Параллельная реализация поиска самой похожей подпоследовательности временного ряда для систем с распределенной памятью*

А.В. Мовчан, М.Л. Цымблер

Южно-Уральский государственный университет (НИУ)

В работе представлена параллельная реализация поиска самой похожей подпоследовательности временного ряда для вычислительного кластера с узлами на базе многоядерных ускорителей Intel MIC. Алгоритм предполагает три уровня параллелизма по данным. На первом уровне временной ряд разбивается на фрагменты равной длины, каждый из которых обрабатывается отдельным узлом вычислительного кластера; взаимодействие узлов реализуется на основе технологии MPI. Второй уровень параллелизма предполагает разбиение фрагмента на сегменты равной длины, обрабатываемые нитями на основе технологии OpenMP. Третий уровень параллелизма заключается в балансировке вычислительной нагрузки между процессором и ускорителем. Процессор выполняет отбрасывание заведомо непохожих подпоследовательностей. Ускоритель выполняет наиболее трудоемкие вычисления меры схожести. Результаты вычислительных экспериментов показывают эффективность разработанного алгоритма.

Ключевые слова: интеллектуальный анализ временных рядов, вычислительный кластер, сопроцессор Intel Xeon Phi, OpenMP, MPI, динамическая трансформация шкалы времени.

1. Введение

Временной ряд представляет собой хронологически упорядоченную последовательность вещественных значений, ассоциированных с отметками времени. Поиск похожих подпоследовательностей временного ряда предполагает нахождение участков данного ряда, которые являются похожими на заданный ряд существенно меньшей длины. Данная задача возникает в широком спектре предметных областей: мониторинг показателей физиологической активности человека, моделирование климата и предсказание погоды, финансовое прогнозирование и др.

В качестве меры схожести временных рядов в настоящее время наиболее часто используется динамическая трансформация шкалы времени (Dynamic Time Warping, DTW) [1]. Мера DTW признается наилучшей для большинства приложений временных рядов [2], несмотря на то, что она является вычислительно более сложной по сравнению с Евклидовым расстоянием и ее вычисление занимает существенную часть времени работы алгоритмов поиска похожих подпоследовательностей [4,14]. Для современных приложений, продуцирующих временные ряды из триллионов элементов, требуются эффективные параллельные алгоритмы поиска для многопроцессорных многоядерных аппаратных платформ.

Существующие на сегодня параллельные реализации алгоритмов поиска похожих подпоследовательностей временного ряда ориентированы в основном на ускорители GPU [6,10, 13,17,18] и FPGA [10,16]. Исследования, посвященные использованию для решения данной задачи кластерных вычислительных систем с узлами на базе многоядерных ускорителей, отсутствуют.

Данная работа продолжает исследование [7], в рамках которого разработан параллельный алгоритм поиска самой похожей подпоследовательности временного ряда для ускори-

*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» (Госконтракт № 14.574.21.0035).

теля с архитектурой Intel Many Integrated Core (MIC) [3]. Разработанный ранее алгоритм переносится на платформу вычислительного кластера, узлы которого оснащены ускорителями на базе архитектуры Intel MIC. Статья организована следующим образом. В разделе 2 приведена формальная постановка задачи и кратко описано аппаратно-программное окружение исследования. Раздел 3 содержит описание принципов проектирования и реализации алгоритма. Результаты вычислительных экспериментов по исследованию эффективности предложенного алгоритма представлены в разделе 4. В заключении суммируются полученные результаты.

2. Формальные определения и контекст исследования

2.1. Постановка задачи

Временной ряд (time series) T представляет собой хронологически упорядоченную последовательность вещественных значений $t_1, t_2, ..., t_N$, ассоциированных с отметками времени, где N — длина последовательности.

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

Запрос (query) Q — подпоследовательность, поиск которой будет осуществляться в T. Пусть n — длина запроса, $n \ll N$.

Задача *поиска самой похожей подпоследовательности (best-match-search)* предполагает нахождение подпоследовательности $T_{j\,n}$ (где $1 \leq j \leq N - n$), максимально похожей на запрос Q в смысле некоторой меры D, т.е. $T_{j\,n} = \underset{1 \leq i \leq N-n}{\operatorname{argmin}} D(T_{i\,n}, Q)$.

В данной работе в качестве меры схожести D временных рядов используется *динамическая трансформация шкалы времени (Dynamic Time Warping, DTW)*, определяемая следующим образом. Пусть имеются два временных ряда $X = x_1, x_2, ..., x_N$ и $Y = y_1, y_2, ..., y_N$, тогда

$$DTW(X, Y) = d(N, N), \text{ где}$$

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

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

2.2. Аппаратно-программное окружение

В настоящее время вычислительные системы с кластерной архитектурой занимают 85% списка TOP500 самых мощных суперкомпьютеров мира[1]. Около 20% систем первой сотни данного списка оснащены многоядерными ускорителями. Вычислительный кластер Tianhe-2, занимающий первую строку списка, оснащен ускорителями на базе архитектуры Intel MIC [3]. На сегодня наиболее часто используемым представителем семейства Intel MIC является ускоритель (сопроцессор) Intel Xeon Phi.

Сопроцессор Intel Xeon Phi состоит из 61 ядра на базе архитектуры x86, соединенных высокоскоростной двунаправленной шиной, где каждое ядро поддерживает 4-кратную гиперпоточность и содержит 512-битный векторный процессор. Каждое ядро имеет собственный кэш 1 и 2 уровня, при этом обеспечивается когерентность кэшей всех ядер. Сопроцессор

[1] `http://top500.org` (ноябрь 2015 г.)

соединяется с хост-компьютером посредством интерфейса PCI Express. Поскольку сопроцессор Intel Xeon Phi основан на архитектуре x86, он поддерживает те же инструменты и модели программирования, что и процессор Intel Xeon (OpenMP, Intel Cilk и др.).

Сопроцессор поддерживает следующие режимы запуска приложений: *native*, *offload* и *symmetric*. В режиме *native* приложение выполняется независимо на сопроцессоре. В режиме *offload* приложение запускается на процессоре и выгружает вычислительно интенсивную часть работы (код и данные) на сопроцессор. Режим *symmetric* поддерживает взаимодействие сопроцессора и процессора в рамках модели обмена сообщениями.

2.3. Результаты предыдущих исследований

Данная статья является продолжением исследования, начатого в работе [7], где авторами разработан параллельный алгоритм поиска самой похожей подпоследовательности временного ряда для сопроцессора Intel Xeon Phi. Данный алгоритм основан на последовательном алгоритме UCR-DTW [9], который в настоящее время является одним из самых быстрых последовательных алгоритмов решения данной задачи [16]. Алгоритм UCR-DTW выполняет последовательно вычисление динамической трансформации шкалы времени для каждой подпоследовательности исходного временного ряда и запроса, используя при этом каскад оценок значения меры DTW для отбрасывания заведомо непохожих подпоследовательностей.

Параллельный алгоритм, разработанный на предыдущем этапе исследования, кратко может быть описан следующим образом. Алгоритм задействует как процессор, так и сопроцессор. Сопроцессор используется исключительно для выполнения вычислительно затратной операции нахождения меры DTW для подпоследовательностей. Процессор выполняет вычисление каскада оценок меры DTW для отбрасывания непохожих подпоследовательностей и подготовку данных для выгрузки на сопроцессор. Процессор поддерживает очередь подпоследовательностей-кандидатов, выгружаемых на сопроцессор для вычисления меры DTW. Распараллеливание осуществляется на основе технологии OpenMP, для чего временной ряд разбивается на сегменты равной длины. Эксперименты на реальных и синтетических данных показали преимущество разработанного алгоритма над аналогами для GPU, особенно в случае поиска запросов длины более 10^3.

Новый алгоритм должен распараллеливать вычисления между процессорами узлов кластерной системы и использовать алгоритм, разработанный ранее.

3. Принципы проектирования и реализации алгоритма

3.1. Уровни параллелизма алгоритма

Алгоритм предполагает три уровня распараллеливания по данным, представленные на рис. 1.

Первый уровень параллелизма представляет распределение нагрузки между вычислительными узлами кластерной системы. На этом уровне временной ряд разбивается на фрагменты равной длины, каждый из которых обрабатывается отдельным узлом вычислительного кластера. В процессе вычислений узлы обмениваются информацией о найденном значении меры DTW для подпоследовательности, которая на текущий момент является самой похожей на запрос. Для реализации данного вида параллелизма используется технология MPI.

Второй и третий уровни распараллеливания осуществляются в рамках одного вычислительного узла, процессор и сопроцессор которого совместно исполняют разработанный ранее алгоритм [7].

Второй уровень параллелизма реализуется посредством разбиения фрагмента на сегменты для того, чтобы каждый сегмент обрабатывался отдельной нитью процессора или

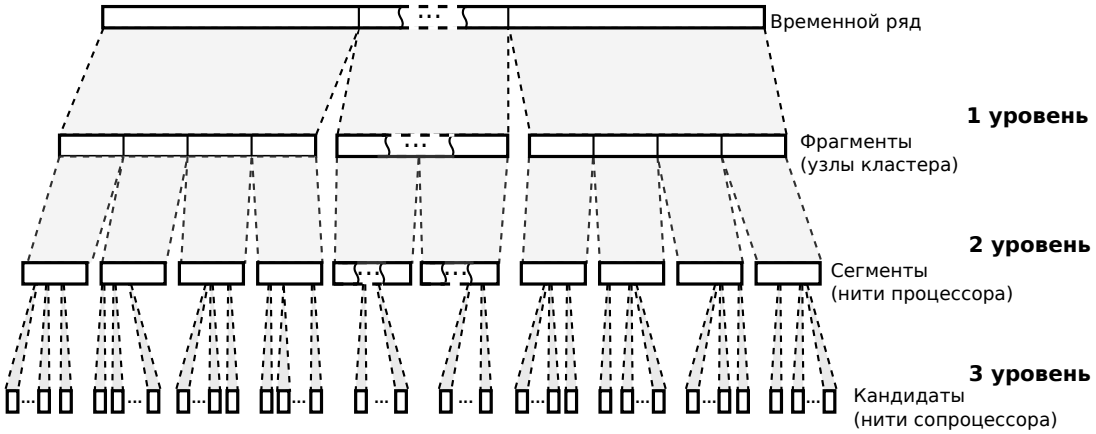


Рис. 1: Уровни параллелизма алгоритма

сопроцессора. Данный вид параллелизма реализуется с помощью технологии программирования OpenMP.

Третий уровень параллелизма заключается в распределении вычислительной нагрузки между процессором и многоядерным сопроцессором таким образом, чтобы выполнять сложные вычисления только на сопроцессоре. Процессор выполняет вычисление каскада оценок меры DTW (LB_{Kim} [9], LB_{Keogh} [4], $LB_{KeoghEC}$ [9]) для отбрасывания заведомо непохожих подпоследовательностей. Сопроцессор используется для вычисления меры DTW. На стороне процессора организуется очередь обрабатываемых подпоследовательностей. После заполнения данные, имеющиеся в очереди, и код для их обработки посредством использования режима *offload* выгружаются на ускоритель, где осуществляется вычисление меры DTW.

3.2. Фрагментация и сегментация временного ряда

Для реализации описанных уровней параллелизма необходимо обеспечить разбиение временного ряда на порции между вычислителями: распределение фрагментов по узлам кластерной системы и разделение каждого фрагмента на сегменты.

Для того, чтобы избежать потери результирующих подпоследовательностей, находящихся на стыке порций, нами используется техника *разбиения с перекрытием*, которая заключается в следующем. В конец каждой порции временного ряда, за исключением последней по порядку, добавляется $n-1$ точек, взятых с начала следующей порции, где n — длина запроса. Формальное определение разбиения с перекрытием выглядит следующим образом.

Пусть P — количество порций, F^k — k-я порция временного ряда $T = t_1, t_2, \ldots, t_N$, где $0 \le k \le P-1$. Обозначим за M общее количество подпоследовательностей, которые необходимо обработать, $M = N - n + 1$. Тогда F^k определяется как подпоследовательность $T_{start\,len}$, где

$$start = k \cdot \lfloor \frac{M}{P} \rfloor + 1$$

$$len = \begin{cases} \lfloor \frac{M}{P} \rfloor + (M \bmod P) + n - 1 & , k = P-1 \\ \lfloor \frac{M}{P} \rfloor + n - 1 & , else \end{cases}$$

Фрагментация предполагает, что временной ряд разделяется на фрагменты равной длины с помощью описанной выше техники, которые распределяются по узлам вычислительного кластера (один фрагмент на один узел). Количество порций-фрагментов P совпа-

дает с количеством узлов вычислительного кластера. В качестве длины запроса при разделении без ограничения общности берется n_{max} — максимальная из всех возможных длин запроса, $n_{max} \ll N$.

Для распределения вычислительной нагрузки между нитями процессора каждый фрагмент подвергается *сегментации* с помощью описанной выше техники. Длина сегмента является параметром алгоритма. Количество порций-сегментов определяется следующим образом.

Пусть H — количество сегментов в фрагменте F^k (где $F^k \equiv T_{start\,len}$ определено выше), S — количество нитей, выделяемых для параллельной обработки сегментов, L — длина сегмента. Тогда

$$H = \lceil \frac{len}{S \cdot L} \rceil \cdot S.$$

Количество сегментов является кратным количеству нитей, выделяемых для их обработки, для обеспечения лучшей балансировки вычислительной нагрузки между нитями. Сегмент должен целиком помещаться в оперативную память как процессора, так и сопроцессора.

Параметр L (длина сегмента) подбирается эмпирическим путем исходя из следующих соображений. Слишком малая длина сегмента приведет к его быстрой обработке, но при этом возрастут накладные расходы на динамическое выделение сегментов нитям процессора. Слишком большая длина сегмента увеличивает вероятность наличия в нем заранее неизвестного количества непохожих (отбрасываемых) подпоследовательностей, что приведет к плохой балансировке вычислительной нагрузки.

3.3. Обновление оценки схожести запроса и самой похожей подпоследовательности

Исходный последовательный алгоритм [9] использует переменную `bsf` (best-so-far) для хранения оценки схожести запроса и подпоследовательности, наиболее похожей на него на текущий момент. Для подпоследовательностей, имеющих схожесть с запросом меньше, чем `bsf`, вычисление меры DTW не производится (они отбрасываются как заведомо непохожие), что позволяет существенно сократить количество вычислений. В параллельном алгоритме [9] стандартными средствами технологии OpenMP переменная `bsf` объявляется разделяемой (shared) для нитей процессора, обрабатывающих сегменты ряда.

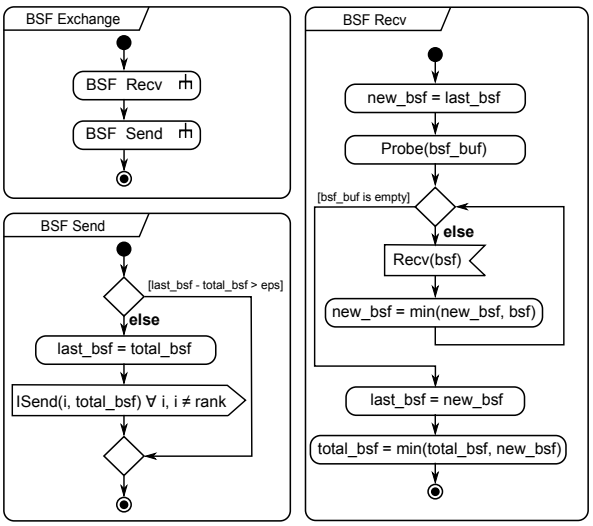


Рис. 2: Обновление оценки схожести

В случае платформы вычислительного кластера необходима альтернатива общей переменной, которая хранит наилучшую текущую оценку и обновляется всеми процессами, обрабатывающими фрагменты ряда. Данная реализация использует следующий подход к обновлению оценки схожести, представленный на рис. 2. Каждый процесс поддерживает локальную оценку схожести, которая обновляется перед началом обработки очередного сегмента. Обновление выполняется в два шага: сначала осуществляется изменение локальной оценки на наилучшую оценку, полученную от других процессов, затем запускается широковещательная рассылка обновленной оценки другим процессам. При этом для сокращения количества обменов между процессами отправка обновленной оценки производится лишь в том случае, когда новая оценка улучшила текущую более чем на некоторое пороговое значение $\mathcal{E} > 0$, являющееся параметром алгоритма. Коммуникации между процессами реализуются посредством технологии MPI.

3.4. Реализация

Параллельный алгоритм поиска самой похожей подпоследовательности временного ряда для вычислительного кластера с сопроцессорами Intel Xeon Phi представлен на рис. 3.

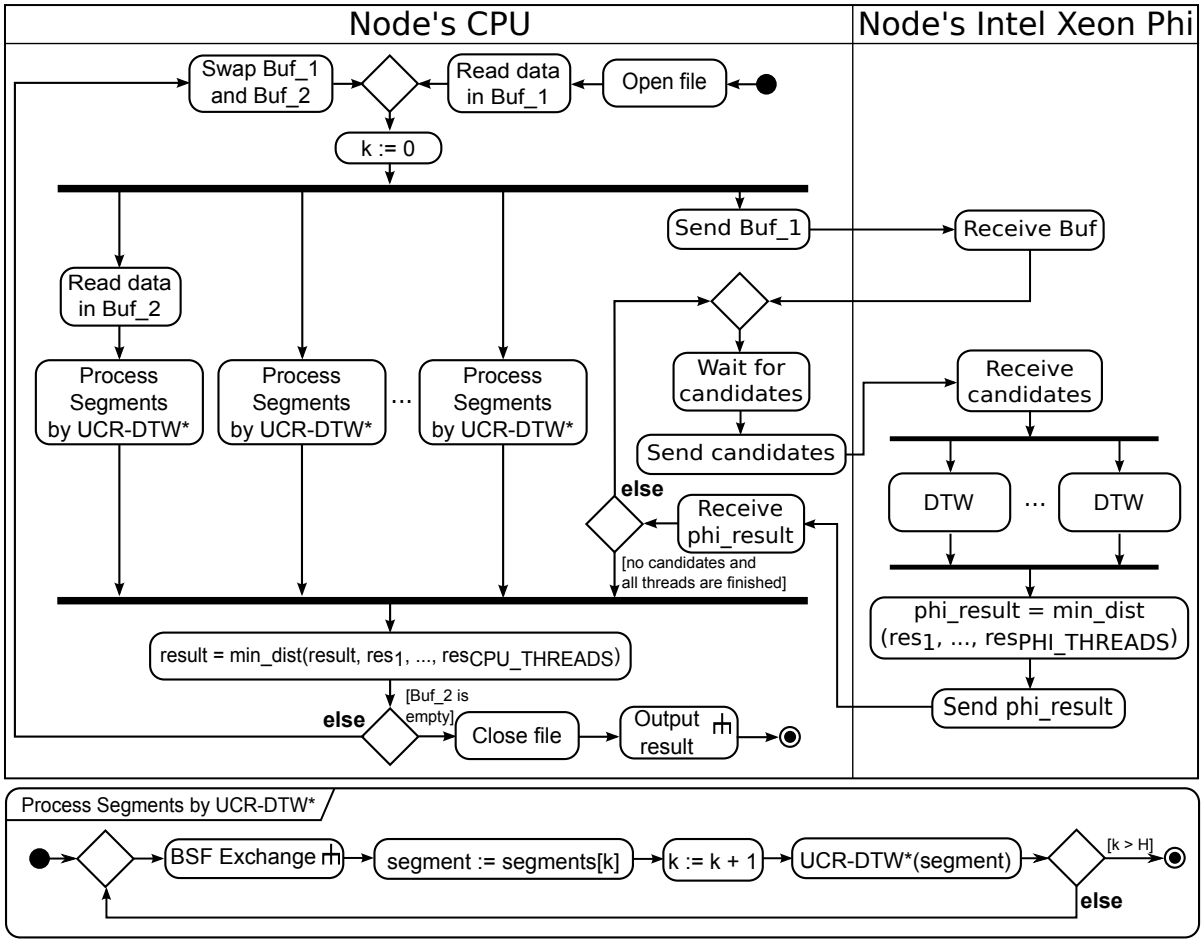


Рис. 3: Поиск самой похожей подпоследовательности на кластере с сопроцессорами

Процесс, запускаемый на узле вычислительного кластера, реализует данный алгоритм для обработки соответствующего фрагмента временного ряда. Процессор выполняет обработку сегментов фрагмента, реализуя модификацию разработанного ранее алгоритма [7]. Процессор поддерживает очередь подпоследовательностей-кандидатов, выгружаемых на сопроцессор, который выполняет для них вычисление меры DTW (деятельность `DTW`). Про-

цессор выполняет вычисление каскада оценок меры DTW для отбрасывания непохожих подпоследовательностей и подготовку данных для выгрузки на сопроцессор (деятельность `UCR-DTW*`). Распараллеливание осуществляется на основе технологии OpenMP, для чего временной ряд разбивается на сегменты равной длины. Для увеличения эффективности отбрасывания непохожих подпоследовательностей перед обработкой сегмента осуществляется обновление оценки схожести на основе алгоритма, описанного в разделе 3.3 (деятельность `BSF Exchange`). Обновление оценки схожести реализуется с помощью технологии MPI.

Результирующая подпоследовательность заданного временного ряда находится следующим образом. Один из процессов (например, с номером 0) объявляется мастером, остальные — рабочими. Рабочие отправляют мастеру результат вычислений соответствующего фрагмента, после чего мастер находит среди полученных результатов наилучший и выдает его пользователю.

4. Эксперименты

Для оценки эффективности разработанного алгоритма нами выполнены эксперименты на суперкомпьютере «Торнадо ЮУрГУ», спецификации вычислительного узла которого представлены в табл. 1.

Таблица 1: Спецификация вычислительного узла суперкомпьютера «Торнадо ЮУрГУ»

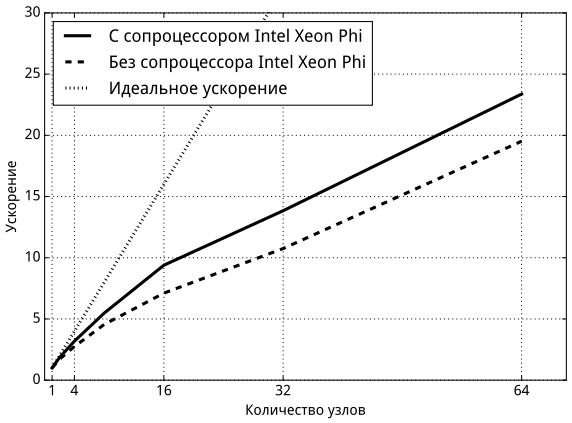
Спецификации	Процессор	Сопроцессор
Модель	Intel Xeon X5680	Intel Xeon Phi SE10X
Количество ядер	6	61
Тактовая частота, ГГц	3.33	1.1
Количество нитей на ядро	2	4
Пиковая производительность, TFLOPS	0.371	1.076

В экспериментах было задействовано от 1 до 64 вычислительных узлов кластера. В качестве данных в экспериментах использовался синтетический временной ряд, полученный на основе модели случайных блужданий [8]. В экспериментах исследовались ускорение и расширяемость предложенного алгоритма и его версии, которая не задействует сопроцессор (т.е. выполняет вычисления меры DTW полностью на центральном процессоре). Также выполнено сравнение ускорения данного алгоритма и его версии с аналогичными разработками.

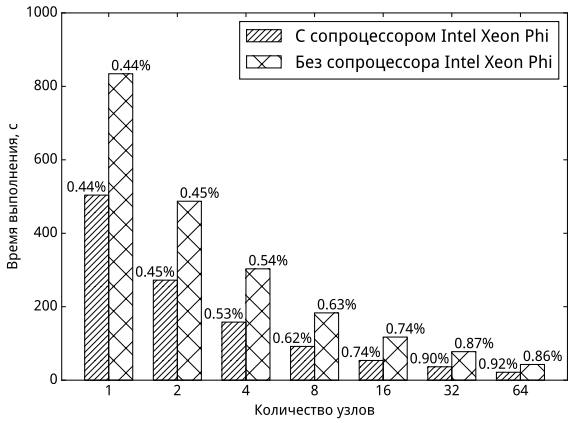
4.1. Ускорение и расширяемость алгоритма

В первой серии экспериментов исследовалось *ускорение* алгоритма, т.е. значение t_1/t_P, где t_1 и t_P — время работы алгоритма на одном и P узлах вычислительного кластера соответственно при одной и той же длине временного ряда, подвергаемого фрагментации и распределяемого по узлам. Использовались следующие значения параметров эксперимента: длина временного ряда $N = 8{\cdot}10^8$ (размер данных 6 Гб), длина запроса $n = 4000$, длина сегмента $L = 10^6$, пороговое значение улучшения оценки $\mathcal{E} = 0.01$. Результаты экспериментов представлены на рис. 4.

Во второй серии экспериментов исследовалась *расширяемость* алгоритма, т.е. значение t_{1T_1}/t_{PT_P}, где t_{1T_1} и t_{PT_P} — время обработки алгоритмом временного ряда T_1 на одном узле и временного ряда T_P на P узлах вычислительного кластера соответственно, где длина



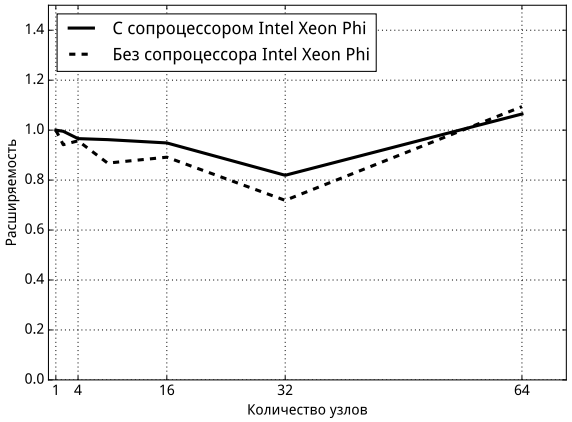
(a) Ускорение



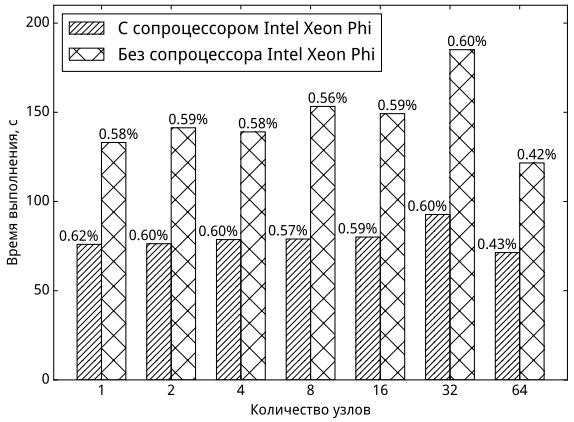
(b) Время выполнения

Рис. 4: Исследование ускорения алгоритма

T_P превосходит длину ряда T_1 в P раз. Использовались следующие значения параметров эксперимента: длина ряда T_1 — 10^8 (размер данных — от 0.7 Гб до 47.7 Гб), длина запроса $n = 4000$, длина сегмента $L = 10^6$, пороговое значение улучшения оценки $\mathcal{E} = 0.01$. Результаты экспериментов представлены на рис. 5.



(a) Расширяемость



(b) Время выполнения

Рис. 5: Исследование расширяемости алгоритма

Проценты над столбцами графиков указывают долю подпоследовательностей соответствующего временного ряда, для которых осуществлялось вычисление меры DTW (т.е. подпоследовательности, которые не были отброшены как заведомо непохожие). Как можно заметить, в экспериментах по исследованию ускорения доля подпоследовательностей, которые не были отброшены как заведомо непохожие, возрастает с увеличением количества узлов, что снижает ускорение. Это можно объяснить следующим образом. Поскольку узлы кластера обрабатывают фрагменты временного ряда одновременно и независимо друг от друга, то количество подпоследовательностей, для которых производится вычисление меры DTW, до обнаружения значительного улучшения оценки `bsf` возрастает. В результате при увеличении количества узлов возрастает общее количество подпоследовательностей, которые не были отброшены как заведомо непохожие.

Во второй серии экспериментов расширяемость остается примерно на уровне единицы, однако на 64 узлах наблюдается расширяемость больше единицы. Это объясняется умень-

шением доли подпоследовательностей, для которых производится вычисление меры DTW. Данное уменьшение связано с тем, что для исследования расширяемости на 64 узлах был задействован синтетический ряд в 2 раза большей длины, чем в экспериментах на 32 узлах, который содержал большее количество подпоследовательностей, позволивших улучшить оценку `bsf` раньше.

4.2. Сравнение с аналогами

Имеющиеся на сегодня параллельные реализации алгоритма UCR-DTW ориентированы на многоядерные ускорители GPU [10,13,17,18] (с возможным совместным использованием в вычислениях центрального процессора и графического ускорителя [6]) и FPGA [10, 16]. Эксперименты, проведенные на предыдущем этапе исследования показали [7], что разработанный авторами параллельный алгоритм для процессора и сопроцессора Intel Xeon Phi способен дать большую эффективность, чем аналоги для GPU.

Работы, в которых предлагаются параллельные реализации рассматриваемой задачи на платформе вычислительного кластера с узлами на базе многоядерных ускорителей, в настоящее время, по-видимому, отсутствуют. Относительно релевантной работой является статья [11], в которой описано распараллеливание алгоритма UCR-DTW для вычислительного кластера с узлами без ускорителей. Распараллеливание выполнено на основе использования фреймворка Apache Spark [12]. В рамках данного фреймворка приложение запускается как процесс, координируемый мастер-узлом вычислительного кластера, на котором установлен соответствующий драйвер. Алгоритм предполагает фрагментацию временного ряда с перекрытием (как и алгоритм, описанный в данной работе), однако количество фрагментов не совпадает с количеством узлов вычислительного кластера. Фрагменты сохраняются в виде отдельных файлов, доступных всем узлам в силу использования распределенной файловой системы HDFS (Hadoop Distributed File System). Каждый фрагмент обрабатывается отдельным процессом, реализующим последовательный алгоритм UCR-DTW. Количество процессов, запускаемых на узле кластера, равно количеству ядер процессора на данном узле. Алгоритм использует обновление оценки `bsf`, которое выполняется следующим образом. Один из процессов объявляется мастером, остальные — рабочими. Мастер хранит наилучшую из локальных оценок рабочих. По завершении обработки фрагментов рабочие пересылают локальные оценки мастеру, который выбирает из этих оценок наилучшую и рассылает ее рабочим.

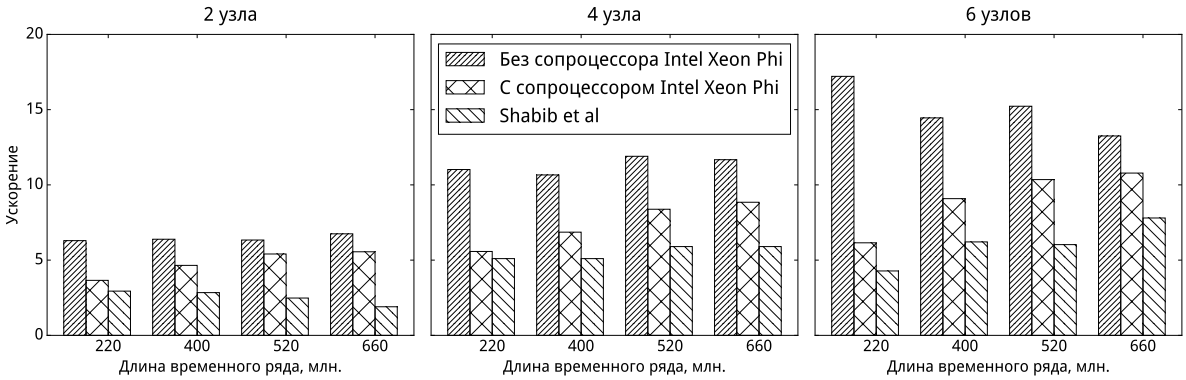


Рис. 6: Сравнение алгоритма с аналогом

В статье [11] приведены результаты экспериментов по исследованию ускорения разработанного авторами статьи алгоритма на 2, 4 и 6 узлах. Эксперименты проводились на узлах с 4-ядерными процессорами, на каждом узле выполнялось по 4 процесса, реализующих алгоритм UCR-DTW, длина запроса $n = 128$, в качестве данных использовался синтетический

временной ряд, полученный на основе модели случайных блужданий. Ускорение считалось по отношению к последовательному алгоритму UCR-DTW.

Нами были проведены эксперименты по сравнению разработанных нами алгоритмов с алгоритмом из статьи [11]. Для создания честных условий сравнения на нашей вычислительной системе использовались 4 нити на узел, каждая из которых выполняла поиск похожих подпоследовательностей, и сравнивалось ускорение алгоритмов. Результаты экспериментов представлены на рис. 6.

Разработанные нами алгоритмы показали большее ускорение по сравнению с алгоритмом из статьи [11] во всех случаях. Однако алгоритм, использующий сопроцессор, показал меньшее ускорение, чем его версия, не использующая сопроцессор. Это объясняется тем, что алгоритм, использующий сопроцессор, показывает наибольшую эффективность при длине запроса больше 10^3 [7].

5. Заключение

В данной статье описаны проектирование и реализация параллельного алгоритма поиска самой похожей подпоследовательности временного ряда для вычислительной системы с кластерной архитектурой, узлы которой оснащены многоядерными ускорителями Intel Xeon Phi.

Алгоритм предполагает три уровня параллелизма по данным. На первом уровне временной ряд разбивается на фрагменты равной длины, распределяемые по узлам вычислительного кластера. В ходе обработки вычислительные узлы обмениваются информацией о найденном значении меры DTW для подпоследовательности, которая на текущий момент является самой похожей на запрос. Для реализации данного вида параллелизма используется технология MPI. Второй уровень параллелизма предполагает разбиение фрагмента на сегменты равной длины, обрабатываемые нитями на основе технологии OpenMP. Третий уровень параллелизма заключается в балансировке вычислительной нагрузки между процессором и ускорителем. Процессор выполняет вычисление каскада оценок меры DTW для отбрасывания заведомо непохожих подпоследовательностей. Сопроцессор используется для вычисления меры DTW. Разбиение данных на порции (фрагменты и сегменты) выполняется с использованием техники перекрытия, предотвращающей потерю результирующих подпоследовательностей, которые могут находиться на стыке порций.

Представлены результаты вычислительных экспериментов, показывающих эффективность разработанного алгоритма и его превосходство над известными аналогами.

Литература

1. Berndt D.J., Clifford J. Using Dynamic Time Warping to Find Patterns in Time Series // Proceedings of the 1994 AAAI Workshop on Knowledge Discovery in Databases, Seattle, Washington, July 1994. AAAI Press, 1994. P. 359–370.

2. 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.

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

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

5. Heinecke A., Klemm M., Pfluger D., Bode A., Bungartz H.-J. Extending a Highly Parallel Data Mining Algorithm to the Intel Many Integrated Core Architecture // Proceedings of the Euro-Par 2011 Workshops, Bordeaux, France, August 29 – September 2, 2011. Lecture Notes in Computer Science. 2012. Vol. 7156. Springer, 2011. P. 375–384.

6. Huang S., Dai G., Sun Y., Wang Z., Wang Y., Yang H. DTW-Based Subsequence Similarity Search on AMD Heterogeneous Computing Platform // Proceedings of the 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. IEEE Computer Society, 2013. P. 1054–1063.

7. 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 Computer Society, 2015. P. 1399–1404.

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

9. 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.

10. 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.

11. Shabib A., Narang A., Niddodi C.P., Das M., Pradeep R., Shenoy V., Auradkar P., Vignesh T.S., Sitaram D. Parallelization of Searching and Mining Time Series Data Using Dynamic Time Warping // Proceedings of the International Conference on Advances in Computing, Communications and Informatics, ICACCI 2015, Kochi, India, August 10–13, 2015. IEEE Computer Society, 2015. P. 343–348.

12. Shanahan J.G., Dai L. Large Scale Distributed Data Science using Apache Spark // Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Sydney, NSW, Australia, August 10–13, 2015. ACM, 2015. P. 2323–2324.

13. Srikanthan S., Kumar A., Gupta R. Implementing the Dynamic Time Warping Algorithm in Multithreaded Environments for Real Time and Unsupervised Pattern Discovery // Proceedings of the Computer and Communication Technology (ICCCT) conference, Allahabad, India, September 15–17, 2011. IEEE Computer Society, 2011. P. 394–398.

14. Takahashi N., Yoshihisa T., Sakurai Y., Kanazawa M. A Parallelized Data Stream Processing System Using Dynamic Time Warping Distance // Proceedings of the 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.

15. 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. IEEE Computer Society, 2013. P 18:1–18:10.

16. 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.

17. Xiao L., Zheng Y., Tang W., Yao G., Ruan L. Parallelizing Dynamic Time Warping Algorithm Using Prefix Computations on GPU // Proceedings of the 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. IEEE Computer Society, 2013. P. 294–299.

18. 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 Computer Society, 2012. P. 5173–5176.

Parallel implementation of searching the most similar subsequence in time series for computer systems with distributed memory

A.V. Movchan, M.L. Zymbler

South Ural State University (Chelyabinsk, Russia)

The paper presents parallel implementation of searching the most similar subsequence in time series for computer cluster system with nodes based on Intel MIC accelerators. The algorithm involves three levels of data parallelism. The first level provides partitioning of time series into equal-length fragments, each of which is processed on a separate node of the computer cluster; nodes interact using MPI technology. The second level of parallelism supposes division of the fragment into equal-length segments and processing of each segment by a separate thread by means of OpenMP technology. The third level provides load balancing between CPU and accelerator. CPU performs pruning of dissimilar subsequences. Accelerator performs heavy-weighted calculations of similarity measure. The results of experiments confirm the efficiency of algorithm.

Keywords: time series data mining, computer cluster, Intel Xeon Phi coprocessor, OpenMP, MPI, dynamic time warping.

References

1. Berndt D.J., Clifford J. Using Dynamic Time Warping to Find Patterns in Time Series // Proceedings of the 1994 AAAI Workshop on Knowledge Discovery in Databases, Seattle, Washington, July 1994. AAAI Press, 1994. P. 359–370.

2. 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.

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

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

5. Heinecke A., Klemm M., Pfluger D., Bode A., Bungartz H.-J. Extending a Highly Parallel Data Mining Algorithm to the Intel Many Integrated Core Architecture // Proceedings of the Euro-Par 2011 Workshops, Bordeaux, France, August 29 – September 2, 2011. Lecture Notes in Computer Science. 2012. Vol. 7156. Springer, 2011. P. 375–384.

6. Huang S., Dai G., Sun Y., Wang Z., Wang Y., Yang H. DTW-Based Subsequence Similarity Search on AMD Heterogeneous Computing Platform // Proceedings of the 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. IEEE Computer Society, 2013. P. 1054–1063.

7. 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 Computer Society, 2015. P. 1399–1404.

8. Pearson K. The Problem of the Random Walk // Nature. 1905. Vol. 72, No. 1865. P. 294.
9. 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.
10. 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.
11. Shabib A., Narang A., Niddodi C.P., Das M., Pradeep R., Shenoy V., Auradkar P., Vignesh T.S., Sitaram D. Parallelization of Searching and Mining Time Series Data Using Dynamic Time Warping // Proceedings of the International Conference on Advances in Computing, Communications and Informatics, ICACCI 2015, Kochi, India, August 10–13, 2015. IEEE Computer Society, 2015. P. 343–348.
12. Shanahan J.G., Dai L. Large Scale Distributed Data Science using Apache Spark // Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Sydney, NSW, Australia, August 10–13, 2015. ACM, 2015. P. 2323–2324.
13. Srikanthan S., Kumar A., Gupta R. Implementing the Dynamic Time Warping Algorithm in Multithreaded Environments for Real Time and Unsupervised Pattern Discovery // Proceedings of the Computer and Communication Technology (ICCCT) conference, Allahabad, India, September 15–17, 2011. IEEE Computer Society, 2011. P. 394–398.
14. Takahashi N., Yoshihisa T., Sakurai Y., Kanazawa M. A Parallelized Data Stream Processing System Using Dynamic Time Warping Distance // Proceedings of the 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.
15. 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. IEEE Computer Society, 2013. P 18:1–18:10.
16. 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.
17. Xiao L., Zheng Y., Tang W., Yao G., Ruan L. Parallelizing Dynamic Time Warping Algorithm Using Prefix Computations on GPU // Proceedings of the 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. IEEE Computer Society, 2013. P. 294–299.
18. 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 Computer Society, 2012. P. 5173–5176.