УДК 004.272.25; 004.421; 004.032.24 doi 10.26089/NumMet.v20r104

# СОВМЕСТНОЕ ИСПОЛЬЗОВАНИЕ ТЕХНОЛОГИЙ MPI и OpenMP ДЛЯ ПАРАЛЛЕЛЬНОГО ПОИСКА ПОХОЖИХ ПОДПОСЛЕДОВАТЕЛЬНОСТЕЙ В СВЕРХБОЛЬШИХ ВРЕМЕННЫХ РЯДАХ НА ВЫЧИСЛИТЕЛЬНОМ КЛАСТЕРЕ С УЗЛАМИ НА БАЗЕ МНОГОЯДЕРНЫХ ПРОЦЕССОРОВ INTEL XEON PHI KNIGHTS LANDING

**Я. А. Краева[1], М. Л. Цымблер[2]**

В настоящее время поиск похожих подпоследовательностей требуется в широком спектре приложений интеллектуального анализа временных рядов: моделирование климата, финансовые прогнозы, медицинские исследования и др. В большинстве указанных приложений при поиске используется мера схожести Dynamic Time Warping (DTW), поскольку на сегодняшний день научное сообщество признает меру DTW одной из лучших для большинства предметных областей. Мера DTW имеет квадратичную вычислительную сложность относительно длины искомой подпоследовательности, в силу чего разработан ряд параллельных алгоритмов ее вычисления на устройствах FPGA и многоядерных ускорителях с архитектурами GPU и Intel MIC. В настоящей статье предлагается новый параллельный алгоритм для поиска похожих подпоследовательностей в сверхбольших временных рядах на кластерных системах с узлами на базе многоядерных процессоров Intel Xeon Phi поколения Knights Landing (KNL). Вычисления распараллеливаются на двух уровнях: на уровне всех узлов кластера — с помощью технологии MPI и в рамках одного узла кластера — с помощью технологии OpenMP. Алгоритм предполагает использование дополнительных структур данных и избыточных вычислений, позволяющих эффективно задействовать возможности векторизации вычислений на процессорных системах Phi KNL. Эксперименты, проведенные на синтетических и реальных наборах данных, показали хорошую масштабируемость алгоритма.

**Ключевые слова:** временной ряд, поиск похожих подпоследовательностей, параллельный алгоритм, OpenMP, Intel Xeon Phi, Knights Landing, представление данных в памяти, векторизация вычислений.

**1. Введение.** В настоящее время поиск похожих подпоследовательностей требуется в широком спектре приложений интеллектуального анализа временных рядов: моделирование климата, финансовые прогнозы, медицинские исследования и др. Задача поиска похожей подпоследовательности неформально может быть определена следующим образом. Пусть имеется временной ряд и поисковый запрос (временной ряд, длина которого существенно меньше длины исходного ряда). Необходимо найти подпоследовательность исходного ряда, которая максимально похожа по форме на поисковый запрос.

На сегодняшний день мера *DTW (Dynamic Time Warping, динамическая трансформация временной шкалы)* [4] признается научным сообществом лучшей мерой схожести временных рядов для многих предметных областей [6], поскольку позволяет адекватно измерять схожесть временных рядов, которые имеют различные длины либо сдвиг, растяжение или сжатие относительно друг друга. Поскольку мера *DTW* имеет квадратичную вычислительную сложность относительно длины поискового запроса, были предложены методы снижения вычислительной нагрузки: индексирование [9], заблаговременный отказ от избыточных вычислений, повторное использование результатов ранее выполненных вычислений [21] и др. Последовательный алгоритм *UCR-DTW* [18], интегрирующий множество указанных техник, является на сегодня, вероятно, самым быстрым последовательным алгоритмом поиска похожих подпоследовательностей [22]. В то же время предложены параллельные алгоритмы поиска похожих подпоследовательностей для устройств FPGA [29, 30], многоядерных ускорителей архитектур GPU [22, 30] и Intel Many Integrated Core (MIC) [31].

Рассматриваемые в настоящей статье многоядерные системы архитектуры Intel MIC являются конкурентоспособной альтернативой более распространенным системам NVIDIA GPU и FPGA. Системы Intel

[1] Южно-Уральский государственный университет, факультет вычислительной математики и информатики, просп. Ленина, 76, 454080, Челябинск; программист, e-mail: kraevaya@susu.ru

[2] Южно-Уральский государственный университет, факультет вычислительной математики и информатики, просп. Ленина, 76, 454080, Челябинск; начальник отдела, e-mail: mzym@susu.ru

© Научно-исследовательский вычислительный центр МГУ им. М. В. Ломоносова

MIC основаны на распространенной архитектуре Intel x86 и поддерживают соответствующие модели и инструменты параллельного программирования. Устройства MIC обеспечивают большое количество энергоэффективных вычислительных ядер с малой (относительно обычных многоядерных процессоров Intel) тактовой частотой, обладающих высокой пропускной способностью локальной памяти и 512-битными векторными регистрами. В настоящее время компания Intel предлагает два поколения MIC-устройств, имеющих базовое имя Xeon Phi: сопроцессор Knights Corner (KNC) [5] с 57–61 ядрами и самостоятельная процессорная система Knights Landing (KNL) [23] с 64–72 ядрами соответственно. Системы MIC, как правило, дают наибольшую производительность в таких приложениях, где имеет место большой объем данных, вовлеченных в вычисления (десятки миллионов элементов и более), которые могут быть векторизованы компилятором [24]. Под векторизацией понимается способность компилятора заменить несколько скалярных операций в теле цикла с фиксированным количеством повторений в одну векторную операцию [3].

Параллельный поиск похожих подпоследовательностей в сверхбольших временных рядах (порядка сотен миллиардов точек) на кластерных системах рассмотрен в работах [16, 25]. В статье [25] предложен способ распараллеливания последовательного алгоритма поиска похожих подпоследовательностей *UCR-DTW* для вычислительного кластера на основе использования фреймворка Apache Spark. В рамках этого фреймворка приложение запускается как задача, координируемая мастер-узлом вычислительного кластера, на котором установлен соответствующий драйвер. Алгоритм предполагает фрагментацию временного ряда, однако количество фрагментов не совпадает с количеством вычислительных узлов кластерной системы. Фрагменты сохраняются в виде отдельных файлов, доступных всем узлам в силу использования распределенной файловой системы HDFS (Hadoop Distributed File System). Каждый фрагмент обрабатывается отдельной нитью, реализующей последовательный алгоритм *UCR-DTW*. Количество процессов, запускаемых на узле кластера, равно количеству ядер процессора на данном узле. В экспериментах разработанный алгоритм опережает *UCR-DTW* в 2–5 раз, однако представленные результаты его запусков на кластере из 2–6 узлов не позволяют говорить о хорошей масштабируемости решения.

В предыдущих исследованиях [16, 31] авторов настоящей статьи разработан параллельный алгоритм поиска похожих подпоследовательностей на вычислительном кластере с узлами на базе сопроцессоров Phi KNC. Предложенная вычислительная схема “CPU+Phi” предполагает фрагментацию ряда по узлам кластера и разделение функций между процессором и KNC внутри узла по следующей схеме. Сопроцессор KNC выполняет вычисление меры *DTW* для подпоследовательностей, загруженных в его память процессором, и выбирает подпоследовательность, максимально похожую на образец поиска. Процессор исключает из обработки заведомо непохожие на поисковый запрос подпоследовательности и на основе вычисления нижних оценок формирует пакеты подпоследовательностей для выгрузки данных на сопроцессор. Предложенная схема вычислений обеспечивает высокую производительность, однако слабо задействует возможности векторизации вычислений Phi KNC. Масштабируемость, близкая к линейной, достигается алгоритмом в случае, когда поисковый запрос имеет длину от 4000 точек, что относительно редко встречается на практике. Помимо этого, данная схема не может быть применена для случая кластерной системы на базе узлов с системами следующего поколения Phi, поскольку KNL является независимым многоядерным устройством, запускающим приложения только в нативном режиме. В работе [11] авторами предложен подход к эффективному поиску похожих подпоследовательностей в оперативной памяти процессора Phi KNL.

В настоящей статье предлагается новый параллельный алгоритм *PhiBestMatch* для поиска похожих подпоследовательностей в сверхбольших временных рядах на кластерных системах с узлами на базе многоядерных процессоров Intel Xeon Phi поколения Knights Landing. Вычисления распараллеливаются на двух уровнях: на уровне всех узлов кластера — с помощью технологии MPI и в рамках одного узла кластера — с помощью технологии OpenMP. Алгоритм предполагает использование дополнительных структур данных и избыточных вычислений, позволяющих эффективно задействовать возможности векторизации вычислений на процессорных системах Phi KNL.

Статья организована следующим образом. В разделе 2 приводится формальная постановка задачи и дано краткое описание последовательного алгоритма *UCR-DTW*, используемого в качестве основы нового параллельного алгоритма. Раздел 3 содержит описание алгоритма *PhiBestMatch*. В разделе 4 описаны вычислительные эксперименты, исследующие эффективность предложенного алгоритма. Заключение резюмирует результаты, полученные в рамках исследования.

**2. Постановка задачи.**

**2.1. Формальные определения и обозначения.** Дадим определения используемых терминов в соответствии с работой [18]. *Временной ряд (time series)* $T$ — это хронологически упорядоченная после-

довательность вещественных значений: $T = (t_1, t_2, \ldots, t_m)$, $t_i \in \mathbb{R}$. Число $m$ обозначается $|T|$ и называется длиной временного ряда.

*Мерой схожести (similarity measure)* $\mathcal{D}$ между двумя временными рядами $X$ и $Y$ называется вещественная неотрицательная функция, вычисляющая расстояние между этими рядами: $\mathcal{D}(X, Y) \geqslant 0$.

Пусть даны два временных ряда $X = (x_1, x_2, \ldots, x_m)$ и $Y = (y_1, y_2, \ldots, y_m)$. Тогда *динамической трансформацией временной шкалы (Dynamic Time Warping, DTW)* называется мера схожести $DTW(X, Y)$, вычисляемая следующим образом [4]:

$$DTW(X,Y) = d(m,m), \quad d(i,j) = (x_i - y_j)^2 + \min \begin{cases} d(i-1,j), \\ d(i,j-1), \\ d(i-1,j-1), \end{cases} \tag{1}$$
$$d(0,0) = 0, \quad d(i,0) = d(0,j) = \infty, \quad 1 \leqslant i \leqslant m, \quad 1 \leqslant j \leqslant m.$$

Мера $DTW$ позволяет сравнивать ряды, которые смещены вдоль временной оси, сжаты, растянуты или имеют разные длины. Далее нами будет рассматриваться случай, когда временные ряды имеют одинаковые длины, поскольку это упрощает изложение и не ограничивает общность [18].

В формуле (1) матрица $(d_{ij}) \in \mathbb{R}^{m \times m}$ выражает соответствие между точками сравниваемых временных рядов и называется *матрицей трансформации.*

*Путь трансформации (warping path)* $P$ представляет собой последовательность элементов матрицы трансформации, которая определяет соответствие между временными рядами $Q$ и $C$. Пусть элемент $p_t$ пути трансформации $P$ определяется как $p_t = (i, j)_t$, тогда

$$P = p_1, p_2, \ldots, p_t, \ldots, p_T, \quad m \leqslant T \leqslant 2n - 1.$$

На путь трансформации накладывается ряд ограничений. Путь должен начинаться и заканчиваться в диагонально противоположных элементах матрицы трансформации, шаги пути ограничены соседними ячейками, а точки пути должны быть монотонно разнесены во времени.

Объем вычислений при подсчете меры схожести $DTW$ может быть уменьшен за счет огрубления схожести. Для этого на путь трансформации могут налагаться дополнительные ограничения. Одним из наиболее часто применяемых ограничений является *полоса Сако–Чиба* [20], не позволяющая пути трансформации отклоняться более чем на $r$ элементов от диагонали матрицы трансформации.

*Подпоследовательностью (subsequence)* $T_{i,n}$ временного ряда $T$ называется непрерывное подмножество $T$, состоящее из $n$ элементов и начинающееся с позиции $i$: $T_{i,n} = (t_i, t_{i+1}, \ldots, t_{i+n-1})$, $1 \leqslant i \leqslant m-n+1$, $k \ll m$.

Определим решаемую в настоящей статье *задачу поиска самой похожей подпоследовательности* ряда в смысле меры $DTW$. Пусть имеется длинный временной ряд $T$ и заданный пользователем существенно более короткий временной ряд $Q$ (называемый *поисковым запросом (query)*), где $m = |T| \gg |Q| = n$. Тогда *самой похожей подпоследовательностью (best match)* назовем такую подпоследовательность $T_{i,n}$, форма которой максимально похожа на запрос $Q$ в смысле меры $DTW$:

$$\exists i \quad \forall k \quad DTW(Q, T_{i,n}) \leqslant DTW(Q, T_{k,n}), \quad 1 \leqslant i, k \leqslant m - n + 1.$$

Далее для краткости изложения алгоритма подпоследовательность $T_{i,n}$ будем называть *кандидатом* (на максимальную схожесть с запросом $Q$) и обозначать как $C$.

**2.2. Последовательный алгоритм.** В настоящее время алгоритм *UCR-DTW* [18] является одним из самых быстрых алгоритмов поиска похожих подпоследовательностей и включает в себя много способов, ускоряющих поиск. Поскольку предлагаемый параллельный алгоритм использует *UCR-DTW* в качестве основы, ниже кратко описаны его основные идеи.

*Использование квадрата евклидова расстояния. Евклидово расстояние (ED)* между двумя временными рядами $Q$ и $C$, где $|Q| = |C|$, определяется следующим образом:

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

Вычисление квадратного корня в $ED$ (2) и $DTW$ (1) можно опустить, так как это не изменит относительного ранжирования подпоследовательностей, похожих на запрос, поскольку обе функции монотонные и вогнутые.

*Z-нормализация.* Перед вычислением меры схожести запрос и подпоследовательность временного ряда необходимо подвергнуть z-нормализации [28]. *Z-нормализацией* временного ряда $T$ называется временной ряд $\widehat{T} = \left(\widehat{t}_1, \ldots, \widehat{t}_m\right)$, элементы которого вычисляются следующим образом:

$$\widehat{t}_i = \frac{t_i - \mu}{\sigma}, \quad 1 \leqslant i \leqslant m; \quad \mu = \frac{1}{m}\sum_{i=1}^{m} t_i; \quad \sigma^2 = \frac{1}{m}\sum_{i=1}^{m} t_i^2 - \mu^2.$$

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

*Каскадное применение нижних границ схожести.* Нижняя граница схожести (*lower bound, LB*) представляет собой функцию, вычислительная сложность которой меньше вычислительной сложности меры $DTW$. Нижняя граница используется для отбрасывания кандидатов, заведомо непохожих на запрос, без вычисления меры $DTW$ [6].

Обозначим через *bsf (best-so-far)* лучшую текущую нижнюю оценку схожести текущей подпоследовательности $T_{i,n}$ и поискового запроса $Q$. Если нижняя граница для кандидата превышает порог *bsf*, то значение меры $DTW$ для данного кандидата тоже превысит *bsf* и кандидат заведомо не похож на запрос. В процессе сканирования временного ряда алгоритм *UCR-DTW* пытается улучшить (уменьшить) значение $bsf$. Значение *bsf* инициализируется значением $+\infty$ и на $i$-м шаге поиска вычисляется следующим образом:

$$bsf_{(i)} = \min\left(bsf_{(i-1)}, \begin{cases} +\infty, & LB(Q, T_{i,n}) > bsf_{(i-1)}, \\ DTW(Q, T_{i,n}), & \text{otherwise} \end{cases}\right).$$

Алгоритм *UCR-DTW* использует следующие нижние границы схожести: $LB_{\text{Kim}}FL$ [18], $LB_{\text{Keogh}}EC$ и $LB_{\text{Keogh}}EQ$ [9], которые применяются каскадным образом.

Нижняя граница $LB_{\text{Kim}}FL$ представляет собой евклидово расстояние между первой и последней парами точек $Q$ и $C$:

$$LB_{\text{Kim}}FL(Q, C) = ED\left(\widehat{q}_1, \widehat{c}_1\right) + ED(\widehat{q}_n, \widehat{c}_n).$$

Нижняя граница $LB_{\text{Keogh}}EC$ показывает минимальную схожесть между оболочкой запроса $E$ *(envelope)* и кандидатом $\widehat{C}$ и вычисляется следующим образом:

$$LB_{\text{Keogh}}EC(Q, C) = \sum_{i=1}^{n} \begin{cases} \left(\widehat{c}_i - u_i\right)^2 & \text{if} \quad \widehat{c}_i > u_i, \\ \left(\widehat{c}_i - \ell_i\right)^2 & \text{if} \quad \widehat{c}_i < \ell_i, \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

В формуле (3) последовательности $U = (u_1, \ldots, u_n)$ и $L = (\ell_1, \ldots, \ell_n)$ обозначают *верхнюю (upper)* и *нижнюю (lower) границы оболочки* запроса $Q$, которые вычисляются по формуле

$$u_i = \max\left(\widehat{q}_{i-r}, \ldots, \widehat{q}_{i+r}\right), \quad \ell_i = \min\left(\widehat{q}_{i-r}, \ldots, \widehat{q}_{i+r}\right).$$

Нижняя граница $LB_{\text{Keogh}}EQ$ представляет собой евклидово расстояние между запросом $Q$ и оболочкой кандидата $C$, т.е. по сравнению с $LB_{\text{Keogh}}EC$ роли запроса и кандидата меняются местами:

$$LB_{\text{Keogh}}EQ(Q, C) := LB_{\text{Keogh}}EC(C, Q).$$

Выполнение алгоритма *UCR-DTW* происходит следующим образом. Сначала нормализуется запрос и вычисляется его оболочка, порог $bsf$ инициализируется значением $\infty$. Далее алгоритм сканирует временной ряд от начала до конца, применяя к текущей подпоследовательности каскад нижних границ схожести. Если подпоследовательность не была отброшена, то вычисляется расстояние $DTW$. Далее значение $bsf$ заменяется на вычисленное значение меры $DTW$, если последнее меньше лучшей текущей оценки схожести. Таким образом, по окончании сканирования ряда алгоритм находит самую похожую подпоследовательность.

**3. Параллельный алгоритм PhiBestMatch.** В данном разделе представлен новый параллельный алгоритм поиска самой похожей подпоследовательности временного ряда *PhiBestMatch* для кластерных систем с узлами на базе многоядерных процессорных систем архитектуры Intel MIC. Распараллеливание

алгоритма основано на следующих основных принципах: параллелизм по данным, выравнивание данных в памяти и векторизация вычислений.

Параллелизм по данным *на уровне вычислительных узлов кластерной системы* реализуется с помощью фрагментации временного ряда. Временной ряд разбивается на фрагменты (подпоследовательности) примерно равной длины, каждый из которых размещается в оперативной памяти отдельного вычислительного узла, выполняющего обработку данного фрагмента. Такая схема предполагает следующее *масштабирование*: если суммарный объем оперативной памяти недостаточен для размещения временного ряда, то в кластер добавляется дополнительный узел (узлы).

Для обработки фрагмента на каждом узле запускается вычислительный процесс, и все процессы используют один и тот же алгоритм. В ходе обработки процессы обмениваются данными для сокращения объема вычислений. Обмены данными между процессами реализуются с помощью технологии MPI [14]. Параллелизм по данным *внутри узла кластерной системы* реализуется следующим образом. В рамках вычислительного процесса на ядрах процессора узла запускаются (*fork*) нити, разделяющие оперативную память узла. Нити осуществляют параллельную обработку фрагмента на базе технологии OpenMP [14].

*Векторизация циклов* является одним из ключевых условий достижения высокой производительности вычислительных программ на параллельных архитектурах [3]. Векторизация циклов заключается в преобразовании компилятором последовательности скалярных операторов из тела цикла в один векторный оператор. Таким образом, чтобы повысить производительность поиска похожих подпоследовательностей на Phi KNL, необходимо организовывать вычисления таким образом, чтобы увеличить в алгоритме, насколько это возможно, количество векторизуемых циклов.

Эффективность векторизации циклов, однако, может быть существенно снижена в силу не выровненного доступа к данным в оперативной памяти, который порождает эффект разделения цикла (loop peeling) [3]. Если начальный адрес массива не выровнен на ширину векторного регистра (т.е. на количество элементов, которые могут быть загружены в векторный регистр), то компилятор разбивает цикл на три части. Первая часть итераций, которые обращаются к памяти с начального адреса до первого выровненного адреса, и третья часть итераций с последнего выровненного адреса до конечного адреса векторизуются отдельно.

В соответствии с этим в алгоритме *PhiBestMatch* предлагается компоновка данных в оперативной памяти, которая обеспечивает выровненный доступ к подпоследовательностям временного ряда, и соответствующая этой компоновке вычислительная схема, в которой вычисления реализованы в виде векторизуемых циклов.

**3.1. Фрагментация временного ряда.** Фрагментация временного ряда обеспечивает параллелизм по данным на уровне вычислительных узлов кластерной системы и осуществляется следующим образом. Для предотвращения потери результирующих подпоследовательностей, находящихся на стыке фрагментов, предлагается техника *разбиения с перекрытием*, которая заключается в следующем. В конец каждого фрагмента временного ряда, за исключением последнего по порядку, добавляется $n-1$ элементов ряда, взятых с начала следующего фрагмента, где $n$ — длина запроса. Формальное определение разбиения с перекрытием выглядит следующим образом.

Пусть $N = |T| - n + 1 = m - n + 1$ — количество подпоследовательностей, которые необходимо обработать, $F$ — количество фрагментов, $T^{(k)}$ — $k$-й фрагмент временного ряда $T = (t_1, t_2, \ldots, t_m)$, где $0 \leqslant k \leqslant F-1$. Тогда *фрагмент* $T^{(k)}$ определяется как подпоследовательность $T_{\text{start,len}}$, где

$$\text{start} = k \cdot \left\lfloor \frac{N}{F} \right\rfloor + 1, \quad \text{len} = \begin{cases} \left\lfloor \dfrac{N}{F} \right\rfloor + (N \bmod F) + n - 1, & k = F-1, \\ \left\lfloor \dfrac{N}{F} \right\rfloor + n - 1 & \text{otherwise.} \end{cases}$$

Количество фрагментов (вычислительных узлов кластерной системы) $F$ выбирается таким образом, чтобы фрагмент и соответствующие вспомогательные данные алгоритма могли быть размещены в оперативной памяти вычислительного узла.

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

Выравнивание данных выполняется следующим образом.

Пусть обработка подпоследовательности $T_{i,n}$ ряда $T$ осуществляется с использованием векторного регистра, вмещающего $w$ вещественных чисел. Если длина подпоследовательности не кратна $w$, то под-

последовательность дополняется фиктивными нулевыми элементами. Обозначим количество фиктивных элементов через $pad = w - (n \bmod w)$, тогда выровненная подпоследовательность $\widetilde{T}_{i,n}$ определяется следующим образом:

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

Согласно определению (1), $\forall Q,\, C : DTW(Q, C) = DTW\left(\widetilde{Q}, \widetilde{C}\right)$. В дальнейшем изложении мы предполагаем, что запрос и подпоследовательности-кандидаты выровнены в памяти, а для упрощения используем обозначения $Q$ и $C$ соответственно. Выравнивание подпоследовательностей позволяет избежать накладных расходов, связанных с разбиением циклов на несколько итераций (loop peeling).

Все (выровненные) подпоследовательности временного ряда сохраняются в виде матрицы подпоследовательностей для обеспечения векторизации вычислений.

*Матрица подпоследовательностей* $S_T^n \in \mathbb{R}^{N \times (n+pad)}$ определяется следующим образом:

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

Обозначим количество нижних границ схожести, используемых в алгоритме поиска самой похожей подпоследовательности, через $lb_{\max}$ ($lb_{\max} \geqslant 1$); через $LB_1$, $LB_2$, …, $LB_{lb_{\max}}$ обозначим эти границы, перечисленные в порядке их вычисления в каскаде применения нижних оценок. Тогда *матрица* $L_T^n \in \mathbb{R}^{N \times lb_{\max}}$ *нижних границ схожести* всех подпоследовательностей длины $n$ временного ряда $T$ с поисковым запросом $Q$ имеет следующий вид:

$$L_T^n(i, j) := LB_j(T_{i,n}, Q).$$

*Карта схожести* представляет собой вектор-столбец $B_T^n \in \mathbb{B}^N$, который для каждой подпоследовательности длины $n$ ряда $T$ хранит конъюнкцию применения всех нижних границ схожести этой подпоследовательности с текущим значением порога $bsf$:

$$B_T^n(i) := \bigwedge_{j=1}^{lb_{\max}} \left(L_T^n(i, j) < bsf\right).$$

Введем матрицу для хранения подпоследовательностей-кандидатов, т.е. тех подпоследовательностей из матрицы $S_T^n$, которые не были отброшены при применении нижней границы схожести. Данная матрица будет обрабатываться параллельно для вычисления значений меры $DTW$ между кандидатами и запросом. Далее минимальное из вычисленных значение меры схожести $DTW$ используется в качестве порога $bsf$.

Параллельная обработка матрицы кандидатов использует разбиение строк этой матрицы на сегменты, обрабатываемые отдельными нитями одного узла кластера.

Обозначим число нитей, используемых параллельным алгоритмом, через $p$ ($p \geqslant 1$). Введем *размер сегмента* $s \in \mathbb{N}$, $s \leqslant \left\lceil \dfrac{N}{p} \right\rceil$ — количество строк матрицы кандидатов, обрабатываемых одной нитью. Тогда *матрица кандидатов* $C_T^n \in \mathbb{R}^{(s \cdot p) \times (n+pad)}$ определяется следующим образом:

$$C_T^n(i, \cdot) := S_T^n(k, \cdot) : B_T^n(i) = \texttt{TRUE}.$$

Несмотря на то что вычисление матрицы кандидатов является распараллеливаемой операцией, векторизация операторов соответствующего цикла затруднена, поскольку в соответствии с определением (1) при вычислении меры схожести $DTW$ имеет место зависимость по данным.

**3.3. Вычислительная схема алгоритма.** Предлагаемая в нашей работе параллельная реализация поиска похожих подпоследовательностей временного ряда представлена в виде приведенного ниже алгоритма и на рис. 1. Алгоритм выполняется следующим образом.

Для *инициализации* вычислений алгоритм определяет номер текущего вычислительного процесса $myrank$ с помощью функций библиотеки MPI. В дальнейшем в рамках алгоритма каждый вычислительный процесс с номером $myrank$ обрабатывает матрицу подпоследовательностей $S_{T^{(myrank)}}^n$ фрагмента $T^{(myrank)}$ исходного временного ряда $T$. Переменная $bsf$ инициализируется значением меры схожести $DTW$ между поисковым запросом и случайной подпоследовательностью из данного фрагмента временного ряда.

**Алгоритм.** PhiBestMatch(in $T, Q, r$; out $bsf, bestmatch$)

1: $myrank \leftarrow$ MPI_Comm_rank()
2: $N \leftarrow \left|T^{(myrank)}\right| - n + 1$
3: $subseq_{rnd} \leftarrow T^{(myrank)}_{random(1..N),n}$
4: $bsf \leftarrow \mathrm{DTW}(subseq_{rnd}, Q, r, \infty)$
5: $processed \leftarrow N$
6: ПОДГОТОВИТЬ
7: **repeat**
8: УЛУЧШИТЬ($T^{(myrank)}$, $bsf$, $bestmatch$)
9: $flagDone \leftarrow (processed = 0)$
10: $\{bsf, bestmatch\} \leftarrow$ MPI_Allreduce($\{bsf, bestmatch\}$, MPI_FLOAT_LONG, MPI_MIN)
11: $Stop \leftarrow$ MPI_Allreduce($myFragDone$, MPI_BOOL, MPI_AND)
12: **until not** $Stop$
13: **return** $\{bsf, bestmatch\}$

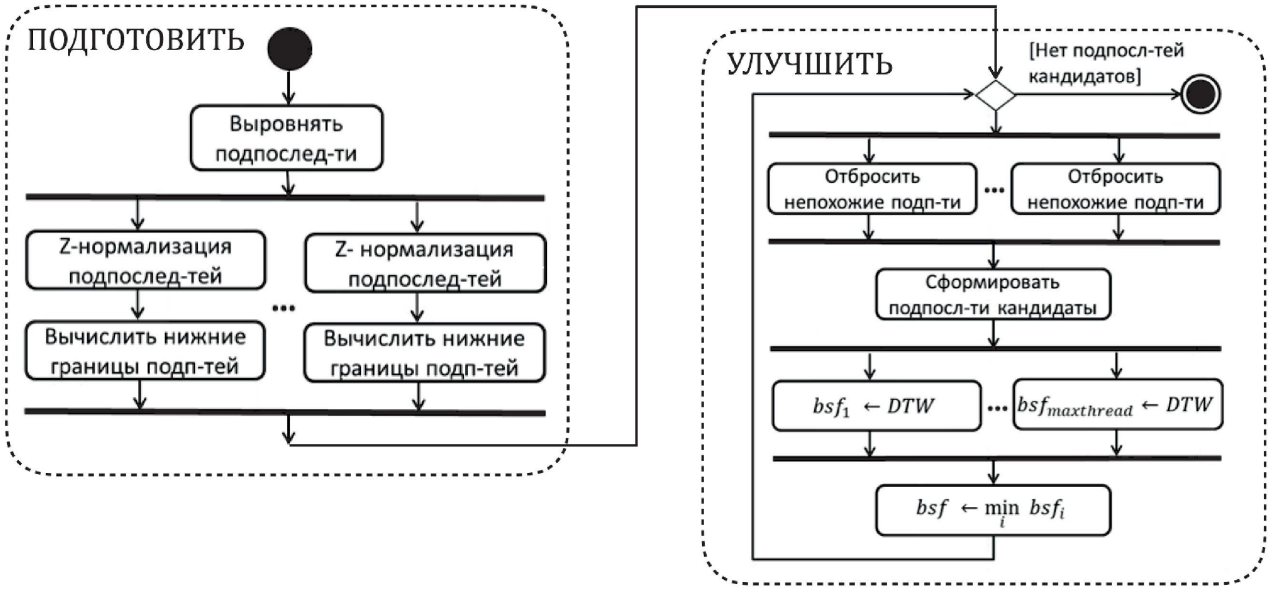



Рис. 1. Схема вычислений алгоритма на одном узле кластера

Далее выполняется *подготовка данных* для вычислений. Формируется матрица выровненных подпоследовательностей. В цикле, выполняемом по строкам матрицы подпоследовательностей, каждая подпоследовательность $z$-нормализуется и выполняются вычисления строк матрицы нижних границ схожести. Следует обратить внимание, что, строго говоря, предварительное вычисление матрицы нижних границ схожести представляет собой избыточные накладные расходы. В этом состоит ключевое отличие от алгоритма *UCR-DTW*, где нижние границы схожести вычисляются каскадом (следующая нижняя граница вычисляется только в случае, если с помощью предыдущей границы не выявлено, что текущий кандидат не является заведомо не похожим на запрос). Тем не менее, эти предвычисления обоснованы тем, что выполняются однократно и могут быть реализованы с помощью распараллеливаемых и векторизуемых компилятором циклов.

Распараллеливание данного цикла осуществляется директивой OpenMP `#pragma omp parallel for`, обеспечивающей статическое разбиение итераций цикла между нитями. Выровненные данные в матрице подпоследовательностей и отсутствие зависимостей по данным в вычислительных формулах нижних границ схожести обеспечивают векторизацию соответствующих вычислений.

После этого алгоритм выполняет следующий цикл действий, направленный на *улучшение* значения порога *bsf*. Цикл выполняется, пока каждый вычислительный узел не завершит обработку своего фрагмента. Сначала вычисляется карта схожести на основе предвычисленной матрицы нижних границ схожести.

Полученное значение элемента карты схожести `FALSE` означает, что соответствующий элемент матрицы подпоследовательностей является заведомо непохожим на образец поиска и может быть отброшен без вычисления меры схожести $DTW$. В противном случае этот элемент добавляется в матрицу кандидатов для последующего вычисления меры схожести $DTW$.

Для распараллеливания описанных выше операций матрица нижних границ схожести разбивается на сегменты (по числу используемых алгоритмом нитей). По построению матрица нижних границ схожести имеет существенно большее количество строк, чем матрица кандидатов. После заполнения и вычисления матрицы кандидатов и обновления порога $bsf$ каждая нить должна продолжить сканирование своего сегмента вплоть до его исчерпания. Для хранения номера последнего обработанного кандидата в сегменте вводится индексный массив $Pos \in \mathbb{N}^p$, где

$$pos_i := k : p \cdot (i-1) + 1 \leqslant k \leqslant \left\lceil \frac{N}{i \cdot p} \right\rceil \ \wedge \quad \forall j, \quad 1 \leqslant j \leqslant lb_{\max}, \quad LB_T^n(k,j) < bsf.$$

Отсутствие кандидатов означает, что обработка фрагмента закончена. В противном случае выполняется подсчет меры схожести $DTW$ для каждой строки матрицы кандидатов.

Чтобы вывести индекс самой похожей на запрос подпоследовательности, введен индексный массив $Idx \in \mathbb{N}^{s \cdot p}$, который предназначен для хранения позиции подпоследовательности во временном ряде и определяется следующим образом:

$$idx_i := k : 1 \leqslant k \leqslant N \wedge \exists S_T^n(i,\cdot) \Leftrightarrow \exists T_{i,n} \Leftrightarrow k = (i-1) \cdot n + 1.$$

После того как матрица кандидатов заполнена, для каждой ее строки вычисляется значение меры схожести $DTW$ между запросом и кандидатом. Распараллеливание цикла осуществляется с помощью директивы OpenMP `#pragma omp parallel for`, обеспечивающей статическое разбиение итераций цикла между нитями. Если вычисленное значение схожести меньше, чем текущее значение $bsf$, то $bsf$ заменяется на вычисленное значение.

Выбор наилучшей среди всех фрагментов ряда нижней оценки схожести и соответствующей подпоследовательности осуществляется с помощью операции глобальной редукции `MPI_Allreduce` стандарта MPI, которая возвращает минимальное значение порога $bsf$ среди всех вычислительных процессов и соответствующую подпоследовательность, копируя их в память каждого процесса. Факт завершения обработки ряда также определяется с помощью операции глобальной редукции, в которой выполняется конъюнкция флагов завершения обработки каждым процессом своего фрагмента ряда.

Таблица 1

Аппаратная платформа экспериментов

| Характеристика | Хост | Сопроцессор |
|---|---|---|
| Модель, Intel Xeon | X5680 | Phi (KNC), SE10X |
| Количество физических ядер | 2×6 | 61 |
| Гиперпоточность | 2× | 4× |
| Количество логических ядер | 24 | 244 |
| Частота, ГГц | 3.33 | 1.1 |
| Размер VPU, бит | 128 | 512 |
| Пиковая производительность, TFLOPS | 0.371 | 1.076 |

**4. Вычислительные эксперименты.**

**4.1. Аппаратная платформа.** Для исследования эффективности разработанного алгоритма были проведены вычислительные эксперименты. В качестве аппаратной платформы экспериментов использовались узлы кластерной системы “Торнадо ЮУрГУ” [10], характеристики которых приведены в табл. 1.

В экспериментах исследована эффективность алгоритма *PhiBestMatch* как на одном вычислительном узле кластерной системы, так и на кластерной системе в целом.

Исследование на одном вычислительном узле кластерной системы позволяет определить, насколько эффективно реализованы параллельные вычисления, выполняемые многоядерной процессорной системой Intel Xeon Phi. Для такого исследования использовалась упрощенная версия [11] приведенного

выше алгоритма, в которой ряд отождествлен с одним фрагментом, вызовы коммуникационной библиотеки MPI не задействуются и цикл обработки подпоследовательностей ряда продолжается до исчерпания подпоследовательностей-кандидатов.

Во всех экспериментах размер сегмента матрицы кандидатов, т.е. количество строк матрицы кандидатов, обрабатываемых одной нитью в рамках одного вычислительного узла кластерной системы, имеет значение $s = 100$.

Таблица 2

Наборы данных для экспериментов на одном узле кластера

| Набор данных | Вид | $\|T\| = m$ | $\|Q\| = n$ |
|---|---|---|---|
| Random Walk | Синтетический | $10^6$ | 128 |
| EPG | Реальный | $2.5 \times 10^5$ | 360 |

**4.2. Эффективность алгоритма на одном узле кластера.** Исследование эффективности алгоритма на одном вычислительном узле кластера “Торнадо ЮУрГУ” производилось на наборах данных, которые представлены в табл. 2.

Временной ряд Random Walk получен искусственно на основе модели случайных блужданий [17]. Проведение экспериментов по исследованию свойств алгоритмов поиска подпоследовательностей на базе меры схожести $DTW$ с использованием подобных временных рядов является общепринятой практикой (см., например, работы [22, 25, 29]).

Временной ряд EPG (Electrical Penetration Graph, график электрического проникновения) использован в экспериментах в работе [22]. EPG представляет собой набор сигналов, используемых энтомологами для сравнения поведения исследуемого насекомого с поведением цикадок *macrosteles quadrilineatus*, которые являются переносчиками болезней растений и ежегодно наносят сельскому хозяйству США ущерб более чем на 2 млн долларов [22].

В экспериментах исследовались производительность и масштабируемость алгоритма *PhiBestMatch*. Под производительностью понимается время работы алгоритма без учета времени загрузки данных в память и выдачи результата. Масштабируемость параллельного алгоритма означает его способность адекватно адаптироваться к увеличению параллельно работающих вычислительных элементов (процессов, процессоров, нитей и др.) и характеризуется ускорением и параллельной эффективностью, которые определяются следующим образом [1]. *Ускорение* и *параллельная эффективность* параллельного алгоритма, запускаемого на $k$ нитях, вычисляются как $s(k) = \dfrac{t_1}{t_k}$ и $e(k) = \dfrac{s(k)}{k}$ соответственно, где $t_1$ и $t_k$ — время работы алгоритма на одной и $k$ нитях соответственно.

В экспериментах рассматривались вышеупомянутые показатели в зависимости от изменения параметра $r$ (ширина полосы Сако–Чиба), значения которого брались в долях длины поискового запроса $n$.

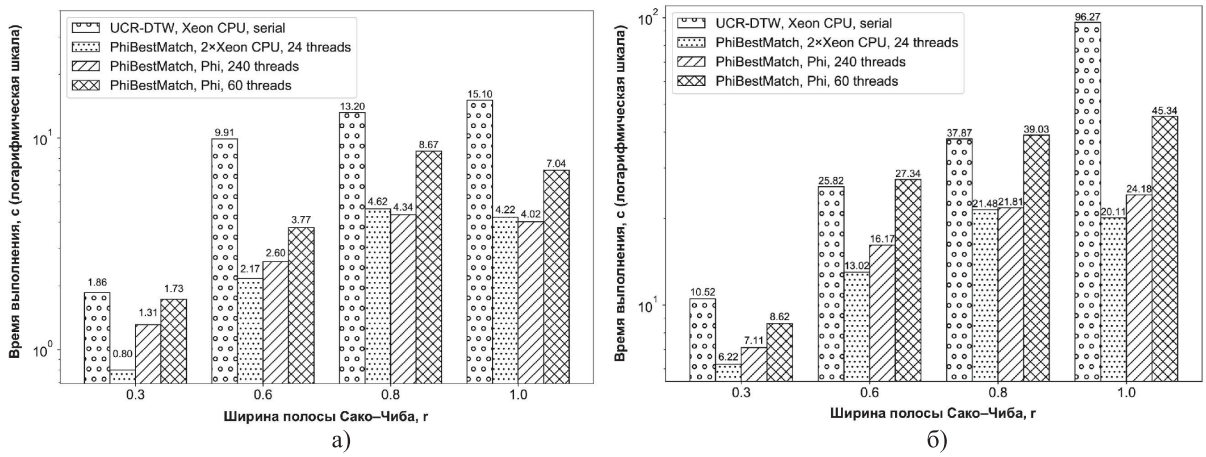
Время работы алгоритма *PhiBestMatch* на многоядерных платформах сравнивалось с временем работы алгоритма *UCR-DTW* [18].

Результаты экспериментов по исследованию производительности алгоритма представлены на рис. 2. Можно видеть, что *PhiBestMatch* работает до 5 раз быстрее, чем алгоритм *UCR-DTW*. Вместе с тем видно, что на производительность алгоритма *PhiBestMatch* на платформах двухпроцессорного узла Intel Xeon и многоядерной системы Intel Xeon Phi влияют следующие два параметра: ширина полосы Сако–Чиба $r$ и длина поискового запроса $n$.

При малых значениях этих параметров (примерно $0 < r \leqslant 0.5n$ и $n < 512$) алгоритм *PhiBestMatch* на платформе двухпроцессорного узла Intel Xeon работает несколько быстрее или примерно с тем же быстродействием, что и на платформе многоядерной системы Intel Xeon Phi. При бóльших значениях данных параметров ($0.5n < r \leqslant n$ и $n \geqslant 512$) алгоритм работает быстрее на платформе Intel Xeon Phi. Это означает, что заложенные в алгоритм при проектировании возможности векторизации наилучшим образом проявляются при увеличении общего объема вычислений.

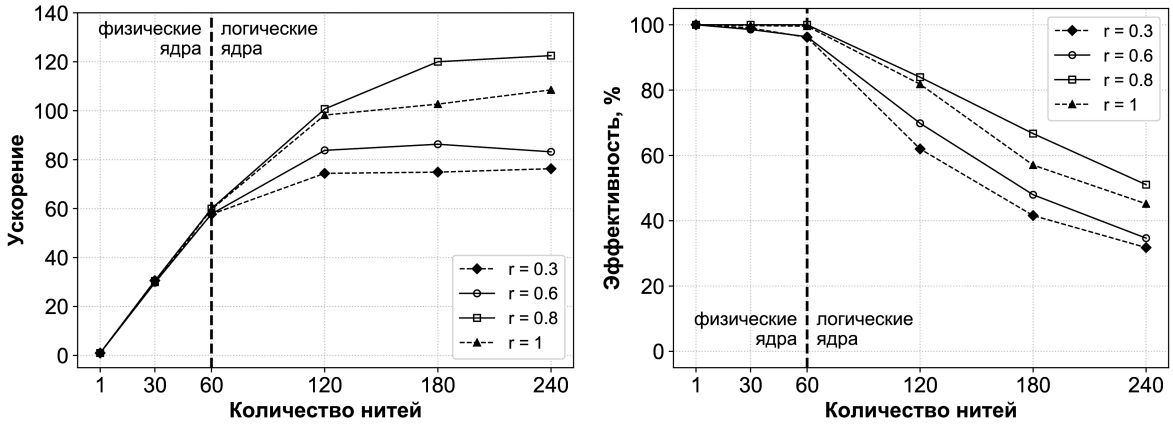
Поисковые запросы с длиной $n \geqslant 512$, равно как и значение параметра $r = 1$ требуются на практике в ряде приложений, требующих при определении схожести подпоследовательностей как можно более высокую точность, например: в медицине при исследовании ЭКГ [15], в энтомологии [2], в астрономии [19] и др.

Результаты экспериментов по исследованию масштабируемости алгоритма представлены на рис. 3 и 4. Можно видеть, что *PhiBestMatch* демонстрирует близкое к линейному ускорение и параллельную



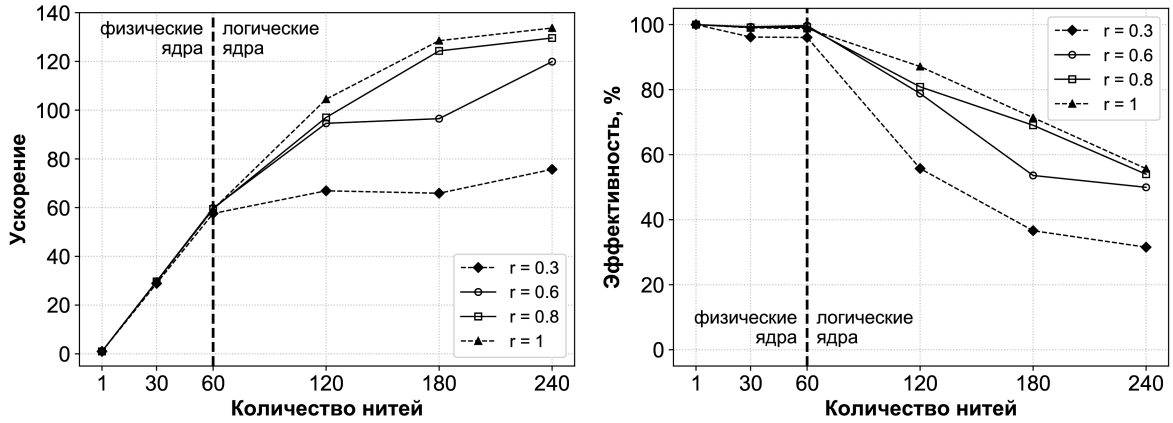

а) б)

Рис. 2. Производительность алгоритма *PhiBestMatch* на одном узле кластера: а) набор синтетических данных Random Walk, $m = 10^6$, $n = 128$; б) набор реальных данных EPG, $m = 2.5 \times 10^5$, $n = 360$




а) б)

Рис. 3. Масштабируемость алгоритма *PhiBestMatch* на одном узле кластера при обработке синтетических данных (ряд Random Walk, $m = 10^6$, $n = 128$): а) ускорение; б) параллельная эффективность




а) б)

Рис. 4. Масштабируемость алгоритма *PhiBestMatch* на одном узле кластера при обработке реальных данных (ряд EPG, $m = 2.5 \times 10^5, n = 360$): а) ускорение; б) параллельная эффективность

эффективность, близкую к 100%, если количество нитей, на которых запущен алгоритм, совпадает с количеством физических ядер системы Intel Xeon Phi.

При увеличении количества нитей, запускаемых на одном физическом ядре системы, ускорение становится сублинейным, равно как наблюдается и падение параллельной эффективности. При этом наилучшие показатели ускорения и параллельной эффективности ожидаемо наблюдаются при значениях параметра $r$ 0.8 и 1 от длины поискового запроса $n$, обеспечивающих алгоритму наибольшую вычислительную нагрузку. Например, при задействовании 240 нитей при $r = 0.8n$ обработка ряда Random Walk выполняется с ускорением 120 и параллельной эффективностью 50%; при $r = n$ обработка ряда EPG — с ускорением 130 и параллельной эффективностью 52%.

Полученные результаты позволяют сделать заключение о хорошей масштабируемости разработанного алгоритма и эффективном использовании им возможностей векторизации вычислений на многоядерной системе Intel Xeon Phi в рамках одного вычислительного узла кластерной системы. Указанные свойства проявляются при значениях параметров $r$ (ширина полосы Сако–Чиба) и $n$ (длина поискового запроса), обеспечивающих алгоритму наибольшую вычислительную нагрузку: $0.8n \leqslant r \leqslant n$ и $n \geqslant 512$ соответственно.

**4.3. Эффективность алгоритма на кластерной системе.** В исследовании эффективности алгоритма на кластерной системе в целом было задействовано от 16 до 128 вычислительных узлов суперкомпьютера “Торнадо ЮУрГУ” (см. табл. 1). Исследование производилось на наборах данных, которые представлены в табл. 3.

Таблица 3

Наборы данных для экспериментов на кластерной системе

| Набор данных | Вид | $\lvert T\rvert = m$ | $\lvert Q\rvert = n$ |
|---|---|---|---|
| Random Walk | Синтетический | $12.8 \times 10^7$ | 128, 512, 1024 |
| EPG | Реальный | $12.8 \times 10^7$ | 432, 512, 1024 |

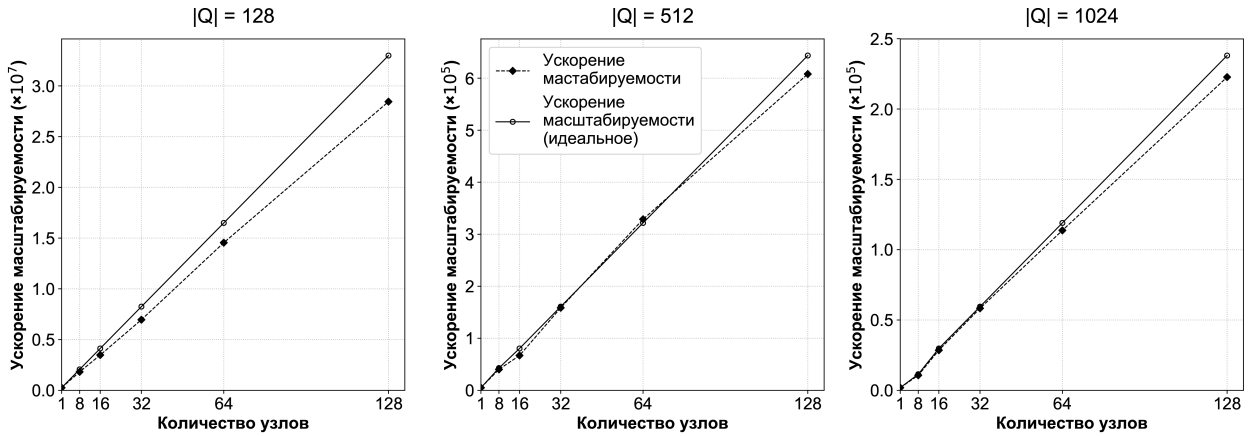



Рис. 5. Ускорение масштабируемости алгоритма *PhiBestMatch* на кластерной системе при обработке синтетических данных

В экспериментах исследовалось *ускорение масштабируемости (scaled speedup)* параллельного алгоритма, которое определяется как ускорение, демонстрируемое алгоритмом при линейном увеличении объема данных и количества используемых вычислительных узлов [12], и вычисляется следующим образом: $s_{scaled} = \dfrac{p \cdot m}{t_{p(p \cdot m)}}$, где $p$ — количество задействованных вычислительных узлов, $m$ — объем исходных данных, $t_{p(p \cdot m)}$ — время выполнения алгоритма на $p$ узлах при обработке исходных данных, имеющих объем $p \cdot m$.

При этом значение параметра $r$ было зафиксировано как $r = n$ и варьировалась длина поискового запроса.

Результаты экспериментов по исследованию ускорения масштабируемости алгоритма представлены на рис. 5 и 6.

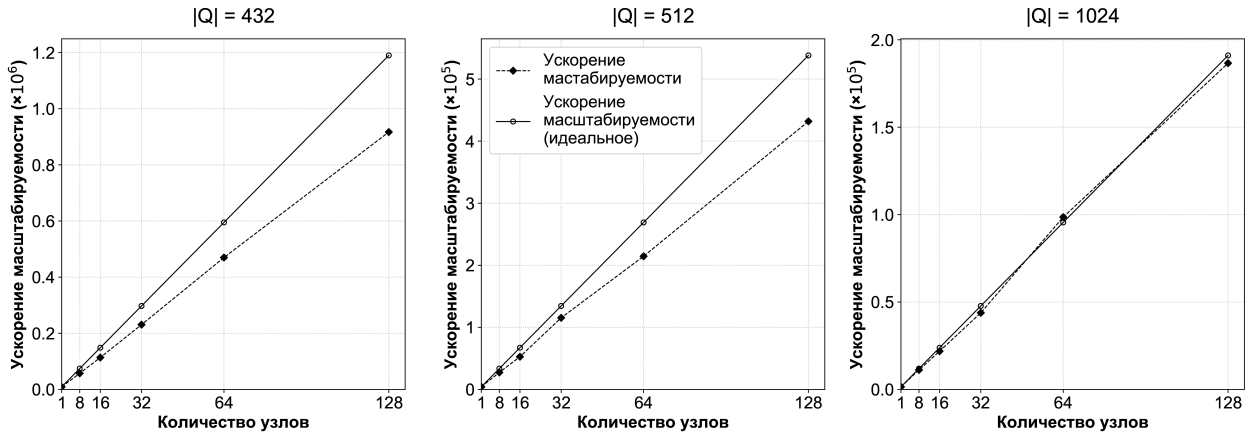



Рис. 6. Ускорение масштабируемости алгоритма *PhiBestMatch* на кластерной системе при обработке реальных данных

Можно видеть, что *PhiBestMatch* демонстрирует ускорение масштабируемости, близкое к линейному, как для синтетических, так и для реальных данных. При этом поиск подпоследовательности большей длины показывает более высокое ускорение масштабируемости, поскольку это обеспечивает больший объем вычислений в рамках одного вычислительного узла кластерной системы.

Полученные результаты позволяют сделать заключение о хорошей масштабируемости разработанного алгоритма при работе на кластерной системе с вычислительными узлами на базе многоядерных процессоров Intel Xeon Phi. Указанные свойства проявляются при значениях параметров $r$ (ширина полосы Сако–Чиба) и $n$ (длина поискового запроса), обеспечивающих алгоритму наибольшую вычислительную нагрузку: $0.8n \leqslant r \leqslant n$ и $n \geqslant 512$ соответственно.

**5. Заключение.** В настоящей статье рассмотрена проблема поиска похожих подпоследовательностей в сверхбольших временных рядах (сотни миллиардов точек) на основе использования меры схожести $DTW$ (динамическая трансформация шкалы времени). Данная задача возникает в широком спектре приложений интеллектуального анализа временных рядов: моделирование климата, финансовые прогнозы, медицинские исследования и др.

Предложен новый параллельный алгоритм для поиска похожих подпоследовательностей временного ряда на кластерной системе с вычислительными узлами на базе многоядерных процессоров Intel Xeon Phi поколения Knights Landing, названный *PhiBestMatch*. Алгоритм предполагает двухуровневое распараллеливание вычислений: на уровне всех узлов кластерной системы используется технология MPI, в рамках одного вычислительного узла кластера используется технология OpenMP. Для эффективного использования возможностей векторизации вычислений процессоров Phi KNL в рамках одного вычислительного узла используются дополнительные матричные структуры данных и избыточные вычисления. Проведены вычислительные эксперименты, исследующие быстродействие и масштабируемость алгоритма на синтетических и реальных наборах данных как на кластерной системе в целом, так и в рамках одного вычислительного узла кластера. Результаты экспериментов показали хорошую масштабируемость алгоритма *PhiBestMatch* на одном узле и на кластерной системе в целом.

Работа выполнена при финансовой поддержке РФФИ (проект № 17–07–00463), Правительства РФ в соответствии с Постановлением № 211 от 16.03.2013 (соглашение № 02.A03.21.0011) и Министерства образования и науки РФ (государственное задание 2.7905.2017/8.9).

СПИСОК ЛИТЕРАТУРЫ

1. *Воеводин В.В., Воеводин Вл.В.* Параллельные вычисления. СПб.: БХВ-Петербург, 2002.
2. *Athitsos V., Papapetrou P., Potamias M., et al.* Approximate embedding-based subsequence matching of time series // Proc. of the ACM SIGMOD Int. Conf. on Management of Data (Vancouver, Canada, June 10–12, 2008). New York: ACM Press, 2008. 365–378.
3. *Bacon D.F., Graham S.L., Sharp O.J.* Compiler transformations for high-performance computing // ACM Comput. Surv. 1994. **26**, N 4. 345–420.
4. *Berndt D.J., Clifford J.* Using dynamic time warping to find patterns in time series // Proc. of the 1994 AAAI Workshop on Knowledge Discovery in Databases (Seattle, Washington, July 31–August 1, 1994). Palo Alto: AAAI

Press, 1994. 359–370.
5. *Chrysos G.* Intel Xeon Phi Coprocessor (codename Knights Corner) // Proc. 2012 IEEE Hot Chips 24th Symposium (HCS) (Cupertino, CA, USA, August 27–29, 2012). New York: IEEE Press, 2012. doi 10.1109/HOTCHIPS.2012.7476487.
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 // J. Proc. VLDB Endowment. 2008. **1**, N 2. 1542–1552.
7. *Fu A.W., Keogh E., Lau L.Y.H., Ratanamahatana C.A., Wong R.C.* Scaling and time warping in time series querying // VLDB J. 2008. **17**, N 4. 899–921.
8. *Gropp W.* MPI 3 and beyond: why MPI is successful and what challenges it faces // Lecture Notes in Computer Science. Vol. 7490. Berlin: Springer, 2012. 1–9.
9. *Keogh E.J., Ratanamahatana C.A.* Exact indexing of dynamic time warping // Knowl. Inf. Syst. 2005. **7**, N 3. 358–386.
10. *Kostenetskiy P.S., Safonov A.Y.* SUSU supercomputer resources // Proc. of the 10th Annual Int. Scientific Conf. on Parallel Computing Technologies (PCT 2016). CEUR Workshop Proceedings. Vol. 1576. 2016. 561–573.
11. *Kraeva Ya., Zymbler M.* An efficient subsequence similarity search on modern Intel many-core processors for data intensive applications // Selected Papers of the XX Int. Conf. on Data Analytics and Management in Data Intensive Domains (DAMDID/RCDL 2018), Moscow, Russia, October 9–12, 2018. CEUR Workshop Proceedings. Vol. 2277. 2018. 143–151.
12. *Kumar V., Grama A., Gupta A., Karypis G.* Introduction to parallel computing. Redwood City: Benjamin/Cummings, 1994.
13. *Lim S., Park H., Kim S.* Using multiple indexes for efficient subsequence matching in time-series databases // Lecture Notes in Computer Science. Vol. 3882. Berlin: Springer, 2006. 65–79.
14. *Mattson T.* Introduction to OpenMP // Proc. of the 2006 ACM/IEEE Conf. on Supercomputing (Tampa, FL, USA, November 11–17, 2006). New York: ACM Press, 2006. doi 10.1145/1188455.1188673.
15. *Mazandarani F.N., Mohebbi M.* Wide complex tachycardia discrimination using dynamic time warping of ECG beats // Computer Methods and Programs in Biomedicine. 2018. **164**. 238–249.
16. *Movchan A.V., Zymbler M.L.* Parallel implementation of searching the most similar subsequence in time series for computer systems with distributed memory // Proc. of the 10th Annual Int. Scientific Conf. on Parallel Computing Technologies (PCT 2016). CEUR Workshop Proceedings. Vol. 1576. 2016. 615–628.
17. *Pearson K.* The problem of the random walk // Nature. 1905. **72**, N 1. 342. doi 10.1038/072342a0.
18. *Rakthanmanon T., Campana B.., Mueen A., Batista G.E.A.P.A., Westover M.B., et al.* Searching and mining trillions of time series subsequences under dynamic time warping // Proc. of the 18th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (Beijing, China, August 12–16, 2012). New York: ACM Press, 2012. 262–270.
19. *Rebbapragada U., Protopapas P., Brodley C.E., Alcock C.* Finding anomalous periodic time series // Machine Learning. 2009. **74**, N 3. 281–313.
20. *Sakoe H., Chiba S.* Readings in speech recognition. San Francisco: Morgan Kaufmann Publ., 1990. 159–165.
21. *Sakurai Y., Faloutsos C., Yamamuro M.* Stream monitoring under the time warping distance // Proc. of the 23rd Int. Conf. on Data Engineering (Istanbul, Turkey, April 15–20, 2007). Washington, DC: IEEE Press, 2007. 1046–1055.
22. *Sart D., Mueen A., Najjar W.A., Keogh E.J., Niennattrakul V.* Accelerating dynamic time warping subsequence search with GPUs and FPGAs // Proc. of the 2010 IEEE Int. Conf. on Data Mining (Sydney, Australia, December 14–17, 2010). Washington, DC: IEEE Press, 2010. 1001–1006.
23. *Sodani A.* Knights Landing (KNL): 2nd generation Intel Xeon Phi processor // 2015 IEEE Hot Chips 27th Symposium (HCS) (Cupertino, CA, USA, August 22–25, 2015). New York: IEEE Press, 2015. 1–24.
24. *Sokolinskaya I., Sokolinsky L.* Revised pursuit algorithm for solving non-stationary linear programming problems on modern computing clusters with manycore accelerators // Communications in Computer and Information Science. Vol. 687. Cham: Springer, 2016. 212–223.
25. *Shabib A., Narang A., Niddodi C.P., et al.* Parallelization of searching and mining time series data using Dynamic Time Warping // Proc. 2015 Int. Conf. on Advances in Computing, Communications and Informatics (Kochi, India, August 10–13, 2015). New York: IEEE Press, 2015. 343–348.
26. *Srikanthan S., Kumar A., Gupta R.* Implementing the dynamic time warping algorithm in multithreaded environments for real time and unsupervised pattern discovery // Proc. 2011 2nd Int. Conf. on Computer and Communication Technology (Allahabad, India, September 15–17, 2011). New York: IEEE Press, 2011. 394–398.
27. *Takahashi N., Yoshihisa T., Sakurai Y., Kanazawa M.* A parallelized data stream processing system using dynamic time warping distance // 2009 Int. Conf. on Complex, Intelligent and Software Intensive Systems (Fukuoka, Japan, March 16–19, 2009). New York: IEEE Press, 2009. 1100–1105.
28. *Tarango J., Keogh E.J., Brisk P.* Instruction set extensions for dynamic time warping // Proc. of the Int. Conf. on Hardware/Software Codesign and System Synthesis (Montreal, QC, Canada, September 29–October 4, 2013). Piscataway: IEEE Press, 2013. doi 10.1109/CODES-ISSS.2013.6659005.
29. *Wang Z., Huang S., Wang L., Li H., Wang Y., et al.* Accelerating subsequence similarity search based on dynamic time warping distance with FPGA // Proc. of the ACM/SIGDA Int. Symposium on Field Programmable Gate Arrays (Monterey, CA, USA, February 11–13, 2013). New York: ACM Press, 2013. 53–62.

30. *Zhang Y., Adl K., Glass J.R.* Fast spoken query detection using lower-bound dynamic time warping on graphical processing units // 2012 IEEE Int. Conf. on Acoustics, Speech and Signal Proc. (Kyoto, Japan, March 25–30, 2012). New York: IEEE Press, 2012. 5173–5176.
31. *Zymbler M.* Best-match time series subsequence search on the Intel Many Integrated Core architecture // Lecture Notes in Computer Science. Vol. 9282. Heidelberg: Springer, 2015. 275–286.

Поступила в редакцию
04.12.2018

---

## The Use of MPI and OpenMP Technologies for Subsequence Similarity Search in Very Long Time Series on a Computer Cluster System with Nodes Based on the Intel Xeon Phi Knights Landing Many-Core Processor

**Ya. A. Kraeva**[1] **and M. L. Zymbler**[2]

[1] *South Ural State University, Faculty of Computational Mathematics and Informatics; prospekt Lenina 76, Chelyabinsk, 454080, Russia; Programmer, e-mail: kraevaya@susu.ru*

[2] *South Ural State University, Faculty of Computational Mathematics and Informatics; prospekt Lenina 76, Chelyabinsk, 454080, Russia; Ph.D., Associate Professor, Head of Department, e-mail: mzym@susu.ru*

Received December 4, 2018

**Abstract:** Nowadays, the subsequence similarity search is required in a wide range of time series mining applications: climate modeling, financial forecasts, medical research, etc. In most of these applications, the Dynamic Time Warping (DTW) similarity measure is used, since DTW is empirically confirmed as one of the best similarity measures for the majority of subject domains. Since the DTW measure has a quadratic computational complexity with respect to the length of query subsequence, a number of parallel algorithms for various many-core architectures are developed, namely FPGA, GPU, and Intel MIC. In this paper we propose a new parallel algorithm for subsequence similarity search in very large time series on computer cluster systems with nodes based on Intel Xeon Phi Knights Landing (KNL) many-core processors. Computations are parallelized on two levels as follows: by MPI at the level of all cluster nodes and by OpenMP within a single cluster node. The algorithm involves additional data structures and redundant computations, which make it possible to efficiently use the capabilities of vector computations on Phi KNL. Experimental evaluation of the algorithm on real-world and synthetic datasets shows that the proposed algorithm is highly scalable.

**Keywords:** time series, similarity search, parallel algorithm, OpenMP, Intel Xeon Phi, Knights Landing, data layout, vectorization.

### References

1. V. V. Voevodin and Vl. V. Voevodin, *The Parallel Computing* (BHV-Petersburg, St. Petersburg, 2002).

2. V. Athitsos, P. Papapetrou, M. Potamias, et al., "Approximate Embedding-Based Subsequence Matching of Time Series," in *Proc. ACM SIGMOD Int. Conf. on Management of Data. Vancouver, Canada, June 10–12, 2008* (ACM Press, New York, 2008), pp. 365–378.

3. D. F. Bacon, S. L. Graham, and O. J. Sharp, "Compiler Transformations for High-Performance Computing," ACM Comput. Surv. **26** (4), 345–420 (1994).

4. D. J. Berndt and J. Clifford, "Using Dynamic Time Warping to Find Patterns in Time Series," in *Proc. 3rd Int. Conf. Knowledge Discovery and Data Mining, Seattle, USA. July 31–August 1, 1994* (AAAI Press, Palo Alto, 1994), pp. 359–370.

5. G. Chrysos, "Intel Xeon Phi Coprocessor (Codename Knights Corner)," in *Proc. 2012 IEEE Hot Chips 24th Symposium (HCS), Cupertino, USA, August 27–29, 2012* (IEEE Press, New York, 2012), doi 10.1109/HOTCHIPS.2012.7476487

6. H. Ding, G. Trajcevski, P. Scheuermann, et al., "Querying and Mining of Time Series Data: Experimental Comparison of Representations and Distance Measures," J. Proc. VLDB Endowment **1** (2), 1542–1552 (2008).

7. A. W. Fu, E. Keogh, L. Y. H. Lau, et al., "Scaling and Time Warping in Time Series Querying," VLDB J. **17** (4), 899–921 (2008).

8. W. Gropp, "MPI 3 and Beyond: Why MPI is Successful and What Challenges It Faces," in *Lecture Notes in Computer Science* (Springer, Berlin, 2012), Vol. 7490, pp. 1–9.

9. E. J. Keogh and C. A. Ratanamahatana, “Exact Indexing of Dynamic Time Warping,” Knowl. Inf. Syst. **7** (3), 358–386 (2005).

10. P. S. Kostenetskiy and A. Y. Safonov, “SUSU Supercomputer Resources,” in *Proc. 10th Annual Int. Scientific Conf. on Parallel Computing Technologies (PCT 2016), Arkhangelsk, Russia, March 29–31, 2016* CEUR Workshop Proc. **1576**, 561–573 (2016).

11. Ya. Kraeva and M. Zymbler, “An Efficient Subsequence Similarity Search on Modern Intel Many-Core Processors for Data Intensive Applications,” in *Selected Papers of XX Int. Conf. on Data Analytics and Management in Data Intensive Domains (DAMDID/RCDL 2018), Moscow, Russia, October 9–12, 2018.* CEUR Workshop Proc. Vol. 2277, 143–151 (2018).

12. V. Kumar, A. Grama, A. Gupta, and G. Karypis, *Introduction to Parallel Computing* (Benjamin/Cummings, Redwood City, 1994).

13. S. Lim, H. Park, and S. Kim, “Using Multiple Indexes for Efficient Subsequence Matching in Time-Series Databases,” in *Lecture Notes in Computer Science* (Springer, Berlin, 2006), Vol. 3882, pp. 65–79.

14. T. Mattson, “Introduction to OpenMP,” in *Proc. 2006 ACM/IEEE Conf. on Supercomputing, Tampa, USA, November 11–17, 2006* (ACM Press, New York, 2006), doi 10.1145/1188455.1188673

15. F. N. Mazandarani and M. Mohebbi, “Wide Complex Tachycardia Discrimination Using Dynamic Time Warping of ECG Beats,” Comput. Meth. Prog. Biomed. **164**, 238–249 (2018).

16. A. V. Movchan and M. L. Zymbler, “Parallel Implementation of Searching the Most Similar Subsequence in Time Series for Computer Systems with Distributed Memory,” in *Proc. 10th Annual Int. Scientific Conf. on Parallel Computing Technologies (PCT 2016), Arkhangelsk, Russia, March 29–31, 2016.* CEUR Workshop Proc. Vol. 1576, 615–628 (2016).

17. K. Pearson, “The Problem of the Random Walk,” Nature **72** (1), 342 (1905).

18. T. Rakthanmanon, B. Campana, A. Mueen, et al., “Searching and Mining Trillions of Time Series Subsequences under Dynamic Time Warping,” in *Proc. 18th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, Beijing, China, August 12–16, 2012* (ACM Press, New York, 2012), pp. 262–270.

19. U. Rebbapragada, P. Protopapas, C. E. Brodley, and C. Alcock, “Finding Anomalous Periodic Time Series,” Mach. Learn. **74** (3), 281–313 (2009).

20. H. Sakoe and S. Chiba, *Readings in Speech Recognition* (Morgan Kaufmann, San Francisco, 1990), pp. 159–165.

21. Y. Sakurai, C. Faloutsos, and M. Yamamuro, “Stream Monitoring under the Time Warping Distance,” in *Proc. 23rd Int. Conf. on Data Engineering, Istanbul, Turkey, April 15–20, 2007* (IEEE Press, Washington, DC, 2007), pp. 1046–1055.

22. D. Sart, A. Mueen, W. A. Najjar, et al., “Accelerating Dynamic Time Warping Subsequence Search with GPUs and FPGAs,” in *Proc. 2010 IEEE Int. Conf. on Data Mining, Sydney, Australia, December 14–17, 2010* (IEEE Press, Washington, DC, 2010), pp. 1001–1006.

23. A. Sodani, “Knights Landing (KNL): 2nd Generation Intel Xeon Phi Processor,” in *2015 IEEE Hot Chips 27th Symposium (HCS), Cupertino, CA, August 22–25, 2015* (IEEE Press, New York, 2015), pp. 1–24.

24. I. Sokolinskaya and L. Sokolinsky, “Revised Pursuit Algorithm for Solving Non-Stationary Linear Programming Problems on Modern Computing Clusters with Manycore Accelerators,” in *Communications in Computer and Information Science* (Springer, Cham, 2016), Vol. 687, pp. 212–223.

25. A. Shabib, A. Narang, C. P. Niddodi, et al., “Parallelization of Searching and Mining Time Series Data using Dynamic Time Warping,” in *Proc. 2015 Int. Conf. on Advances in Computing, Communications and Informatics, Kochi, India, August 10–13, 2015* (IEEE Press, New York, 2015), pp. 343–348.

26. S. Srikanthan, A. Kumar, and R. Gupta, “Implementing the Dynamic Time Warping Algorithm in Multithreaded Environments for Real Time and Unsupervised Pattern Discovery,” in *Proc. 2011 2nd Int. Conf. on Computer and Communication Technology, Allahabad, India, September 15–17, 2011* (IEEE Press, New York, 2011), pp. 394–398.

27. N. Takahashi, T. Yoshihisa, Y. Sakurai, and M. Kanazawa, “A Parallelized Data Stream Processing System Using Dynamic Time Warping Distance,” in *Proc. 2009 Int. Conf. on Complex, Intelligent and Software Intensive Systems, Fukuoka, Japan, March 16–19, 2009* (IEEE Press, New York, 2009), pp. 1100–1105.

28. J. Tarango, E. Keogh, and P. Brisk, “Instruction Set Extensions for Dynamic Time Warping,” in *Proc. Int. Conf. on Hardware/Software Codesign and System Synthesis, Montreal, Canada, September 29–October 4, 2013* (IEEE Press, Piscataway, 2013), doi 10.1109/CODES-ISSS.2013.6659005.

29. Z. Wang, S. Huang, L. Wang, et al., “Accelerating Subsequence Similarity Search Based on Dynamic Time Warping Distance with FPGA,” in *Proc. ACM/SIGDA Int. Symp. on Field Programmable Gate Arrays, Monterey, USA, February 11–13, 2013* (ACM Press, New York, 2013), pp. 53–62.

30. Y. Zhang, K. Adl, and J. R. Glass, “Fast Spoken Query Detection Using Lower-Bound Dynamic Time Warping on Graphical Processing Units,” in *Proc. 2012 IEEE Int. Conf. on Acoustics, Speech and Signal Proc., Kyoto, Japan, March 25–30, 2012* (IEEE Press, New York, 2012), pp. 5173–5176.

31. M. Zymbler, “Best-Match Time Series Subsequence Search on the Intel Many Integrated Core Architecture,” in *Lecture Notes in Computer Science* (Springer, Heidelberg, 2015), Vol. 9282, pp. 275–286.