рядов

Рассмотрим задачу определения схожести временных рядов с исполь-

зованием алгоритма динамической трансформации шкалы времени. Пусть имеются два временных ряда $X = x_1, x_2, ..., x_i, ..., x_N$ и $Y = y_1, y_2, ..., y_i, ..., y_N$. Задача заключается в поиске пути трансформации $W = w_1, w_2, ..., w_i, ..., w_K$, причем соблюдаются следующие условия:

$$w_1=(1,1), w_K=(N,N), w_k=(i,j), w_{k+1}=(i',j'), i\le i'\le i+1, j\le j'\le j+1, \quad (2)$$

$$DTW(X,Y)=\sum_{k=1}^{k=K} d(w_{ki}, w_{kj})\to min, \quad (3)$$

где i, i' – индексы временного ряда X;

j, j' – индексы временного ряда Y.

Поиск похожих подпоследовательностей применяется во многих сферах. Например, можно определить, когда продажи были такими же как сегодня. Или можно применить данный метод для распознавания рукописного текста. Кроме широких возможностей применения поиска похожих подпоследовательностей напрямую, можно использовать его как подпрограмму для других алгоритмов анализа данных.

С другой стороны, в настоящее время развивается использование гетерогенных архитектур, совместно использующих центральный процессор и сопроцессоры (или графические ускорители). Данный метод позволяет значительно увеличить скорость вычислений. Многоядерный сопроцессор Intel Xeon Phi позволяет увеличить производительность узла для высокопараллельных задач. Особенностью сопроцессора Intel Xeon Phi является сочетание высокой производительности и возможности использовать стандартные программные инструменты и методы для разработки программных продуктов. Сопроцессор Intel Xeon Phi применяется для решения задач в различных областях науки [18, 21].

В работе [17] рассматривается ускорение поиска похожих подпоследовательностей с помощью GPU и FPGA, однако в данной работе используется алгоритм перебора без использования оптимизаций. В то же время,

отсутствуют научные работы, посвященные реализации задачи поиска похожих подпоследовательностей для сопроцессора Intel Xeon Phi.

Таким образом, на сегодняшний день является актуальной задача разработки параллельного алгоритма поиска похожих подпоследовательностей для сопроцессора Intel Xeon Phi.

Цель и задачи исследования

Целью данной выпускной квалификационной работы является разработка параллельного алгоритма поиска похожих подпоследовательностей временного ряда для сопроцессора Intel Xeon Phi.

Для достижения поставленной цели необходимо решить следующие *задачи*:

1) выбрать и реализовать последовательный алгоритм поиска похожих подпоследовательностей временного ряда;

2) спроектировать параллельный алгоритм на основе технологии OpenMP;

3) адаптировать параллельный алгоритм для сопроцессора Intel Xeon Phi;

4) провести анализ эффективности разработанного алгоритма.

Структура и объем работы

Выпускная квалификационная работа состоит из введения, четырех разделов, заключения, библиографии. Объем работы составляет 30 страниц, объем библиографии – 21 наименование.

Содержание работы

Первый раздел, «Методы поиска похожих подпоследовательностей», содержит обзор работ по тематике исследования и описание последовательного алгоритма поиска похожих подпоследовательностей временного ряда.

Второй раздел, «Архитектура сопроцессора Intel Xeon Phi», содер-

жит краткое описание архитектурных особенностей сопроцессора Intel Xeon Phi и методов разработки для сопроцессора.

Третий раздел, «Проектирование и реализация параллельного алгоритма», содержит описание параллельного алгоритма на основе технологии OpenMP и параллельного алгоритма для сопроцессора Intel Xeon Phi.

В четвертом разделе, «Вычислительные эксперименты», представлены результаты экспериментов и анализ эффективности разработанных алгоритмов.

В заключении подводятся основные итоги проделанной работы.

1. Методы поиска похожих подпоследовательностей

1.1. Обзор работ по тематике исследования

Рассмотрим существующие алгоритмы поиска похожих подпоследовательностей временного ряда. Одной из первых работ, посвященных данной задаче, является работа [5]. В этой работе временной ряд рассматривается как набор многомерных прямоугольников, по которым строится пространственный индекс, а поиск подпоследовательностей заключается в выполнении пространственных запросов к этому индексу. Основываясь на этой работе, другие исследователи улучшили производительность поиска похожих подпоследовательностей. Например, алгоритм DualMatch, описанный в работе [12], заключается в разделении временного ряда на непересекающиеся окна и разделении запроса на скользящие окна для составления индекса. Эти алгоритмы, как и многие другие, используют дискретное преобразование Фурье в качестве оценки евклидовой метрики и используют построение индекса для поиска похожих подпоследовательностей.

Алгоритмы поиска похожих подпоследовательностей, в которых схожесть определяется DTW, используют суффиксные деревья для построения индекса [10], индексы на основе префикса запросов [13], оптимизации последовательного сравнения всех возможных подпоследовательностей [15] и др. Подход, основанный на оптимизациях, менее требователен к памяти, что позволяет осуществлять поиск в очень больших временных рядах, а использование оптимизаций позволяет добиться высокой производительности. Например, алгоритм, описанный в работе [15] и представляющий собой точный последовательный поиск с оптимизациями, позволяет осуществлять поиск во временных рядах длиной в триллион точек данных за приемлемое время.

В работе [19] рассматривается создание многопоточного алгоритма поиска похожих подпоследовательностей. Для определения схожести вре-

менных рядов данный алгоритм использует DTW.

В работе [17] рассматривается применение GPU и FPGA для ускорения поиска похожих подпоследовательностей. Однако в данной работе используется алгоритм перебора без использования оптимизаций.

Подробный обзор алгоритмов поиска похожих подпоследовательностей временного ряда приведен в работе [6].

1.2. Последовательный алгоритм поиска похожих подпоследовательностей временного ряда

Для распараллеливания был выбран алгоритм, представленный в работе [15], по следующим причинам:

- данный алгоритм позволяет осуществлять поиск похожих подпоследовательностей в больших временных рядах, так как он не использует индекс для поиска данных;
- алгоритм обладает высокой степенью параллелизма по данным;
- алгоритм обладает высоким быстродействием по сравнению с аналогичными алгоритмами.

Данный алгоритм представляет собой алгоритм последовательного поиска, то есть заключается в переборе всех возможных подпоследовательностей. В качестве метрики используется динамическая трансформация шкалы времени. Перед сравнением к временным рядам применяется Z-нормализация. *Z-нормализация* заключается в преобразовании входного вектора в выходной вектор, для которого среднее арифметическое приблизительно равно 0, а среднеквадратичное отклонение близко к 1 [8]. Z-нормализация вычисляется по формуле (4).

$$x_i' = \frac{x_i - \mu}{\sigma}, i \in N, \quad (4)$$

где μ – среднее арифметическое;

σ – среднеквадратичное отклонение.

Для ускорения расчетов в данном алгоритме применяются различные оптимизации. Рассмотрим подробно эти оптимизации.

1. Использование квадрата расстояния.

Алгоритм DTW использует операцию вычисления квадратного корня при расчете расстояния. Оптимизация заключается в том, что при вычислении расстояния не вычисляется квадратный корень, а используется квадрат расстояния.

2. Использование оценки снизу.

Классическим приемом, позволяющим ускорить последовательный поиск, использующий такую сложную меру расстояния как DTW, является использование вычислительно простой оценки снизу, позволяющей отбросить бесперспективные варианты. Существуют различные алгоритмы вычисления оценок снизу. Одной из таких оценок является оценка LB_{Kim}, которая вычисляется как квадрат расстояния между первой и последней парой точек, максимумами и минимумами двух последовательностей. Но так как для нормализованных рядов расстояние между максимумами и минимумами довольно мало, то в качестве оценки используется только квадрат расстояния между первой и последней парой точек, что позволяет снизить сложность с O(n) до O(1). Назовем эту упрощенную оценку $LB_{Kim}FL$.

3. Заблаговременная отмена расчета LB_{Keogh}.

Другая оценка снизу LB_{Keogh} является более точной, но в то же время более сложной. Для вычисления этой оценки для запроса Q строятся две новые последовательности: U и L. Эти последовательности основываются на ограничении DTW, которое называется Sakoe-Chiba Band. Оно ограничивает путь трансформации в алгоритме DTW, не позволяя пути трансформации уходить больше, чем на R элементов от диагональной линии матрицы трансформации. Используя данное ограничение, можно построить две последовательности, которые будут описывать оболочку для запроса. По-

следовательности U и L вычисляются по формулам (5) и (6):

$$U_i = max(Q_{i-R} : Q_{i+R}), \quad (5)$$

$$L_i = max(Q_{i-R} : Q_{i+R}). \quad (6)$$

Дальше оценка LB_{Keogh} определяется по формуле (7):

$$LB_{keogh}(Q, C) = \sqrt{\sum_{i=1}^{n} \begin{cases} (C_i - U_i)^2, \text{если } C_i > U_i \\ (C_i - L_i)^2, \text{если } C_i < L_i \\ 0, \text{в остальных случаях} \end{cases}}. \quad (7)$$

Оптимизация заключается в том, что если при расчете оценка превысит лучшее на данный момент расстояние, то можно остановить вычисление и отбросить рассматриваемую подпоследовательность.

4. Заблаговременная отмена расчета DTW.

Вычисляя DTW последовательно от 1 до K, можно посчитать сумму частичного значения DTW и значения LB_{Keogh} от K+1 до n. Если сумма $DTW(Q_{1:K}, C_{1:K}) + LB_{Keogh}(Q_{K+1:n}, C_{K+1:n})$ превышает лучшее на данный момент расстояние, то можно отбросить рассматриваемую подпоследовательность.

5. Заблаговременная отмена расчета Z-нормализации.

Данная оптимизация заключается в том, что Z-нормализация проводится одновременно с расчетом оценки LB_{Keogh}, и, если заблаговременно остановится расчет оценки, то остановится и расчет Z-нормализации.

6. Переупорядочивание расчета для заблаговременной отмены.

Для ранней отмены расчета оценки LB_{Keogh} применяется переупорядочивание. Порядок расчета определяется индексами запроса, которые отсортированы по абсолютным Z-нормализованным значениям запроса.

7. Обмен ролей запроса и данных в LB_{Keogh}.

В оценке LB_{Keogh}, рассмотренной ранее, оболочка (последовательности U и L) строится для запроса. Это позволяет вычислить оболочку один раз и затем использовать для всех подпоследовательностей. Однако, можно

построить оболочку для подпоследовательности и посчитать оценку новым способом, если оценка LB_{Keogh} не отбросила данную подпоследовательность.

8. Каскадное применение оценок.

Каскадное применение оценок заключается в последовательном применении различных оценок. Сначала применяется самая простая оценка $LB_{Kim}FL$. Если подпоследовательность не была отброшена этой оценкой, то вычисляем оценку LB_{Keogh}, где оболочка строится для запроса. Затем снова применяется LB_{Keogh}, но оболочка строится по подпоследовательности. Если оценки не смогли отбросить подпоследовательность, то выполняется расчет DTW.

Использование этих оптимизаций позволяет достичь высокой производительности, но можно ускорить расчет, если обрабатывать временной ряд параллельно. Далее будет рассмотрено ускорение данного алгоритма с помощью сопроцессора Intel Xeon Phi.

2. Архитектура сопроцессора Intel Xeon Phi

Intel Xeon Phi представляет собой сопроцессор, основанный на архитектуре Intel Many Integrated Core (Intel MIC) [4]. Intel Xeon Phi может содержать до 61 ядра, соединенных высокопроизводительной шиной. Архитектура сопроцессора представлена на рис. 3.

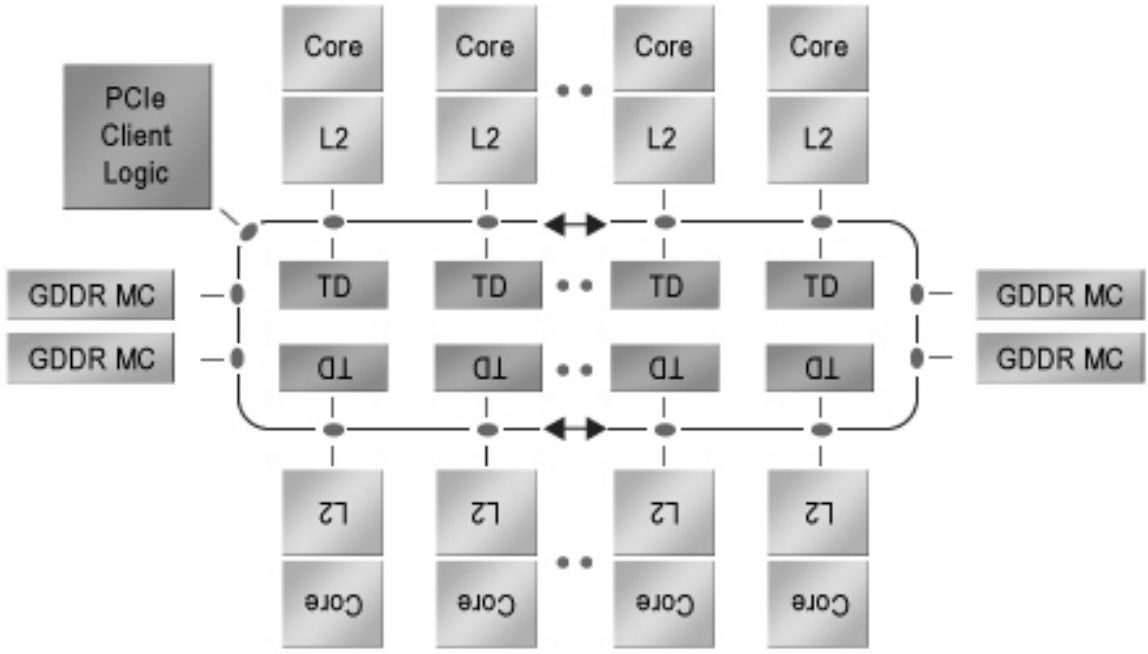


Рис. 3. Архитектура сопроцессора Intel Xeon Phi

Сопроцессор имеет собственную операционную систему на базе ОС Linux и поддерживает все важные инструменты разработки, такие как компиляторы C/C++ и Fortran, MPI и OpenMP, высокопроизводительные библиотеки, например, MKL, средства для отладки и профилирования. Сопроцессор соединяется с центральным процессором Intel Xeon через шину PCI Express.

Особенностью сопроцессора Intel Xeon Phi является сочетание высокой производительности и возможности использовать стандартные программные инструменты и методы для разработки программных продуктов. Применение стандартных программных инструментов и методов позволяет

использовать существующий программный код, предназначенный для процессоров Intel Xeon, с минимальными изменениями.

Каждое ядро сопроцессора имеет 4 потока за счет технологии Hyper-Threading и очень широкие векторные АЛУ (512-разрядные SIMD), что обеспечивает до 16 операций над float или до 8 операций над double в инструкции.

Существует три режима параллельной обработки:

1) *выполнение на сопроцессоре (Native Execution)*. Для выполнения на сопроцессоре приложение компилируется со специальным флагом -mmic для генерации кода под архитектуру сопроцессора. Затем бинарный файл может быть скопирован и запущен на сопроцессоре;

2) *режим выгрузки (Offload)*. Данный режим заключается в выполнении программы на центральном процессоре с выгрузкой некоторых участков кода для выполнения на сопроцессоре. Режим доступен при использовании технологий OpenMP, Intel Cilk, Intel TBB, а также при использовании библиотеки MKL;

3) *симметричный режим (Symmetric Mode)*. Данный режим используется при применении технологии MPI.

3. Проектирование и реализация параллельного алгоритма

Чтобы разработать параллельный алгоритм для сопроцессора Intel Xeon Phi, необходимо было сначала реализовать последовательный алгоритм, описанный ранее, затем разработать параллельную реализацию алгоритма на основе технологии OpenMP, а после адаптировать параллельный алгоритм для сопроцессора.

Авторы последовательного алгоритма предоставили исходные коды [20], так что реализовывать последовательный алгоритм не было необходимости. Эти исходные коды использовались в качестве основы для параллельного алгоритма.

3.1. Распараллеливание алгоритма на основе технологии OpenMP

Далее мы рассмотрим параллельный алгоритм, основанный на технологии OpenMP. Параллельный алгоритм использует параллелизм по данным, разбивая временной ряд на равные участки, для обработки каждого из них отдельной OpenMP-нитью.

Диаграмма деятельности параллельного алгоритма приведена на рис. 4. UCR-DTW представляет собой подпрограмму, реализующую последовательный алгоритм, описанный в разделе 1.

В подпрограмме UCR-DTW используется общая (shared) переменная, которая хранит информацию о расстоянии до ближайшей подпоследовательности. Эта информация позволяет отбрасывать неподходящие подпоследовательности, используя оценки, и реже считать расстояние с использованием DTW. А так как переменная является общей для всех потоков, то можно считать, что у всех потоков имеется актуальная информация о расстоянии до ближайшей подпоследовательности.

Кроме того, предложенный алгоритм позволяет считывать новые данные из файла одновременно с обработкой уже считанных данных. Это позволяет уменьшить последовательную часть программы.

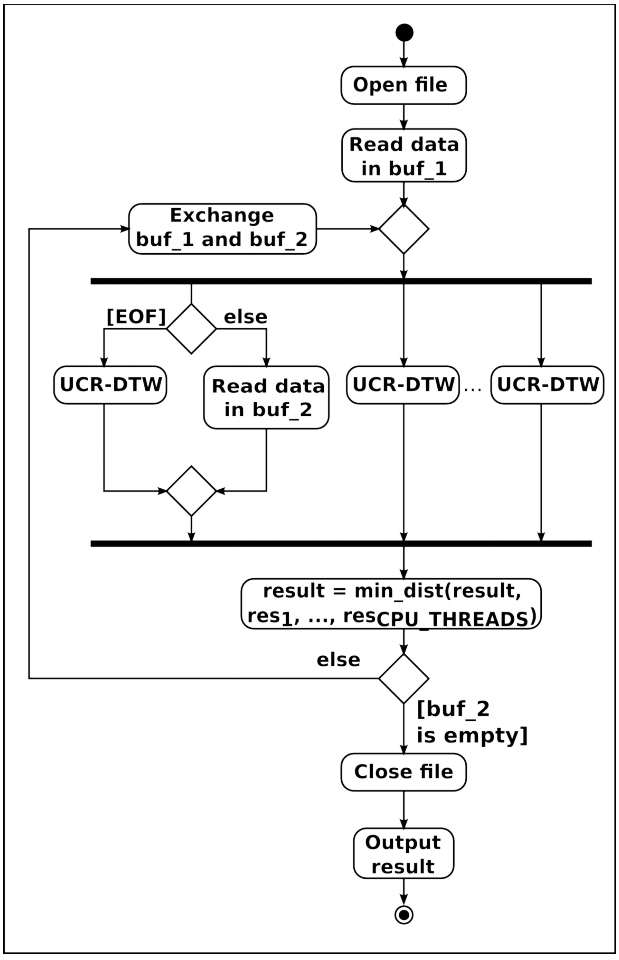


Рис. 4. Диаграмма деятельности параллельного алгоритма

3.2. Адаптация параллельного алгоритма для сопроцессора Intel Xeon Phi

В ходе проведения исследования было разработано две версии алгоритма. Первую версию параллельного алгоритма для сопроцессора будем называть наивным параллельным алгоритмом для сопроцессора, так как его смысл заключается в перенесении некоторой части работы с процессора на сопроцессор, что является тривиальной, но, как оказалось, неэффективной реализацией. Подробное описание данного алгоритма приведено в разделе 3.2.2. Другой параллельный алгоритм для сопроцессора использует более сложный алгоритм. Он описан в разделе 3.2.1.

Для организации взаимодействия между процессором и сопроцессо-

ром используется режим выгрузки (offload mode). В данном режиме программа выполняется на центральном процессоре, а некоторый код и данные выгружаются для выполнения на сопроцессоре.

3.2.1. Параллельный алгоритм для сопроцессора Intel Xeon Phi

Рассмотрим, как работает параллельный алгоритм для сопроцессора Intel Xeon Phi. Диаграмма деятельности данного алгоритма приведена на рис. 5.

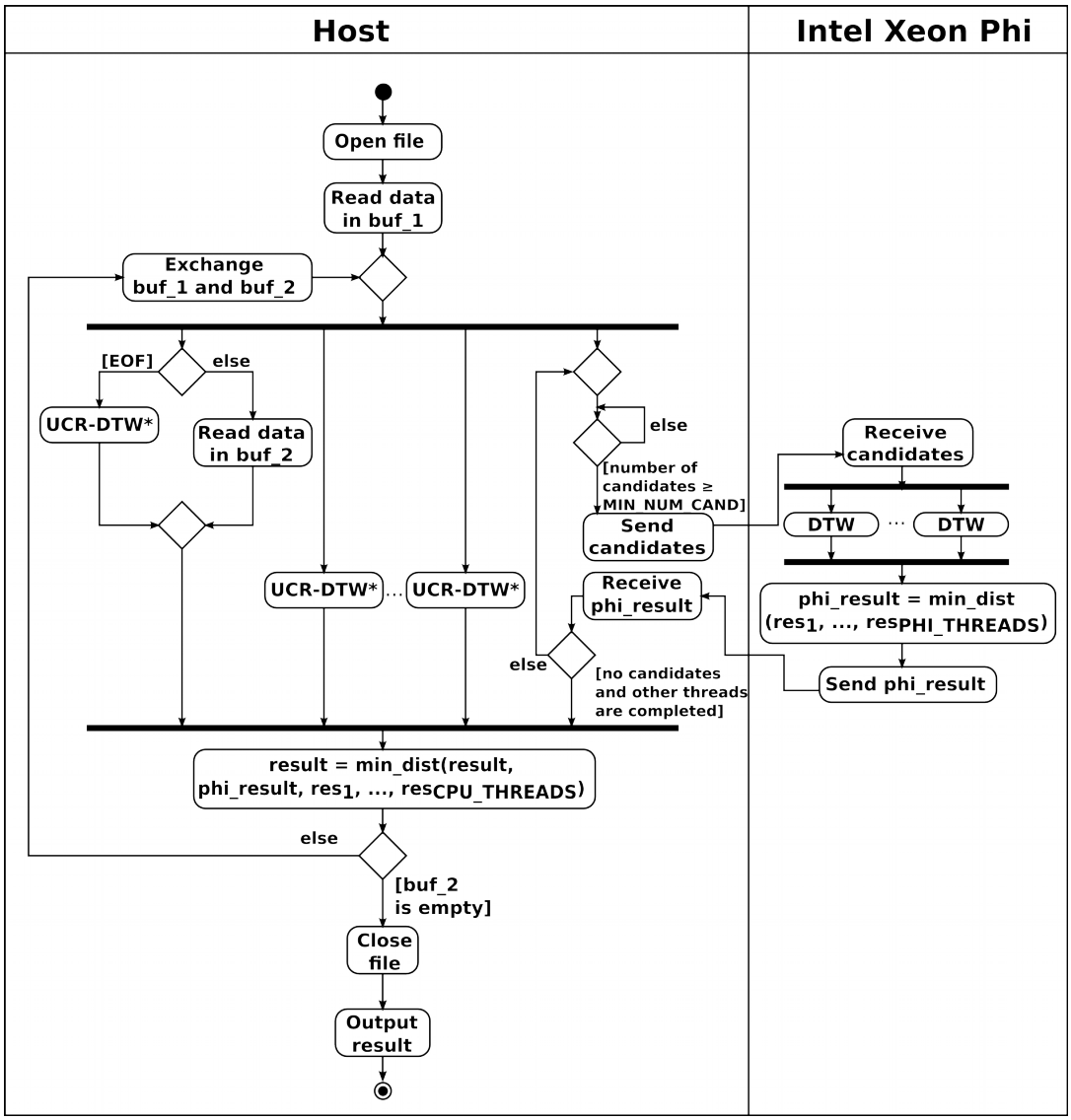


Рис. 5. Диаграмма деятельности параллельного алгоритма для сопроцессора Intel Xeon Phi

Данный алгоритм основан на алгоритме, рассмотренном в предыдущем разделе. Одна из нитей процессора управляет работой сопроцессора, а остальные нити центрального процессора выполняют подпрограмму UCR-DTW* для выделенного им участка временного ряда. Подпрограмма UCR-DTW* параллельного алгоритма для сопроцессора является модифицированной версией подпрограммы UCR-DTW параллельного алгоритма для процессора. Подпрограмма UCR-DTW* также реализует последовательный алгоритм, но при необходимости произвести расчет DTW либо производит расчет, либо добавляет подпоследовательность в очередь для последующего вычисления DTW на сопроцессоре.

Поток, управляющий работой сопроцессора, следит за количеством подпоследовательностей в очереди. Если количество подпоследовательностей превышает MIN_NUM_CAND, то MIN_NUM_CAND подпоследовательностей из очереди отправляются на сопроцессор, где для каждой подпоследовательности выполняется расчет DTW. Постоянная MIN_NUM_CAND должна быть кратна количеству рабочих потоков сопроцессора.

Рассмотрим работу подпрограммы UCR-DTW*. Диаграмма деятельности приведена на рис. 6. В данной подпрограмме осуществляется перебор всех возможных подпоследовательностей, длина которых равна длине запроса. Для каждой подпоследовательности производится каскадное вычисление оценок. Если одна из оценок превышает bsf (расстояние до самой похожей подпоследовательности), то подпоследовательность отбрасывается. Если ни одна из оценок не отбросила подпоследовательность, то проверяется количество подпоследовательностей в очереди. Если количество подпоследовательностей превышает MAX_NUM_CAND, то вычисление DTW выполняется на процессоре, иначе подпоследовательность добавляется в очередь для последующего вычисления на сопроцессоре.

Кроме информации о подпоследовательности, на сопроцессор пересылается массив, содержащий информацию об оценке LB_{Keogh}. Эта информация используется для заблаговременного отмена расчета DTW. Подробнее об оптимизации написано в разделе 1.2.

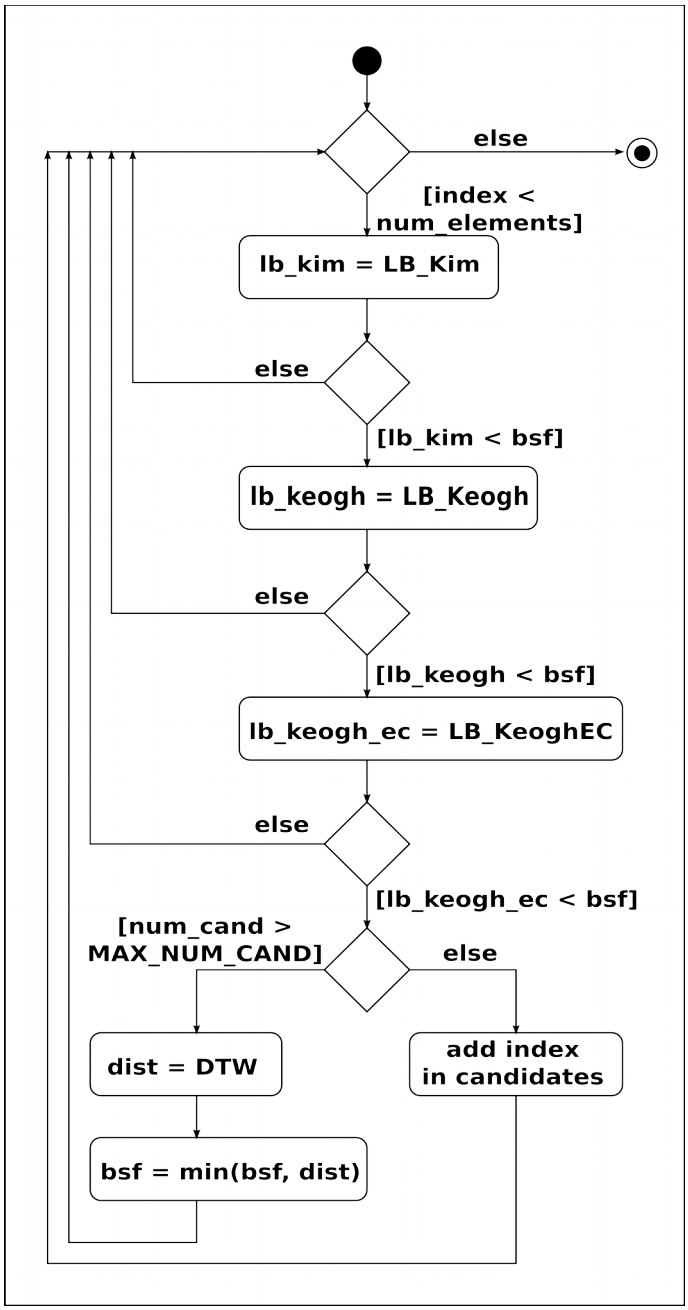


Рис. 6. Диаграмма деятельности подпрограммы UCR-DTW*

Важным инструментом ускорения вычислений для сопроцессора яв-

ляется векторизация. Векторизация позволяет выполнять набор однотипных операций над несколькими элементами массива одновременно. Поэтому очень важно применять векторизацию во всех циклах, в которых это возможно. В связи с этим функция вычисления DTW была модифицирована так, чтобы максимизировать использование векторных инструкций. Для этого внутренний цикл функции DTW был разделен на два цикла, один из который векторизуется. Это позволило увеличить скорость вычислений функции DTW.

3.2.2. Наивный параллельный алгоритм для сопроцессора Intel Xeon Phi

В ходе работы был спроектирован и реализован еще один параллельный алгоритм для сопроцессора. Он был разработан раньше, чем алгоритм, предложенный выше. Будем называть данный алгоритм *наивным параллельным алгоритмом для сопроцессора*, так как его смысл заключается в перенесении некоторой части работы с процессора на сопроцессор. Работа данного алгоритма заключается в следующем. Часть данных буфера, равная константе ALPHA($0 < ALPHA \leq 1$), пересылается для обработки на сопроцессор. После загрузки данных в память сопроцессора осуществляется параллельная обработка данных временного ряда. Затем результаты отправляются в главный поток, который выбирает самые лучшие результаты. Остальные данные буфера обрабатываются центральным процессором. Одновременно с обработкой данных один из потоков центрального процессора загружает из файла следующую часть данных в другой буфер. Схема работы параллельного алгоритма изображена на рис. 7.

Но данный алгоритм имеет плохую производительность. Плохая производительность возникает из-за отсутствия возможности постоянной синхронизации общей переменной между процессором и сопроцессором. Синхронизация общей переменной происходит только при входе и выходе

из секции выполнения кода сопроцессора. Таким образом, невозможно постоянно синхронизировать переменную total_bsf между процессором и сопроцессором. Из-за чего намного меньше подпоследовательностей отбрасываются с помощью оценок.

Сравнение производительности этих алгоритмов приведено в разделе «Вычислительные эксперименты».

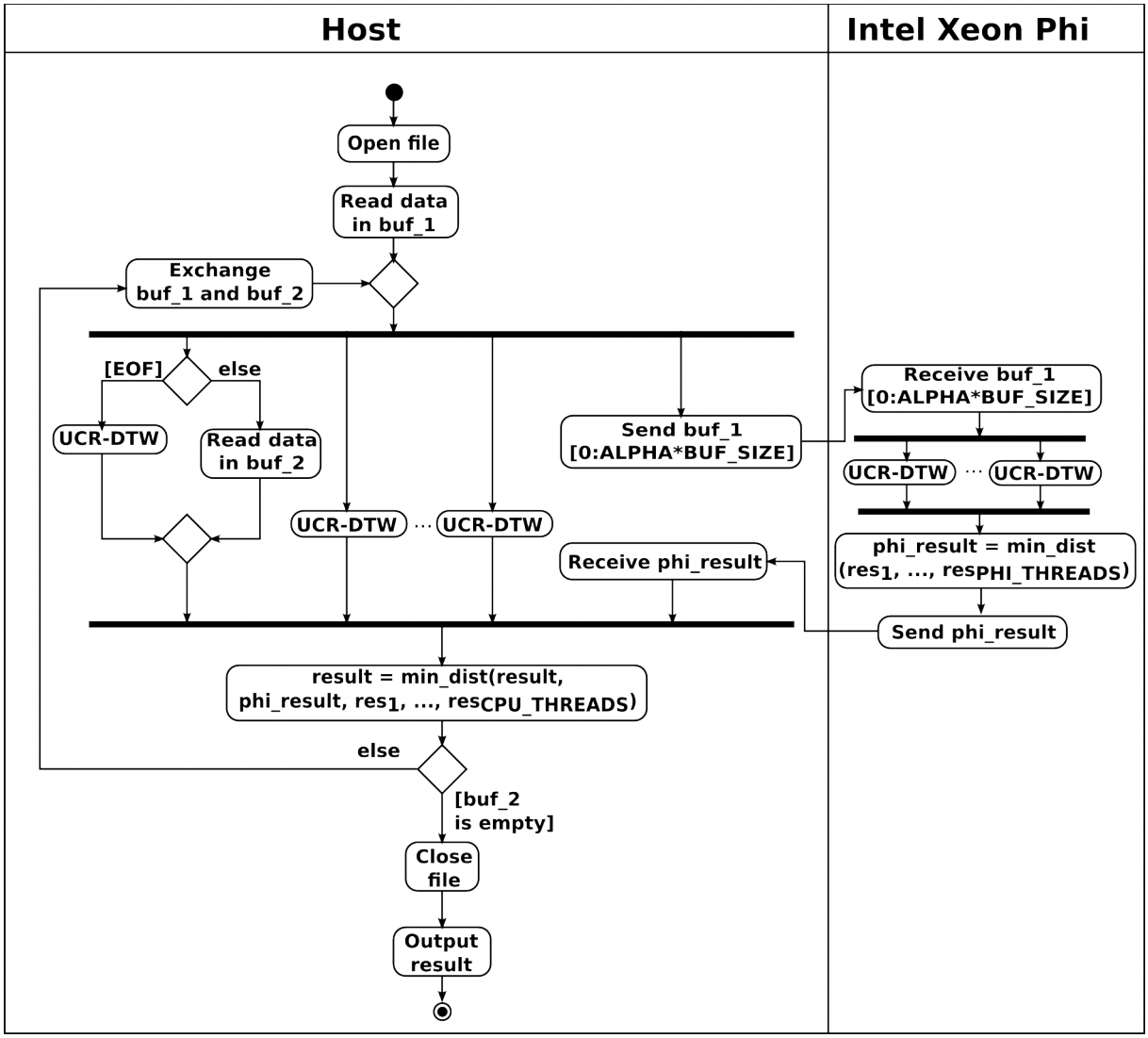


Рис. 7. Диаграмма деятельности наивного параллельного алгоритма для сопроцессора

4. Вычислительные эксперименты

В данном разделе приведены результаты вычислительных экспериментов по исследованию эффективности разработанных алгоритмов. Эксперименты исследуют изменение времени выполнения в зависимости от алгоритма и длины запроса.

Вычислительные эксперименты проводились на узле суперкомпьютера «Торнадо». Характеристики узла приведены в табл. 1.

Табл. 1. Аппаратные характеристики вычислительного узла суперкомпьютера «Торнадо ЮУрГУ»

Характеристики	Процессор	Сопроцессор
Модель	Intel Xeon X5680	Intel Xeon Phi SE10X
Количество ядер	6	61
Частота ядер, GHz	3.33	1.1
Количество потоков на ядро	2	4
Производительность, Тфлопс	0.371	1.076

Эксперименты проводились на синтетических данных, полученных с использованием *модели случайных блужданий (random walk)*. Случайное блуждание – математическая модель процесса случайных изменений – шагов в дискретные моменты времени. При этом предполагается, что изменение на каждом шаге не зависит от предыдущих и от времени. Чтобы сформировать одномерный временной ряд мы будем использовать одномерное дискретное случайное блуждание, которое начинается с 0 и на каждом шаге увеличивается или уменьшается на определенное значение с одинаковой вероятностью. С помощью модели случайных блужданий можно сформировать временной ряд любой длины. Для проведения тестов используется оригинальная реализация алгоритма.

На рис. 8 приведены результаты экспериментов по исследованию эф-

фективности разработанных алгоритмов. Длина временного ряда, используемого в данных экспериментах, составляет 100 000 000 точек данных. Время выполнения приведено без учета времени загрузки данных в память. На графике видно, что параллельный алгоритм для сопроцессора более эффективен при большой длине запроса. При небольшой длине запроса время выполнения параллельного алгоритма для сопроцессора практически равно скорости выполнения параллельного алгоритма для процессора. Но в любом случае, параллельные алгоритмы показывают большое ускорение по сравнению с последовательным алгоритмом. На графике также видно, что первая версия параллельного алгоритма для сопроцессора показывает плохую производительность.

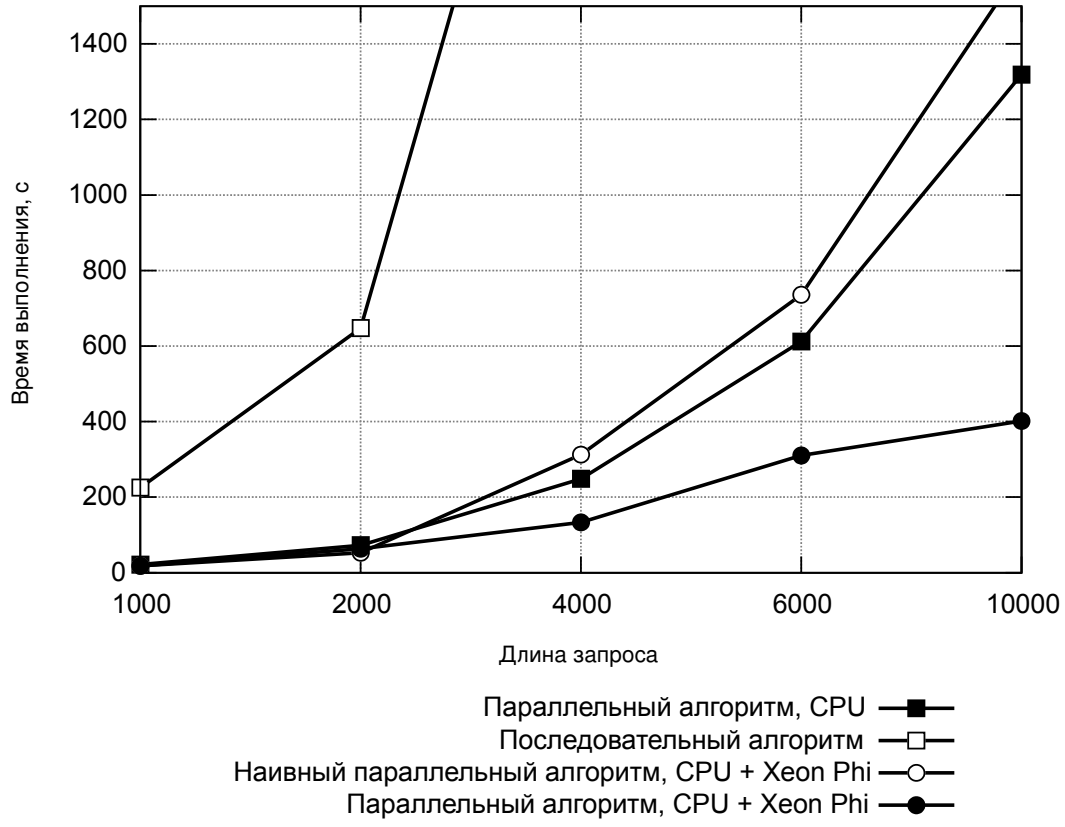


Рис. 8. Сравнение производительности разработанных алгоритмов на синтетических данных

Также были проведены эксперименты на реальных данных. В качестве таких данных использовались данные электрокардиограммы человека. Длина временного ряда, используемого в этих экспериментах, составляет 20 040 000 точек данных. Такая длина соответствует примерно 22 часам при частоте дискретизации 250 Гц. Результаты экспериментов приведены на рис. 9.

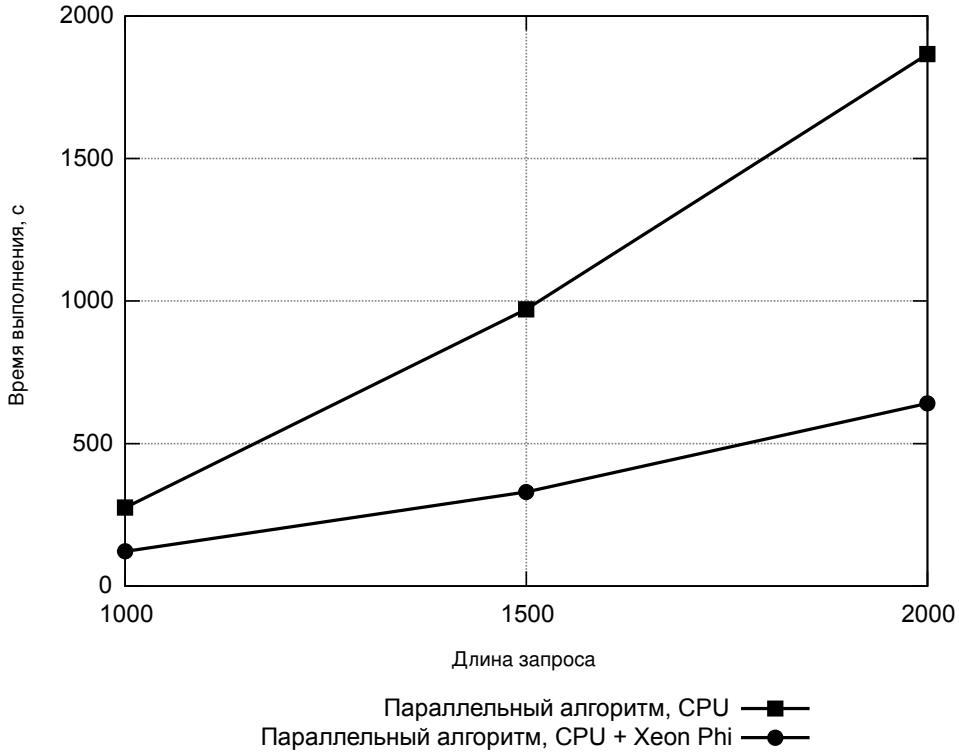


Рис. 9. Сравнение производительности разработанных алгоритмов на реальных данных

Эксперименты показали, что использование сопроцессора Intel Xeon Phi оправдано при большой длине запроса, но в то же время применение параллельного алгоритма в любом случае позволяет уменьшить время выполнения по сравнению с последовательным алгоритмом.

Таким образом, эксперименты доказывают эффективность параллельного алгоритма поиска похожих подпоследовательностей для сопроцессора Intel Xeon Phi.

Заключение

Данная работа посвящена разработке параллельного алгоритма поиска похожих подпоследовательностей для сопроцессора Intel Xeon Phi. В ходе выполнения исследования были получены следующие основные результаты:

1) изучены существующие алгоритмы поиска похожих подпоследовательностей временного ряда;

2) изучены методы разработки параллельных алгоритмов для сопроцессора Intel Xeon Phi;

3) разработан параллельный алгоритм поиска похожих подпоследовательностей на основе технологии OpenMP;

4) разработан параллельный алгоритм поиска похожих подпоследовательностей для сопроцессора Intel Xeon Phi;

5) проведены эксперименты, подтвердившие эффективность разработанного алгоритма;

6) по теме исследования опубликована работа [1] и сделаны доклады на международной конференции ПаВТ’2014 и 67-ой студенческой научной конференции ЮУрГУ.

Литература

1. Мовчан А.В., Цымблер М.Л. Разработка параллельного алгоритма поиска похожих подпоследовательностей временного ряда для сопроцессора Intel Xeon Phi // Параллельные вычислительные технологии (ПаВТ'2014): труды международной научной конференции (1-3 апреля 2014 г., г. Ростов-на-Дону). – Челябинск: Издательский центр ЮУрГУ, 2014. – С. 372.

2. Ding H. Querying and mining of time series data: experimental comparison of representations and distance measures./ H. Ding, G. Trajcevski, P. Scheuermann, X. Wang, E. Keogh // Proceedings of the VLDB Endowment, 2008. – Vol. 1. – No. 2. – P. 1542–1552.

3. Duchon C., Hale R. Time Series Analysis in Meteorology and Climatology: An Introduction. – USA: Wiley, 2012. – 262 p.

4. Duran A., Klemm M. The Intel® Many Integrated Core Architecture // High Performance Computing and Simulation (HPCS), Madrid, Spain, 2-6 July, 2012. IEEE, 2012. P. 365–366.

5. Faloutsos C., Ranganathan M., Manolopoulos Y. Fast subsequence matching in time-series databases.// The 1994 ACM SIGMOD international conference on Management of data, New York, NY, 24-27 May, 1994. ACM, 1994. – P. 419–429.

6. Fu T.-C. A Review on Time Series Data Mining // Engineering Applications of Artificial Intelligence, 2011. – Vol. 24. – No. 1. – P. 164–181.

7. Funk P., Xiong N. Case Based Reasoning and Knowledge Discovery in Medical Applications with Time Series // Computational Intelligence, 2006. Vol. 22. – No. 3–4. – P. 238–253.

8. Goldin D., Kanellakis P. On Similarity Queries for Time-Series Data: Constraint Specification and Implementation // The First International Conference on Principles and Practice of Constraint Programming, Cassis, France, 19-

22 September, 1995. – Berlin: Springer-Verlag, 1995. – P. 137–153.

9. Jeffers J., Reinders J. Intel Xeon Phi Coprocessor High Performance Programming. – USA: Morgan Kaufmann, 2013. – 409 p.

10. Kim S.W. Shape-based retrieval of similar subsequences in time-series databases / S.W. Kim, J. Yoon, S. Park, T.H. Kim // The 2002 ACM symposium on Applied computing, Madrid, Spain, 10-14 March, 2002. – ACM, 2002. – P. 438–445.

11. Liu H. Electricity Consumption Time Series Profiling: A Data Mining Application in Energy Industry / H. Liu, Z. Yao, T. Eklund, B. Back // The 12th Industrial conference on Advances in Data Mining: applications and theoretical aspects, Berlin, Germany, 13-20 July, 2012. – Berlin: Springer-Verlag, 2012. – P. 52–66.

12. Moon Y.-S., Whang K.-Y., Loh W.-K. Duality-Based Subsequence Matching in Time-Series Databases // The 17th International Conference on Data Engineering, Washington, DC, USA, 2-6 April, 2001. – IEEE Computer Society, 2001. – P. 263–272.

13. Park S. Prefix-querying: an approach for effective subsequence matching under time warping in sequence databases / S. Park, S.W. Kim, J.S. Cho, S. Padmanabhan // The tenth international conference on Information and knowledge management, Atlanta, Georgia, USA, 5-10 November, 2001. – ACM, 2001. – P. 255–262.

14. Senin P. Dynamic Time Warping Algorithm Review. [Электронный ресурс] URL: http://www2.hawaii.edu/~senin/assets/papers/DTW-review2008-draft.pdf (дата обращения: 31.05.2014).

15. Rakthanmanon T. Searching and Mining Trillions of Time Series Subsequences under Dynamic Time Warping / T. Rakthanmanon, B. Campana, A. Mueen, G. Batista, B. Westover, et al. // The 18th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Beijing, China, 12-16 August,

2012. – ACM, 2012. – P. 262–270.

16. Ruey S. Tsay Analysis of Financial Time Series. – USA: John Wiley & Sons, 2005. – 605 p.

17. Sart D. Accelerating Dynamic Time Warping Subsequence Search with GPUs and FPGAs / D. Sart, A. Mueen, W. Najjar, E. Keogh, V. Niennattrakul // The 10th IEEE International Conference on Data Mining, Sydney, NSW, Australia, 13-17 December, 2010. – IEEE, 2010. – P. 1001–1006.

18. Sinha P. Biosequence Analysis Using Intel® Xeon Phi / P. Sinha, G. Misra, D. Vikranman, A. Das // Seventh UKSim/AMSS European Modelling Symposium, Manchester, UK, 20-22 November, 2013. – IEEE, 2013. – P. 497–499.

19. Srikanthan S., Kumar A., Gupta R. Implementing the dynamic time warping algorithm in multithreaded environments for real time and unsupervised pattern discovery // Computer and Communication Technology (ICCCT), Allahabad, India, 15-17 September, 2011. – IEEE Computer Society, 2011. – P. 394–398.

20. UCR Suite for Time Series Subsequence Search. [Электронный ресурс] URL: http://www.cs.ucr.edu/~eamonn/UCRsuite.html (дата обращения: 31.05.2014).

21. Woop S. Embree ray tracing kernels for CPUs and the Xeon Phi architecture / S. Woop, L. Feng, I. Wald, C. Benthin // International Conference on Computer Graphics and Interactive Techniques, SIGGRAPH, Anaheim, CA, USA, 21-25 July, 2013. – ACM, 2013. – P. 44.