Параллельный алгоритм поиска локально похожих подпоследовательностей временного ряда для ускорителей на базе архитектуры Intel MIC*

А.В. Мовчан, М.Л. Цымблер

Южно-Уральский государственный университет (НИУ)

В работе рассматривается задача поиска локально похожих подпоследовательностей временного ряда. Необходимо найти все подпоследовательности ряда, расстояние от которых до заданного поискового запроса минимально среди всех соседних подпоследовательностей, отстоящих от запроса не более чем на заданное пороговое значение. В качестве расстояния (меры схожести) используется динамическая трансформация шкалы времени (Dynamic Time Warping, DTW), которая на сегодня признается лучшей мерой для большинства приложений временных рядов. Вычисление DTW, однако, является затратной операцией, несмотря на существующие алгоритмические техники сокращения вычислений. Имеющиеся подходы к вычислению DTW с помощью многоядерных ускорителей задействуют архитектуры GPU и FPGA, оставляя без внимания потенциал архитектуры Intel Many Integrated Core. В работе предлагается параллельный алгоритм решения указанной задачи, использующий как центральный процессор, так и многоядерный сопроцессор Intel Xeon Phi. Реализация основана на технологии параллельного программирования OpenMP и режиме выполнения приложения, при котором часть кода и данных выгружается на сопроцессор. Алгоритм предполагает использование на стороне процессора очереди подпоследовательностей, данные о которых выгружаются на сопроцессор для вычисления DTW, что обеспечивает высокую интенсивность вычислений, выполняемых на сопроцессоре. Приведены результаты экспериментов, подтверждающих эффективность разработанного алгоритма.

1. Введение

Временной ряд представляет собой совокупность вещественных значений, каждое из которых ассоциировано с последовательными отметками времени. Поиск похожих подпоследовательностей временного ряда представляет собой одну из основных проблем интеллектуального анализа временных рядов, возникающую в широком спектре предметных областей: мониторинг показателей функциональной диагностики организма человека [3], моделирование климата [1], финансовое прогнозирование [2] и др.

Поиск локально похожих подпоследовательностей предполагает, что имеется временной ряд, в котором необходимо осуществить поиск заданного ряда меньшей длины (поискового запроса), и задано пороговое значение расстояния (меры схожести подпоследовательностей). Решением задачи является множество подпоследовательностей исходного ряда, каждая из которых удовлетворяет следующему условию: расстояние от подпоследовательности до запроса минимально среди соседних с ней подпоследовательностей, отстоящих от запроса не более чем на пороговое значение.

На сегодня динамическая трансформация времени (Dynamic Time Warping, DTW) [5] является наиболее популярной мерой во многих приложениях интеллектуального анализа временных рядов [6]. Однако по сравнению с Евклидовым расстоянием DTW вычислительно более сложна. На сегодня предложено большое количество подходов для решения данной задачи: отбрасывание заведомо непохожих подпоследовательностей на основе оценки ниж-

*Работа выполнена при финансовой поддержке Минобрнауки России в рамках ФЦП «Исследования и разработки по приоритетным направлениям развития научно-технологического комплекса России на 2014–2020 годы» (Соглашение № 14.574.21.0035 от 17.06.2014, идентификатор RFMEFI57414X0035).

ней границы расстояния [6], повторное использование вычислений [16], индексирование [11], раннее прекращение заведомо нерезультативных вычислений [14] и др. Тем не менее, вычисление DTW по-прежнему занимает существенную часть времени работы алгоритмов. В силу этого актуальными являются исследования, посвященные использованию параллельных вычислений для решения данной задачи на кластерных системах [18], многоядерных процессорах [17], FPGA и GPU [16, 19, 20]. В этой связи представляется недооцененным потенциал многоядерных ускорителей на базе архитектуры Intel Many Integrated Core [7] для решения задач поиска похожих подпоследовательностей временных рядов.

В данной статье предлагается параллельный алгоритм поиска локально похожих подпоследовательностей временного ряда для узла, оснащенного многоядерным сопроцесором Intel Xeon Phi. Статья организована следующим образом. Раздел 2 содержит формальное определение задачи и краткое описание архитектуры и модели программирования Intel Xeon Phi, а также обзор работ по теме исследования. В разделе 3 описан предложенный алгоритм. Результаты экспериментов представлены в разделе 4. В заключении суммируются полученные результаты и указываются направления будущих исследований.

2. Контекст исследования и обзор работ

2.1. Формальная постановка задачи

Временной ряд (time series) T — упорядоченная последовательность $t_1, t_2, ..., t_N$ (где N — длина последовательности) вещественных значений, каждое из которых ассоциировано с отметкой времени.

Подпоследовательность (subsequence) T_{im} временного ряда T представляет собой непрерывное подмножество T, начинающееся с позиции i, и имеющее длину m,
т.е. $T_{im} = t_i, t_{i+1}, \ldots, t_{i+m-1}$, где $1 \leq i \leq N$ и $i + m \leq N$.

Запрос (query) Q — временной ряд, поиск которого необходимо осуществить во временном ряде T. Пусть n — длина запроса, $n \ll N$.

Задача *поиска локально похожих подпоследовательностей (local-best-match subsequence search)* [19] определяется следующим образом. Пусть D является мерой схожести, $\mathcal{E} > 0$ — пороговое значение меры и $\mathbb{L}$ означает результирующее множество подпоследовательностей. Тогда $T_{im} \in \mathbb{L} \Leftrightarrow T_{im}$ удовлетворяет следующим условиям:

1. $m = n$;
2. $D(T_{im}, Q) < \mathcal{E}$;
3. $i = \underset{j \in \{i-1, i, i+1\}}{\operatorname{argmin}} D(T_{jm}, Q)$.

Динамическая трансформация шкалы времени (Dynamic Time Warping, DTW) представляет собой меру схожести двух временных рядов. Расстояние на основе DTW между двумя временными рядами X и Y, где $X = x_1, x_2, ..., x_N$ и $Y = y_1, y_2, ..., y_N$, обозначается как $D(X, Y)$ и определяется следующим образом.

$$D(X, Y) = d(N, N),$$

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

$$d(0, 0) = 0; d(i, 0) = d(0, j) = \infty; i = 1, 2, \ldots, N; j = 1, 2, \ldots, N.$$

2.2. Архитектура и модель программирования сопроцессора Intel Xeon Phi

Многоядерный сопроцессор Intel Xeon Phi состоит из 61 ядра на базе архитектуры x86, соединенных высокоскоростной двунаправленной шиной, где каждое ядро поддерживает 4× гипертрединг и содержит 512-битный векторный процессор. Каждое ядро имеет собственный кэш 1 и 2 уровня, при этом обеспечивается когерентность кэшей всех ядер. Сопроцессор соединяется с хост-компьютером посредством интерфейса PCI Express. Поскольку сопроцессор Intel Xeon Phi основан на архитектуре Intel x86, он поддерживает те же программные инструменты и модели программирования, что и ординарный процессор Intel Xeon.

Сопроцессор поддерживает следующие режимы запуска приложений: *native*, *offload* и *symmetric*. В режиме *native* приложение выполняется независимо исключительно на сопроцессоре. В режиме *offload* приложение запускается на процессоре и выгружает вычислительно интенсивную часть работы (код и данные) на сопроцессор. Режим *symmetric* позволяет сопроцессору и процессору взаимодействовать в рамках модели обмена сообщениями (Message Passing Interface).

2.3. Работы по тематике исследования

В настоящее время DTW рассматривается научным сообществом как наилучшая мера схожести для большинства приложений интеллектуального анализа временных рядов [6], несмотря на большие временны́е затраты, связанные с ее вычислением [9,18]. Исследования, посвященные ускорению вычисления DTW, представлены следующими работами.

Алгоритм SPRING [15] использует технику повторного использования вычислений. Однако, данная техника ограничивает приложение алгоритма, поскольку повторное использование данных предполагает использование последовательностей, не подвергаемых нормализации. В работе [11] для ускорения вычислений используется техника индексирования, которая требует заранее задавать длину поискового запроса, что не всегда приемлемо. Авторами работы [10] предложен многомерный индекс для поисковых запросов с различными длинами. Техника отбрасывания заведомо непохожих подпоследовательностей на основе оценки нижней границы расстояния предложена в работе [8]. Алгоритм UCR-DTW [14] интегрирует большое количество существующих техник ускорения вычислений DTW и является на сегодня, вероятно, самым быстрым последовательным алгоритмом поиска похожих подпоследовательностей.

Вышеперечисленные алгоритмы направлены на сокращение количества вызовов процедуры вычисления DTW, но не ускорения процедуры вычисления DTW как таковой. Однако, в силу своей вычислительной сложности, вычисление DTW по-прежнему занимает существенную часть времени выполнения поиска подпоследовательностей. Данное обстоятельство стимулировало исследования, направленные на использование параллельного аппаратного обеспечения для распределения вычислений DTW для разных подпоследовательностей на разные вычислительные устройства.

В работе [17] подпоследовательности, начинающиеся с разных позиций временного ряда, направляются для вычисления DTW на различные процессоры Intel Xeon. В работе [18] разные поисковые запросы распределяются на разные ядра процессора, и каждая подпоследовательность пересылается на различные ядра для сравнения с запросами. Реализация на GPU [20] распараллеливает создание матрицы трансформации шкалы времени, однако путь трансформации вычисляется последовательно. В работе [16] предложена GPU-реализация, использующая те же идеи, что и в работе [17]. Реализация для FPGA, описанная в работе [16], предлагает наивный поиск похожих подпоследовательностей, не использующий предварительную обработку данных. Приложение, осуществляющее поиск, генерируется с помощью инструмента C-to-VHDL и ввиду отсутствия знания внутреннего устройства FPGA не может быть применено к задачам большой размерности. Для преодоления ука-

занных проблем в работе [19] предложен потоково-ориентированнный фреймворк, в котором реализован крупнозернистый параллелизм путем повторного использования данных различных вычислений DTW.

В данной работе на базе последовательного алгоритма поиска наиболее похожей подпоследовательности временного ряда [14] нами разработан алгоритм поиска локально похожих подпоследовательностей, который затем распараллелен с помощью технологии OpenMP и адаптирован для многоядерного сопроцессора Intel Xeon Phi на основе идей, предложенных нами ранее в работах [4, 12].

3. Параллельный алгоритм для сопроцессора Intel Xeon Phi

В данном разделе описаны этапы разработки параллельного алгоритма поиска локально похожих подпоследовательностей для многоядерного сопроцессора Intel Xeon Phi. В разделе 3.1 описан последовательный алгоритм решения данной задачи, построенный на основе алгоритма UCR-DTW [14]. Раздел 3.2 содержит описание распараллеливания последовательного алгоритма с помощью технологии OpenMP. В разделе 3.3 описана адаптация алгоритма, полученного на предыдущем шаге, для исполнения на сопроцессоре Intel Xeon Phi.

3.1. Последовательный алгоритм

Разработанный нами последовательный алгоритм поиска локально похожих подпоследовательностей представлен на рис. 1. Алгоритм получил название lbm-UCR-DTW, поскольку базируется на алгоритме UCR-DTW [14]. Оригинальный алгоритм использует каскад оценок динамической трансформации шкалы времени для отбрасывания заведомо непохожих подпоследовательностей (составная деятельность `Lower Bounding` на рис. 1).

Мы полагаем, что $|\mathbb{L}| \leq K$, т.е. результирующее множество содержит не более K подпоследовательностей, где K является параметром алгоритма. Данное ограничение является практически полезным ввиду возможного лимита оперативной памяти для хранения найденных подпоследовательностей и не ограничивает общность, поскольку мы можем рассмотреть случай $K = \infty$.

Переменная алгоритма `bsf` (best-so-far) используется для хранения текущей лучшей оценки расстояния от подпоследовательностей до запроса и вычисляется следующим образом:

$$bsf = \begin{cases} \mathcal{E} & , |\mathbb{L}| < K \\ \min(\mathcal{E}, \max\limits_{T_{mn} \in \mathbb{L}} DTW(T_{mn}, Q)) & , else \end{cases}$$

Для поиска локально похожих подпоследовательностей алгоритм осуществляет последовательный просмотр каждых трех соседних подпоследовательностей. Текущую обрабатываемую подпоследовательность T_{in} мы обозначаем как C_R, а две предшествующие ей и уже обработанные подпоследовательности T_{i-1n} и T_{i-2n} как C_M и C_L соответственно. Расстояние от данных подпоследовательностей до запроса обозначаются как $dist_R$, $dist_M$ и $dist_L$ соответственно. Вспомогательный алгоритм `Sliding` осуществляет обновление указанных переменных во время работы алгоритма.

В случае, если подпоследовательность C_M является локальным минимумом, то вспомогательный алгоритм `Update Result` включает подпоследовательность C_M в результирующее множество $\mathbb{L}$. Если после этого мощность множества $\mathbb{L}$ превышает K, то из данного множества исключается подпоследовательность, расстояние от которой до запроса максимально среди всех подпоследовательностей, входящих в $\mathbb{L}$, и далее обновляется значение переменной `bsf` в соответствии с вышеуказанной формулой.

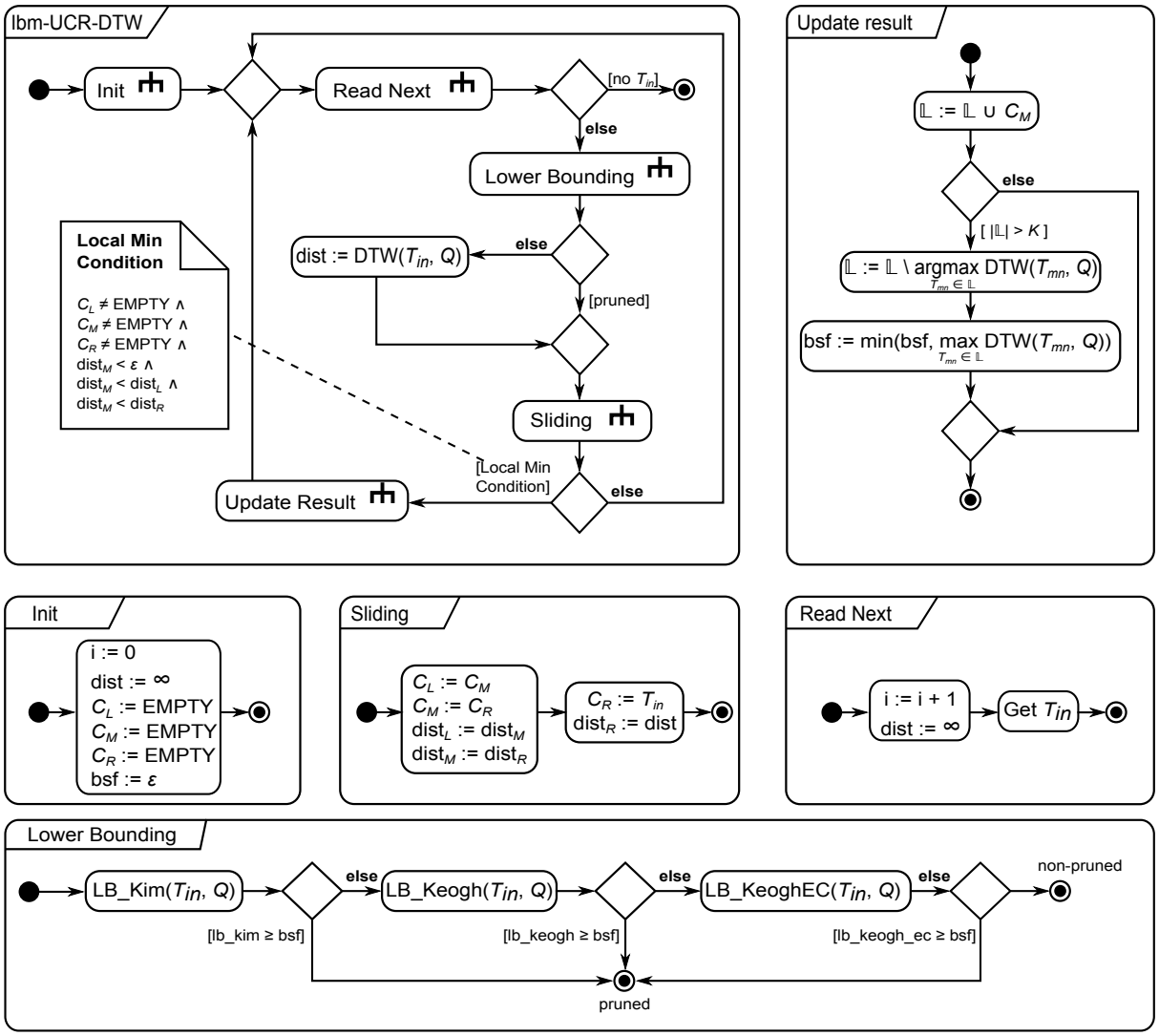

Рис. 1. Последовательный алгоритм lbm-UCR-DTW

Таким образом, схема работы алгоритма кратко может быть представлена следующим образом. Считанная подпоследовательность проверяется с помощью каскада оценок вспомогательного алгоритма `Lower Bounding`. Если подпоследовательность не отбрасывается как заведомо непохожая, то вычисляется DTW до запроса. Затем вспомогательный алгоритм `Sliding` выполняет обновление переменных C_L, C_M, C_R и соответсвующих им расстояний $dist_L$, $dist_M$, $dist_R$. Если выполнено условие локального минимума, то вспомогательный алгоритм `Update Result` обновляет результирующее множество $\mathbb{L}$. Алгоритм заканчивает работу, когда исчерпаны все подпоследовательности исходного временного ряда.

3.2. Параллельный алгоритм для процессора

На основе последовательного алгоритма, предложенного в предыдущем разделе, нами разработан параллельный алгоритм для процессора, представленный на рис. 2.

Для распараллеливания используется технология программирования OpenMP. Временной ряд разбивается на промежутки равной длины, каждый из которых обрабатывается отдельной OpenMP-нитью. Для того, чтобы избежать потери результирующих подпоследовательностей, находящихся на стыке промежутков, разбиение на промежутки осуществляется с перекрытием, равным длине запроса. Это означает, что начало каждого промежутка,

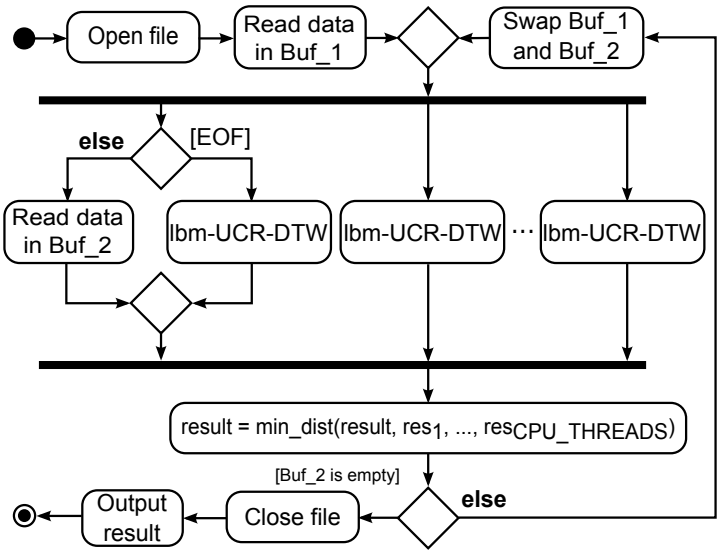

Рис. 2. Параллельный алгоритм для процессора

за исключением первого, пересекается с концом предыдущего промежутка в n точках.

Данный принцип разбиения формально может быть описан следующим образом.

Пусть P – количество OpenMP-нитей, тогда промежуток, назначаемый k-й нити, есть последовательностью T_{sl}, где

$$s = \begin{cases} 1 & , k = 0 \\ k \cdot \lfloor \frac{N}{P} \rfloor - n + 2 & , else \end{cases}$$

$$l = \begin{cases} \lfloor \frac{N}{P} \rfloor + 1 & , k = 0 \\ \lfloor \frac{N}{P} \rfloor + n - 1 + (N \ mod \ P) & , k = P - 1 \\ \lfloor \frac{N}{P} \rfloor + n & , else \end{cases}$$

Здесь `lbm-UCR-DTW` представляет собой подпрограмму, реализующую последовательный алгоритм, представленный в предыдущем разделе. Этот алгоритм использует разделяемую переменную `bsf`, что позволяет каждой нити отбросить заведомо непохожую подпоследовательность, если расстояние до нее больше значения указанной переменной. Главная нить считывает данные из файла одновременно с обработкой остальными нитями уже прочитанных данных.

3.3. Параллельный алгоритм для сопроцессора

Параллельный алгоритм для процессора и сопроцессора Intel Xeon Phi представлен на рис. 3.

Основная идея данного параллельного алгоритма заключается в том, что сопроцессор используется только для вычислений DTW, а процессор выполняет вычисление каскада оценок, подготавливая подпоследовательности для сопроцессора, и вычисление DTW, если сопроцессор полностью занят. Процессор поддерживает очередь кандидатов (т.е. подпоследовательностей, для каждой из которых сопроцессор должен вычислить DTW).

Элементами очереди являются кортежи вида (i, A), соответствующие подпоследовательности-кандидату T_{in}. Здесь A представляет собой массив длины n, содержащий оценки

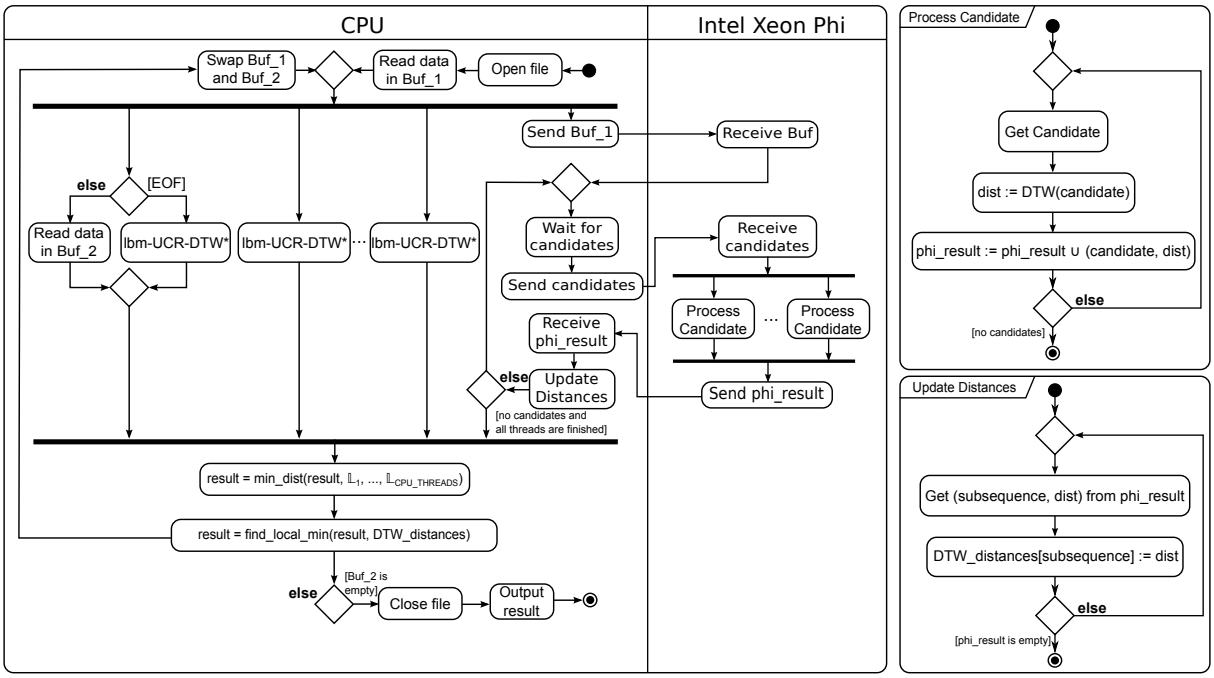

Рис. 3. Параллельный алгоритм для процессора и сопроцессора Intel Xeon Phi

LB_{Keogh} для каждой позиции подпоследовательности, который используется для заблаговременной отмены вычисления DTW [14].

При заполнении очереди она выгружается на сопроцессор для выполнения вычислений DTW. Максимальное количество элементов в очереди является параметром алгоритма и вычисляется как $C \cdot h \cdot W$, где C — количество ядер сопроцессора, доступных для вычислений (например, для Intel Xeon Phi SE10X $C = 60$), h — фактор гипертредининга сопроцессора ($h = 4$ для ранее упомянутой модели сопроцессора), W — количество кандидатов, обрабатываемых одной нитью сопроцессора, которое является параметром алгоритма.

Работа алгоритма может быть описана следующим образом. Одна из нитей процессора объявляется *мастером*, остальные — *рабочими*. Сначала мастер отправляет буфер с текущим фрагментом временного ряда на сопроцессор. Как только очередь заполняется, мастер выгружает ее на сопроцессор, а сопроцессор выполняет вычисление DTW для соответствующих подпоследовательностей.

Вспомогательный алгоритм `lbm-UCR-DTW*`, изображенный на рис. 4, реализует поведение рабочего. Рабочий выполняет вычисление каскадных оценок для подпоследовательности. Если подпоследовательность не похожа на запрос, то рабочий отбрасывает ее, иначе подпоследовательность добавляется в очередь. Если очередь заполнена, и данные, загруженные ранее на сопроцессор, еще не обработаны, то рабочий вычисляет DTW самостоятельно. При этом реализация поиска локально похожих подпоследовательностей идеологически близка последовательному алгоритму, описанному в разделе 3.1. Результаты вычислений DTW рабочий сохраняет в разделяемом массиве для последующего поиска локально похожих подпоследовательностей.

После выгрузки кандидатов на сопроцессор для каждого кандидата выполняется вычисление DTW, и пары «индекс-расстояние» выгружаются обратно на процессор.

После того, как вычисления на процессоре и сопроцессоре завершены, выполняется поиск локально похожих подпоследовательностей в разделяемом массиве, заполненном ранее.

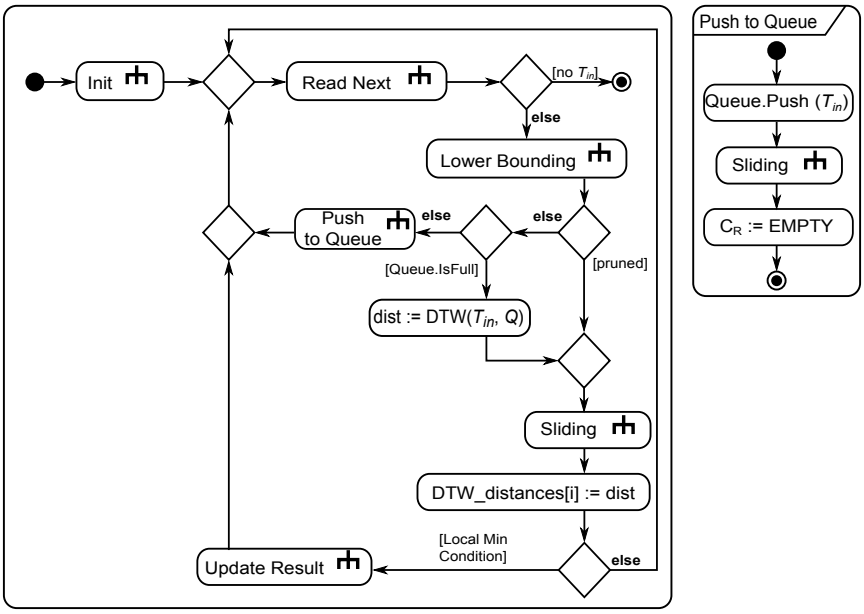

Рис. 4. Вспомогательный алгоритм lbm-UCR-DTW*

4. Вычислительные эксперименты

Для оценки разработанного алгоритма мы выполнили эксперименты на узле суперкомпьютера «Торнадо ЮУрГУ»[1], спецификации которого представлены в табл. 1.

Таблица 1. Спецификация узла суперкомпьютера «Торнадо ЮУрГУ»

Спецификации	Процессор	Сопроцессор
Модель	Intel Xeon X5680	Intel Xeon Phi SE10X
Количество ядер	6	61
Тактовая частота, ГГц	3.33	1.1
Количество нитей на ядро	2	4
Пиковая производительность, TFLOPS	0.371	1.076

В качестве временного ряда фигурировали как синтетические, так и реальные данные. В экспериментах измерялось время поиска похожих подпоследовательностей для запросов различной длины. Мы также исследовали влияние порогового значения $\mathcal{E}$ на время работы алгоритма.

В экспериментах использовалось пороговое значение $\mathcal{E} = 2 + D_{min}$, где D_{min} представляет расстояние до самой похожей подпоследовательности, предварительно вычисленное с помощью алгоритма, предложенного в работах [4, 12]. Мощность результирующего множества ограничивалась сверху значением $K = 10000$.

В первой серии экспериментов мы использовали синтетический временной ряд, состоящий из 100 млн. точек, сгенерированный на основе модели случайных блужданий [13].

Результаты экспериментов на синтетических данных, представленные на рис. 5, показывают, что разработанный алгоритм более эффективен для запросов большей длины. В слу-

[1] supercomputer.susu.ru/en/computers/tornado/

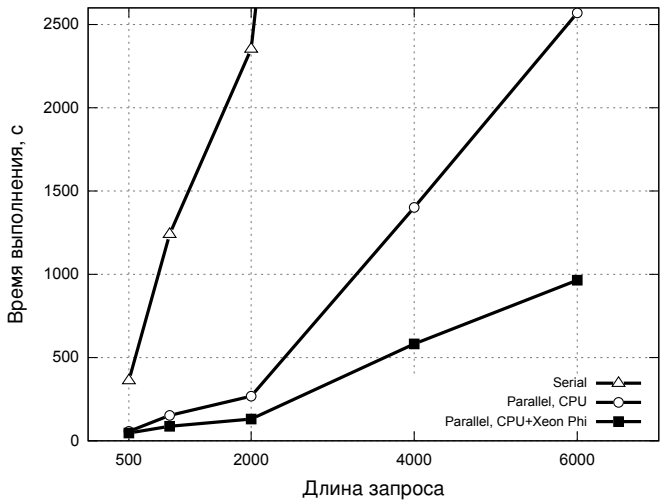

Рис. 5. Производительность на синтетических данных

чае, когда запросы имеют меньшую длину, наш алгоритм показывает производительность, сходную с параллельным алгоритмом для процессора (не использующим сопроцессор).

Вторая серия экспериментов исследует производительность разработанного алгоритма на реальных данных ЭКГ, состоящих из 20 млн. точек (около 22 час. ЭКГ, снятой с дискретизацией 250 Гц).

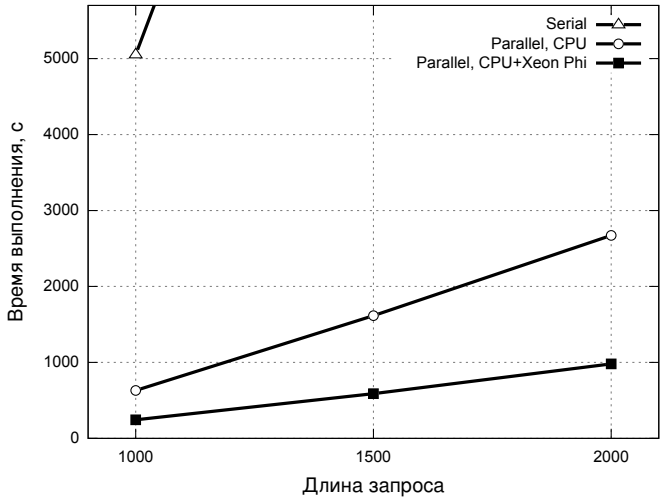

Рис. 6. Производительность на реальных данных

Результаты экспериментов на реальных данных показаны на рис. 6. Разработанный алгоритм показывает в почти три раза бóльшую производительность, чем параллельный алгоритм, не использующий сопроцессор.

Влияние порогового значения $\mathcal{E}$ на время работы алгоритма показано на рис. 7. Данный эксперимент производился с использованием параллельного алгоритма поиска локально похожих подпоследовательностей для процессора и сопроцессора Intel Xeon Phi. В качестве данных эксперимента использовались синтетический и реальный временные ряды, рассмотренные в предыдущих экспериментах. Как и ожидалось, с увеличением порогового значения $\mathcal{E}$ время выполнения увеличивается.

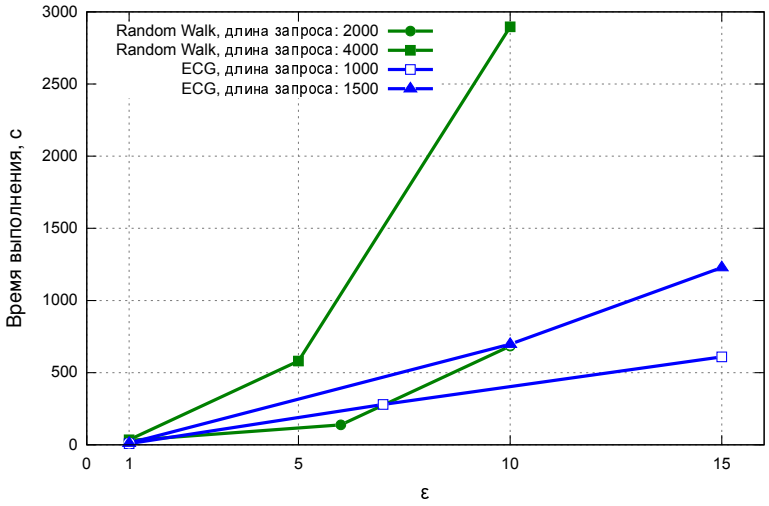

Рис. 7. Влияние порогового значения $\mathcal{E}$ на время выполнения

5. Заключение

В данной статье описаны проектирование и реализация параллельного алгоритма поиска локально похожих подпоследовательностей временного ряда, использующего в качестве мероы схожести динамическую трансформацию шкалы времени, для архитектуры Intel Many Integrated Core.

На основе последовательного алгоритма поиска самой похожей подпоследовательности временного ряда, комбинирующего известные на сегодня техники редукции вычислений, разработан последовательный алгоритм решения указанной задачи.

На следующем шаге разработки выполнено распараллеливание полученного алгоритма на основе технологии OpenMP. Временной ряд разбивается на промежутки равной длины, обрабатываемые отдельными OpenMP-нитями. Для исключения потерь результирующих подпоследовательностей, находящихся на стыке промежутков, разбиение осуществляется с частичным нахлестом соседних промежутков.

Наконец, алгоритм, полученный на предыдущем шаге, адаптирован для выполнения вычислений как на центральном процессоре, так и на многоядерном сопроцессоре Intel Xeon Phi. Сопроцессор используется только для вычисления расстояний между подпоследовательностями и запросом. Процессор выполняет отбрасывание заведомо непохожих подпоследовательностей и подготовку подпоследовательностей для обработки сопроцессором, вычисляя расстояния лишь при отсуствии другой работы. Взаимодействие процессора и сопроцессора реализовано с помощью режима *offload*, когда часть кода и данных выгружается на сопроцессор. Высокая интенсивность вычислений, выполняемых на сопроцессоре, достигается за счет использования на стороне процессора очереди подпоследовательностей, данные о которых выгружаются на сопроцессор для вычисления динамической трансформации шкалы времени.

Эксперименты, проведенные на синтетических и реальных данных, показали эффективность разработанного алгоритма и его превосходство над последовательным алгоритмом и параллельным алгоритмом, использующим только процессор.

В качестве возможного направления дальнейших исследований интересными представляются следующие задачи: модернизация разработанного алгоритма для случая, когда процессор оснащен несколькими сопроцессорами Intel Xeon Phi, и расширение разработанного алгоритма для кластерной системы, каждый вычислительный узел которой оснащен сопроцессором Intel Xeon Phi.

Литература

1. Абдуллаев С.М., Ленская О.Ю., Гаязова А.О., Иванова О.Н., Носков А.А., Соболев Д.Н., Радченко Г.И. Алгоритмы краткосрочного прогноза с использованием радиолокационных данных: оценка трасляции и композиционный дисплей жизненного цикла // Вестник ЮУрГУ. Серия: Вычислительная математика и информатика. 2014. Т. 3, № 1. С. 17–32.

2. 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.

3. Епишев В.В., Исаев А.П., Миниахметов Р.М., Мовчан А.В., Смирнов А.С., Соколинский Л.Б., Цымблер М.Л., Эрлих В.В. Система интеллектуального анализа данных физиологических исследований в спорте высших достижений // Вестник ЮУрГУ. Серия: Вычислительная математика и информатика. 2013. Т. 2, № 1. С. 44–54.

4. Мовчан А.В., Цымблер М.Л. Нахождение похожих подпоследовательностей временного ряда с помощью многоядерного сопроцессора Intel Xeon Phi // Параллельные вычислительные технологии (ПаВТ'2015): труды международной научной конференции (Екатеринбург, 31 марта – 2 апреля 2015 г.). Челябинск: Издательский центр ЮУрГУ, 2015. С. 212–224.

5. Berndt D.J., Clifford J. Using Dynamic Time Warping to Find Patterns in Time Series // Knowledge Discovery in Databases: Papers from the 1994 AAAI Workshop, Seattle, Washington, July 1994. AAAI Press, 1994. P. 359–370.

6. 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.

7. Duran A., Klemm M. The Intel Many Integrated Core Architecture // 2012 International Conference on High Performance Computing and Simulation, HPCS 2012, Madrid, Spain, July 2–6, 2012. IEEE, 2012. P. 365–366.

8. Fu A.W-C., Keogh E.J., Lau L.Y.H., Ratanamahatana C., Wong R.C.-W. Scaling and Time Warping in Time Series Querying // Proceedings of the 31st International Conference on Very Large Data Bases, Trondheim, Norway, August 30 – September 2, 2005. P. 649–660.

9. Fu A.W-C., Keogh E.J., Lau L.Y.H., Ratanamahatana C., Wong R.C.-W. Scaling and Time Warping in Time Series Querying // VLDB Journal. 2008. Vol. 17, No. 4. P. 899–921.

10. Keogh E.J., Wei L., Xi X., Vlachos M., Lee S.H., Protopapas P. Supporting Exact Indexing of Arbitrarily Rotated Shapes and Periodic Time Series under Euclidean and Warping Distance Measures // VLDB Journal. 2009. Vol. 18, No. 3. P. 611–630.

11. Lim S.-H., Park H., Kim S.-W. Using Multiple Indexes for Efficient Subsequence Matching in Time-series Databases // Database Systems for Advanced Applications, 11th International Conference, DASFAA 2006, Singapore, April 12–15, 2006, Proceedings. Lecture Notes in Computer Science. Vol. 3882. Springer, 2006. P. 65–79.

12. Miniakhmetov R.M., Movchan A.V., Zymbler M.L. Accelerating Time Series Subsequence Matching on the Intel Xeon Phi Many-core Coprocessor // Proceedings of the 38th International Convention on Information and Communication Technology, Electronics and

Microelectronics, MIPRO'2015, Opatija, Croatia, May 25–29, 2015. IEEE, 2015. P. 1675–1680.

13. Pearson K. The Problem of the Random Walk // Nature. 1905. Vol. 72, No. 1865. P. 294.

14. 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 // The 18th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Beijing, China, 12–16 August, 2012. ACM, 2012. P. 262–270.

15. Sakurai Y., Faloutsos C., Yamamuro M. Stream Monitoring under the Time Warping Distance // Proceedings of the 23rd International Conference on Data Engineering, ICDE 2007, The Marmara Hotel, Istanbul, Turkey, April 15–20, 2007. P. 1046–1055.

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

17. 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.

18. Takahashi N., Yoshihisa T., Sakurai Y., Kanazawa M. A Parallelized Data Stream Processing System Using Dynamic Time Warping Distance // 2009 International Conference on Complex, Intelligent and Software Intensive Systems, CISIS 2009, Fukuoka, Japan, March 16–19, 2009. IEEE Computer Society, 2009. P. 1100–1105.

19. Wang Z., Huang S., Wang L., Li H., Wang Y., Yang H. Accelerating Subsequence Similarity Search Based on Dynamic Time Warping Distance with FPGA // Proceedings of the 2013 ACM/SIGDA International Symposium on Field Programmable Gate Arrays, FPGA '13, Monterey, CA, USA, February 11–13, 2013. ACM, 2013. P. 53–62.

20. Zhang Y., Adl K., Glass J.R. Fast Spoken Query Detection Using Lower-bound Dynamic Time Warping on Graphical Processing Units // Proceedings of the 2012 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2012, Kyoto, Japan, March 25–30, 2012. IEEE, 2012. P. 5173–5176.