Поиск аномалий в больших временных рядах на кластере с GPU узлами

Я.А. Краева, М.Л. Цымблер

Южно-Уральский государственный университет

В настоящее время обнаружение аномалий во временных рядах является одной из наиболее актуальных исследовательских проблем и возникает в широком спектре предметных областей: цифровая индустрия, здравоохранение, моделирование климата и прогноз погоды, финансовая аналитика и др. Одной из наиболее часто используемых на практике формализаций понятия аномалии является концепция диссонанса. Диссонанс определяется как подпоследовательность ряда, которая имеет расстояние до своего ближайшего соседа, не превышающее наперед заданного аналитиком порога. Ближайшим соседом подпоследовательности является та подпоследовательность ряда, которая не пересекается с данной и имеет минимальное расстояние до нее. В предыдущих исследованиях авторы разработали параллельный алгоритм поиска диссонансов, имеющих длину в заданном диапазоне длин, на графическом процессоре. Однако данный алгоритм ограничивает длину ряда размером оперативной памяти GPU. В статье представлено продолжение указанного исследования, в котором предлагается алгоритм поиска диссонансов временного ряда на вычислительном кластере, каждый узел которого оснащен графическим процессором. Вычислительные эксперименты с временными рядами из реальных предметных областей показывают высокую масштабируемость разработанного алгоритма.

Ключевые слова: временной ряд, поиск аномалий, диссонанс, параллельный алгоритм, вычислительный кластер, графический процессор, CUDA, DRAG, MERLIN, PD3, PALMAD.

Введение

В настоящее время обнаружение аномалий во временных рядах является одной из наиболее актуальных исследовательских проблем и возникает в широком спектре предметных областей: цифровая индустрия, здравоохранение, моделирование климата и прогноз погоды, финансовая аналитика и др. [1]. Наиболее востребованной и сложной здесь является задача поиска аномальных подпоследовательностей, состоящих из последовательных во времени значений, коллективное поведение которых необычно, хотя каждое из них в отдельности не обязательно является выбросом (аномалией) [1].

Концепция диссонанса временного ряда является в настоящее время одним из наиболее перспективных подходов к формализации понятия аномальной подпоследовательности [2, 3]. Диссонанс [4] определяется как подпоследовательность ряда, которая имеет максимальное расстояние до своего ближайшего соседа. Ближайшим соседом подпоследовательности является та подпоследовательность ряда, которая не пересекается с данной и имеет минимальное расстояние до нее.

Недавно разработанный алгоритм MERLIN [5] обеспечивает эффективный поиск диссонансов временного ряда, имеющих длину в заданном диапазоне. Авторами настоящей статьи в предыдущих исследованиях разработан алгоритм PALMAD [6], распараллеливающий MERLIN на платформе графического процессора. Однако MERLIN и PALMAD ограничивают длину временного ряда размером оперативной памяти GPU. В настоящей статье авторы продолжают свои исследования и представляют алгоритм поиска диссонансов временного ряда, имеющих длину в заданном диапазоне, на вычислительном кластере, каждый узел которого оснащен графическим процессором.

Статья имеет следующую структуру. В разделе 1 приводится обзор работ по темати-

ке исследования. Раздел 2 содержит нотацию и формальные определения, а также краткое описание последовательных и параллельных алгоритмов поиска диссонансов, задействованных в данном исследовании. В разделе 3 представлен новый алгоритм поиска диссонансов на вычислительном кластере с GPU-узлами. Результаты вычислительных экспериментов по исследованию эффективности предложенного алгоритма описаны в разделе 4. Заключение подводит итоги исследования и описывает направление будущих работ.

1. Обзор связанных работ

Концепция диссонансов была предложена в работе [4] одновременно с алгоритмом HOTSAX (Heuristically Ordered Time series using Symbolic Aggregate ApproXimation) для их поиска. Диссонанс определяется как подпоследовательность ряда, которая имеет максимальное расстояние до своего ближайшего соседа. Ближайшим соседом подпоследовательности является та подпоследовательность ряда, которая не пересекается с данной и имеет минимальное расстояние до нее. Однако HOTSAX является приближенным алгоритмом, поскольку основан на сжатии подпоследовательностей исходного временного ряда с помощью символьной агрегатной аппроксимации [7]. В работе [8] была предложена схема распараллеливания данного алгоритма для многоядерных процессоров Intel и графических процессоров NVIDIA. Общей особенностью указанных алгоритмов является ограничение длины временного ряда размером, допускающим размещение исходных данных в оперативной памяти.

В работе [9] предложена концепция диапазонных диссонансов и алгоритм DRAG (Discord Range Aware Gathering) для их обнаружения, предполагающий хранение временного ряда на диске, а не в оперативной памяти. Диапазонный диссонанс представляет собой подпоследовательность ряда, расстояние от которой до ее ближайшего соседа не ниже r, где r – заданный порог. DRAG предполагает два сканирования ряда с помощью скользящего окна размером, равным длине искомых диссонансов. На первом проходе выполняется отбор подпоследовательностей-кандидатов в диссонансы, на втором – очистка полученного множества кандидатов от ложноположительных образцов. Позднее в работе [10] те же авторы предложили схему распараллеливания DRAG на основе парадигмы MapReduce, однако в вычислительных экспериментах ограничились симуляцией выполнения алгоритма на распределенной памяти.

Авторами настоящей статьи в работе [11] предложен алгоритм PD3 (Parallel DRAG-based Discord Discovery), распараллеливающий DRAG на платформе графического процессора. В работе [12] авторы настоящей статьи предложили схему распараллеливания DRAG для кластера с многоядерными процессорами, которая состоит из следующих шагов. На первом шаге формируются множества локальных кандидатов в диссонансы путем выполнения на каждом отдельном узле кластера процедур отбора и очистки для соответствующих фрагментов ряда. Далее один из узлов кластера выступает в роли сборщика и формирует множество глобальных кандидатов, объединяя множества локальных кандидатов, полученные от остальных узлов кластера. Затем узел-сборщик рассылает полученное им множество всем узлам кластера, каждый из которых выполняет процедуру очистки глобальных кандидатов для своего фрагмента, формируя тем самым множество локальных диссонансов. Наконец, узел-сборщик формирует итоговый ответ, выполняя пересечение множеств локальных диссонансов, получаемых от всех узлов кластера. В данной схеме применены технологии OpenMP для реализации процедур отбора и очистки на одном узле кластера и MPI для обменов данными между узлами кластера соответственно. В экспериментах данный алгоритм показал существенно бóльшую производительность, чем алгоритмы DDD (Distributed Discord Discovery) [13] и PDD (Parallel Discord Discovery) [14], которые применяют иные схемы распараллеливания, предполагающие интенсивные обмены данными между узлами кластера.

Тем не менее, указанные выше алгоритмы обеспечивают обнаружение диссонансов лишь

только одной (заданной исследователем) длины. Прямолинейным обобщением данного решения на случай диапазона длин диссонансов был бы циклический запуск алгоритма DRAG для каждой из длин в заданном диапазоне. Однако на практике это решение, как правило, является вычислительно невозможным, поскольку требует заново подбирать параметр r на каждом шаге такого цикла. Предложенный авторами алгоритма DRAG новый последовательный алгоритм MERLIN [5] снимает указанное выше ограничение. MERLIN многократно вызывает DRAG и адаптивно подбирает параметр r. В экспериментах MERLIN обнаруживает диссонансы любой длины из заданного диапазона длин, опережая конкурентов как по точности, так и по производительности [5]. Авторами настоящей статьи предложен алгоритм PALMAD (Parallel Arbitrary Length MERLIN-based Anomaly Detection) [6], в котором вычислительная схема MERLIN распараллеливается для графического процессора. PALMAD многократно вызывает PD3 и сокращает объем вычислений за счет применения выведенных авторами рекуррентных формул расчета среднего и стандартного отклонения соседних подпоследовательностей при нахождении расстояний.

Подводя итоги обзора, можно заключить, что в настоящее время MERLIN является одним из наиболее эффективных последовательных алгоритмов поиска аномальных подпоследовательностей временного ряда. Однако вместе с тем пока не предложен подход к распараллеливанию MERLIN на платформе вычислительного кластера с узлами на базе графических процессоров, который мог бы существенно повысить эффективность поиска аномалий больших временных рядов.

2. Предварительный теоретический базис

2.1. Формальные определения и обозначения

Временной ряд (time series) T представляет собой последовательность хронологически упорядоченных вещественных значений:

$$T = (t_1, \ldots, t_n), \quad t_i \in \mathbb{R}. \tag{1}$$

Число n обозначается $|T|$ и называется длиной ряда.

Подпоследовательность (subsequence) $T_{i,\,m}$ временного ряда T представляет собой непрерывный промежуток из m элементов, начиная с позиции i:

$$T_{i,m} = (t_i, \ldots, t_{i+m-1}), \quad 1 \leqslant m \ll n, \quad 1 \leqslant i \leqslant n - m + 1. \tag{2}$$

Множество всех подпоследовательностей ряда T, имеющих длину m, обозначим как S_T^m, а мощность такого множества за N, $N = |S_T^m| = n - m + 1$.

Подпоследовательности $T_{i,\,m}$ и $T_{j,\,m}$ ряда T называются *непересекающимися (non-self match)*, если $|i - j| \geqslant m$. Подпоследовательность, которая является непересекающейся к данной подпоследовательности C, обозначим как M_C.

Подпоследовательность D ряда T является *диапазонным диссонансом (range discord)*, если

$$\min_{M_D \in T} \big(\mathrm{Dist}(D, M_D)\big) \geqslant r, \tag{3}$$

где $\mathrm{Dist}(\cdot\,,\cdot\,)$ представляет собой неотрицательную симметричную функцию, порог расстояния r – наперед заданный параметр. Другими словами, некая подпоследовательность ряда является диапазонным диссонансом, если ее ближайший сосед (ближайшая и не пересекающаяся с ней подпоследовательность) находится на расстоянии не менее чем r. Далее для краткости мы будем использовать термин “диссонанс”, подразумевая диапазонный диссонанс, если не указано обратное.

Рассматриваемые далее последовательные и параллельные алгоритмы поиска диссонансов предполагают, что обрабатываемые подпоследовательности временного ряда предварительно подвергнуты z-нормализации. *Z-нормализация* подпоследовательности (ряда) T

представляет собой подпоследовательность (ряд) $\hat{T} = (\hat{t}_1, \ldots, \hat{t}_m)$, элементы которого вычисляются следующим образом:

$$\hat{t}_i = \frac{t_i - \mu}{\sigma}, \quad \mu = \frac{1}{m}\sum_{i=1}^{m} t_i, \quad \sigma^2 = \frac{1}{m}\sum_{i=1}^{m} t_i^2 - \mu^2. \tag{4}$$

Для обеспечения лучшей производительности в качестве функции расстояния $\mathrm{Dist}(\cdot,\cdot)$ в нашем исследовании применяется квадрат z-нормализованного евклидова расстояния, эффективное вычисление которого производится по следующей формуле [15]:

$$\mathrm{Dist}(X,\, Y) = 2m\left(1 - \frac{X\cdot Y - m\cdot \mu_X \cdot \mu_Y}{m\cdot \sigma_X \cdot \sigma_Y}\right), \tag{5}$$

где $X\cdot Y$ обозначает скалярное произведение векторов $X, Y \in \mathbb{R}^m$.

2.2. Последовательные алгоритмы DRAG и MERLIN

Алгоритм DRAG [9] предполагает две фазы: отбор и очистка кандидатов, где выполняется соответственно поиск потенциальных диссонансов и отбрасывание ложноположительных экземпляров. На первой фазе множество кандидатов в диссонансы $\mathcal{C}$ инициализируется первой подпоследовательностью ряда, имеющей заданную длину m. Далее DRAG просматривает временной ряд T и для каждой подпоследовательности $s \in S_T^m$ проверяет, что каждый кандидат $c \in \mathcal{C}$ является диссонансом в соответствии с определением (3). Если кандидат c не проходит проверку, то он удаляется из $\mathcal{C}$. В конце концов, s либо добавляется в множество кандидатов как потенциальный диссонанс, либо удаляется из него. На второй фазе алгоритм инициализирует расстояния всех кандидатов до их ближайших соседей значением $+\infty$ и сканирует ряд, вычисляя расстояние между каждой подпоследовательностью $s \in S_T^m$ и каждым кандидатом c. Если расстояние меньше r, то кандидат является ложноположительным и навсегда удаляется из $\mathcal{C}$. Если вышеупомянутое расстояние меньше текущего лучшего (минимального) расстояния до ближайшего соседа, то обновляется текущее лучшее расстояние до ближайшего соседа.

Алгоритм MERLIN [5] предписывает следующую процедуру подбора параметра r. Поиск диссонансов выполняется последовательно для каждого значения длины диссонанса в диапазоне $minL..maxL$. На каждом шаге MERLIN вычисляет среднее арифметическое μ и стандартное отклонение σ расстояний от последних пяти найденных диссонансов до их ближайших соседей, а затем вызывает алгоритм DRAG, передавая ему параметр $r = \mu - 2\sigma$. Если DRAG не обнаружил диссонанс, то σ вычитается из r до тех пор, пока DRAG не завершится успешно (т.е. пока не будет найден диссонанс). Для первых пяти длин диссонансов параметр r устанавливается следующим образом. Для диссонансов длины $minL$ берется значение $r = 2\sqrt{minL}$, поскольку это максимально возможное расстояние между любой парой подпоследовательностей длины $minL$, и тогда r уменьшается вдвое до тех пор, пока DRAG с таким параметром не завершится успешно. Чтобы обработать следующие четыре длины диссонанса, DRAG принимает расстояние от диссонанса до его ближайшего соседа, полученное на предыдущем шаге, за вычетом небольшого значения, равного 1%. Вычитание дополнительного 1% продолжается до тех пор, пока обнаружение диссонанса с таким параметром не приведет к успеху.

2.3. Параллельные алгоритмы PD3 и PALMAD

Алгоритм PD3 [11] распараллеливает каждую из фаз отбора и очистки алгоритма DRAG на платформе GPU на основе концепции параллелизма по данным. Временной ряд разбивается на *сегменты равной длины*, каждый из которых обрабатывается отдельным блоком нитей GPU. *Параллельный отбор кандидатов* в диссонансы организуется следую-

щим образом. Блок нитей полагает подпоследовательности сегмента (локальными) кандидатами и выполняет обработку тех из них, которые не пересекаются с кандидатами и расположены справа от данного сегмента, следующим образом. Если расстояние от кандидата до подпоследовательности меньше порога r, то кандидат и подпоследовательность исключаются из дальнейшей обработки как заведомо не являющиеся диссонансами. Если все локальные кандидаты отброшены, блок досрочно завершает работу. Блок нитей обрабатывает подпоследовательности ряда порциями, количество элементов в которых равно длине сегмента. Используя формулу (5), блок вычисляет расстояния от всех подпоследовательностей своего сегмента до всех подпоследовательностей текущей порции следующим образом. Сначала нити блока вычисляют скалярные произведения между первой подпоследовательностью сегмента и всеми подпоследовательностями текущей порции и сохраняют результаты в массиве, хранящемся в разделяемой памяти GPU. Затем они вычисляют скалярные произведения между первой подпоследовательностью текущей порции и всеми подпоследовательностями сегмента, сохраняя результат в другом массиве, также хранящемся в разделяемой памяти. Далее на основе полученных результатов вычисляются расстояния между первой подпоследовательностью порции и всеми подпоследовательностями сегмента. С помощью вычисленного расстояния отбрасываются бесперспективные кандидаты в сегменте и текущей порции. Если отброшены все кандидаты сегмента, блок заканчивает работу. После этого нити выполняют аналогичные действия над оставшимися подпоследовательностями текущей порции. *Параллельная очистка кандидатов* сходна с описанной выше процедурой параллельного отбора. В очистке участвуют те сегменты ряда, множество локальных кандидатов которых не пусто. Очистка кандидатов сегмента заключается в сканировании и обработке подпоследовательностей ряда, которые не пересекаются с кандидатами и расположены слева от данного сегмента. Если расстояние от кандидата до подпоследовательности меньше порога r, то кандидат отбрасывается.

Алгоритм PALMAD [6] в целом повторяет вычислительную схему оригинального алгоритма MERLIN на графическом процессоре, многократно вызывая алгоритм PD3. Перед первым вызовом PD3 выполняется параллельное вычисление двух векторов, хранящих соответственно средние значения и стандартные отклонения всех подпоследовательностей ряда, имеющих длину, равную заданной минимальной длине диссонанса. Далее указанные векторы обеспечивают сокращение объема вычислений за счет применения следующих выведенных авторами рекуррентных формул расчета среднего и стандартного отклонения соседних подпоследовательностей при нахождении расстояний:

$$\mu_{T_{i,\,m+1}} = \frac{1}{m+1}\big(m\mu_{T_{i,\,m}} + t_{i+m}\big), \tag{6}$$

$$\sigma^2_{T_{i,\,m+1}} = \frac{m}{m+1}\Big(\sigma^2_{T_{i,\,m}} + \frac{1}{m+1}\big(\mu_{T_{i,\,m}} - t_{i+m}\big)^2\Big). \tag{7}$$

3. Поиск диссонансов на кластере с GPU-узлами

3.1. Общая схема вычислений

Рис. 1 отражает общую схему вычислений предлагаемого алгоритма. Новый алгоритм модифицирует схему вычислений, реализованную авторами ранее в алгоритме PALMAD [6], применяя концепцию параллелизма по данным. В модифицированной схеме ряд разбивается на фрагменты, распределяемые по узлам вычислительного кластера, и каждый узел выполняет многократные вызовы алгоритма PD3 [11] для обработки собственного фрагмента на графическом процессоре. При этом выполнении процедуры подбора параметра r требуются обмены данными между узлами: для каждого фиксированного значения r каждому узлу кластера необходимо проверить, что найденные в текущем фрагменте кандидаты в диссонансы являются таковыми для фрагментов всех остальных узлов.

Для описания предложенного алгоритма введем следующие обозначения. Пусть име-

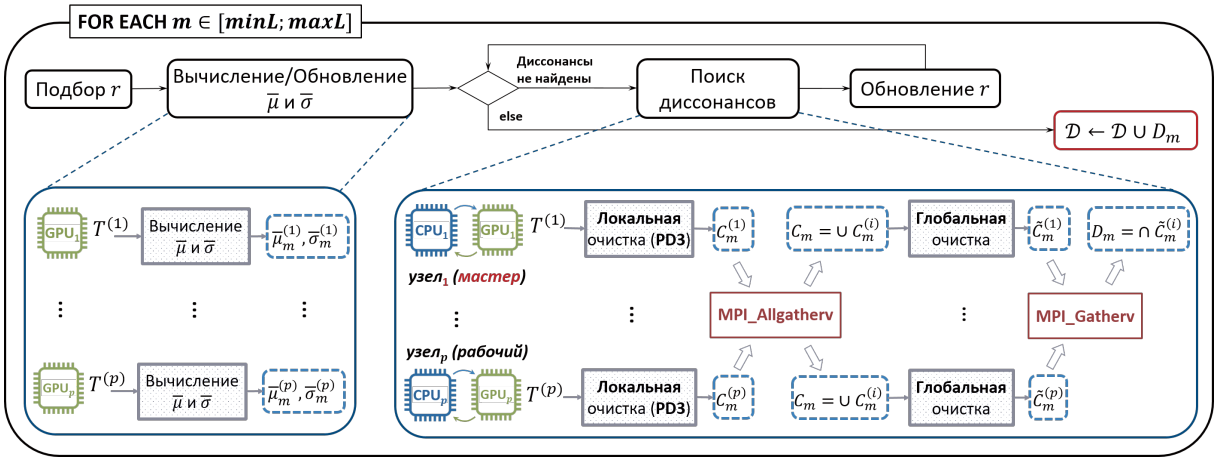


Рис. 1. Общая схема вычислений

ется временной ряд T, $|T| = n$, и требуется найти все его диссонансы, имеющие длину в диапазоне $minL..maxL$ ($3 \leqslant minL \leqslant maxL \ll n$). Пусть алгоритму выделено p узлов вычислительного кластера, каждый из которых оснащен центральным и графическим процессорами.

Временной ряд разбивается на фрагменты равной длины по числу доступных алгоритму узлов кластера, обозначаемые как $T^{(1)}, \ldots, T^{(p)}$. При этом в конец каждого фрагмента $T^{(i)}$ (за исключением последнего $T^{(p)}$) добавляется $maxL - 1$ элементов, взятых из начала следующего фрагмента $T^{(i+1)}$, для предотвращения потери результатов на стыке фрагментов.

Далее выполняется выравнивание каждого фрагмента для обеспечения баланса загрузки нитей графического процессора, обрабатывающих фрагмент. Длина фрагмента устанавливается кратной размеру варпа графического процессора. Если таковая кратность не имеет место, то фрагмент дополняется справа фиктивными элементами, имеющими значение $+\infty$. Мы предполагаем, что в итоге описанных преобразований фрагмент ряда может быть целиком размещен в оперативной памяти как центрального, так и графического процессоров.

Для поиска множества диссонансов $\mathcal{D}_m$, имеющих длину m ($minL \leqslant m \leqslant maxL$) каждый шаг подбора параметра r на вычислительном кластере с графическими процессорами выполняется по следующей схеме. Сперва каждый i-й узел выполняет *локальный отбор*: с помощью алгоритма PD3 находит во фрагменте $T^{(i)}$ множество $\mathcal{C}_m^{(i)}$ локальных кандидатов в диссонансы. Узлы кластера осуществляют обмен найденными локальными кандидатами по принципу "каждый с каждым", после чего на каждом узле формируется множество глобальных кандидатов в диссонансы $\mathcal{C}_m = \cup_{i=1}^{p} \mathcal{C}_m^{(i)}$. Далее каждый узел выполняет *глобальную очистку* множества $\mathcal{C}$ от ложноположительных диссонансов (см. ниже раздел 3.2) и формирует множество локальных диссонансов $\tilde{\mathcal{C}}_m^{(i)}$. После этого один из узлов вычислительного кластера (например, нулевой узел) объявляется мастером, остальные – рабочими. Каждый узел-рабочий отправляет узлу-мастеру свое множество локальных диссонансов, после чего мастер формирует результирующее множество диссонансов заданной длины как $\mathcal{D}_m = \cap_{i=1}^{p} \tilde{\mathcal{C}}_m^{(i)}$. Пороговое расстояние вычисляется как $r = \max\limits_{c \in \mathcal{D}_m} c.nnDist$, где запись $c.nnDist$ означает расстояние от кандидата в диссонансы c до его ближайшего соседа.

По завершении шага подбора параметра r узел-мастер пополняет итоговое множество диссонансов исходного ряда $\mathcal{D}$, добавляя в него только что найденное множество $\mathcal{D}_m$. Таким образом, итоговое множество диссонансов вычисляется как $\mathcal{D} = \cup_{m=minL}^{maxL} \mathcal{D}_m$.

В описанной выше схеме мы используем функции стандарта MPI `MPI_Allgatherv` и `MPI_Gatherv` для реализации приема-передачи локальных кандидатов и локальных диссонансов соответственно.

3.2. Распараллеливание глобальной очистки кандидатов

Глобальная очистка заключается в проведении каждым узлом кластера очистки множества кандидатов $\mathcal{C}_m$ для множества подпоследовательностей соответствующего фрагмента $S^m_{T^{(i)}}$ в соответствии с процедурой, предусмотренной в алгоритме DRAG (см. раздел 2.2). В соответствии с этим нам необходимо вычислить расстояния от каждого кандидата $c \in \mathcal{C}_m$ до каждой подпоследовательности $s \in S^m_{T^{(i)}}$, используя формулу (5).

В памяти графического процессора узла кластера множества подпоследовательностей-кандидатов $\mathcal{C}_m$ и подпоследовательностей фрагмента $S^m_{T^{(i)}}$ представляются в виде матриц $C \in \mathbb{R}^{|\mathcal{C}_m| \times m}$ и $S \in \mathbb{R}^{N \times m}$ соответственно. Для хранения средних арифметических и стандартных отклонений подпоследовательностей-кандидатов предусматриваются векторы $\overline{\mu}, \overline{\sigma} \in \mathbb{R}^{|\mathcal{C}_m|}$ соответственно. Для хранения вычисленных расстояний используется массив $nnDist \in \mathbb{R}^{|\mathcal{C}_m|}$.

Результатом умножения матриц S и C является матрица $DP \in \mathbb{R}^{N \times |\mathcal{C}_m|}$, каждый элемент которой представляет скалярное произведение соответствующих строк матрицы S (подпоследовательностей) и столбцов транспонированной матрицы C^{T} (кандидатов). Для выполнения указанного умножения на графическом процессоре формируется двумерная сетка нитей, состоящая из блоков нитей размера $block \times block$, где параметр $block$ (количество нитей в блоке) выбирается кратным размеру варпа и не превышает максимальное количество нитей в блоке. Каждый блок нитей вычисляет одну подматрицу (тайл) $Tile_{\mathrm{DP}} \in \mathbb{R}^{block \times block}$, а нить блока отвечает за вычисление одного элемента тайла.

Далее каждая нить блока, имеющая координаты (i, j), используя формулу (5), вычисляет расстояние между кандидатом $C(i, \cdot)$ и подпоследовательностью $S(j, \cdot)$ как

$$dist = 2m \left(1 - \frac{DP(i, j) - m \cdot \overline{\mu}(i) \cdot \overline{\mu}(j)}{m \cdot \overline{\sigma}(i) \cdot \overline{\sigma}(j)}\right).$$

Затем, используя операцию атомарного минимума (выполняемого над данными в приватной памяти текущей нити), блок нитей вычисляет расстояние от кандидата $C(i, \cdot)$ до его ближайшего соседа как минимум всех вышеуказанных расстояний, помещая результат в $nnDist(i)$.

На финальном шаге глобальной очистки, используя подсчитанные выше расстояния до ближайших соседей кандидатов, множество локальных диссонансов формируется как $\tilde{\mathcal{C}}^{(i)}_m = \{c \in \mathcal{C}^{(i)}_m \mid c.nnDist \geqslant r\}$.

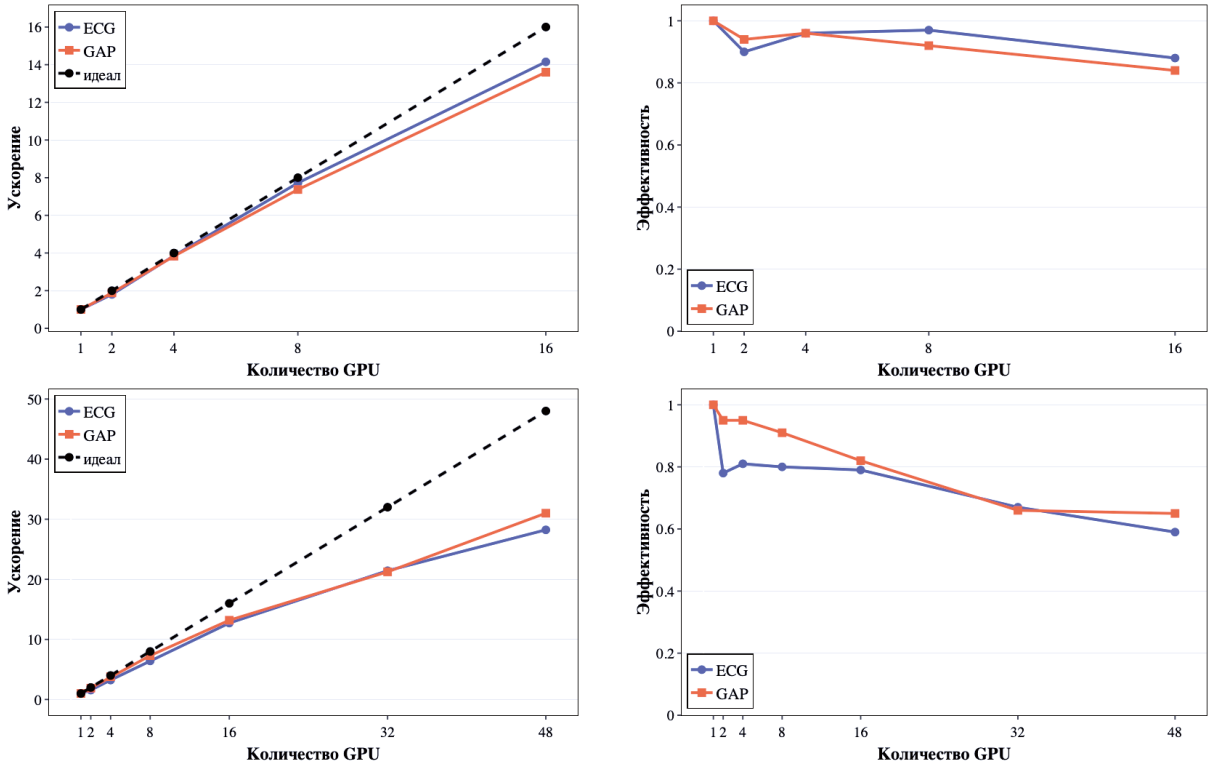
4. Вычислительные эксперименты

Для исследования эффективности разработанного алгоритма нами были проведены вычислительные эксперименты на платформе суперкомпьютера Ломоносов-2 [16] для следующих двух конфигураций вычислительного кластера с узлами на базе графического процессора. *Конфигурация 16×K40* задействовала 16 узлов раздела Compute суперкомпьютера, каждый из которых оснащен графическим процессором NVIDIA Tesla K40 (2 880 ядер @745 МГц, пиковая производительность 1.682 TFLOPS для арифметики двойной точности). *Конфигурация 48×P100* использовала 48 узлов раздела Pascal суперкомпьютера, на каждом из которых установлен графический процессор NVIDIA Tesla P100 (3 584 ядер @1 328 МГц, пиковая производительность 5.3 TFLOPS для арифметики двойной точности).

В экспериментах исследовались производительность, ускорение и эффективность распараллеливания алгоритма, определяемые следующим образом. *Производительность* представляет собой усредненное по 10 запускам время работы алгоритма, не включающее затраты на ввод исходных данных и вывод полученных результатов. *Ускорение* алгоритма $s(p)$ на p узлах кластера вычисляется как $s(p) = \frac{t_1}{t_p}$, где t_1 и t_p – производительность соответственно ранее разработанного алгоритма PALMAD на одном графическом процессоре узла кластера и алгоритма, разработанного в рамках данной статьи, на p узлах кластера.

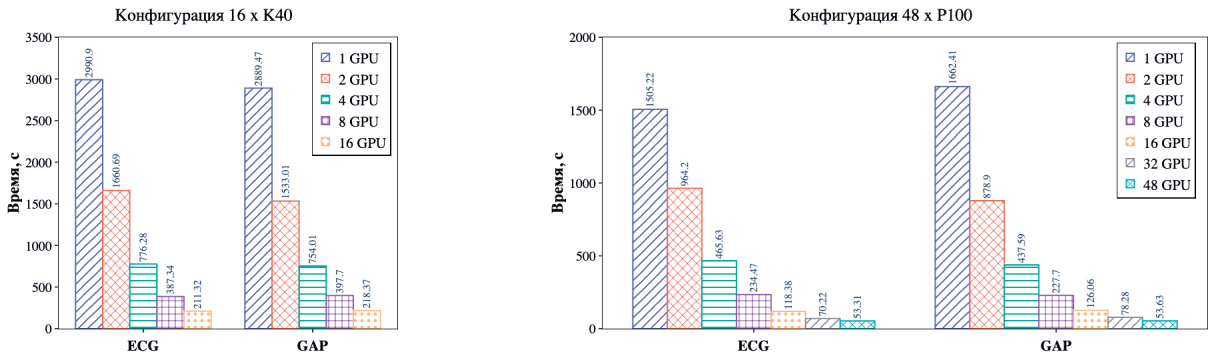
Эффективность распараллеливания алгоритма $e(p)$ на p узлах кластера определяется как $e(p) = \frac{s(p)}{p}$.

В экспериментах использовались следующие *наборы данных*. Ряд ECG [17] содержит показания электрокардиограммы взрослого пациента и имеет длину 2 000 000 точек. Ряд GAP [18] представляет собой поминутные показатели общего энергопотребления частного дома во Франции в период 2006–2010 гг. и содержит 2 000 000 точек. Для обоих рядов поиск диссонансов осуществлялся в диапазоне длин 64..128.



(a) Ускорение

(b) Эффективность

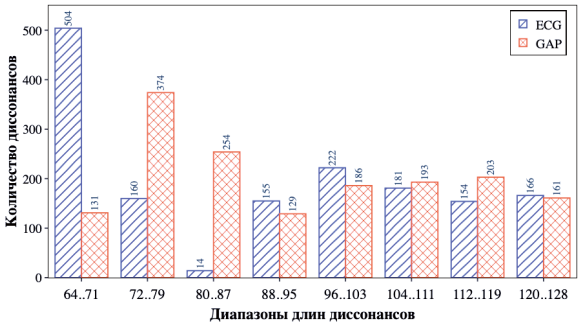


(c) Производительность

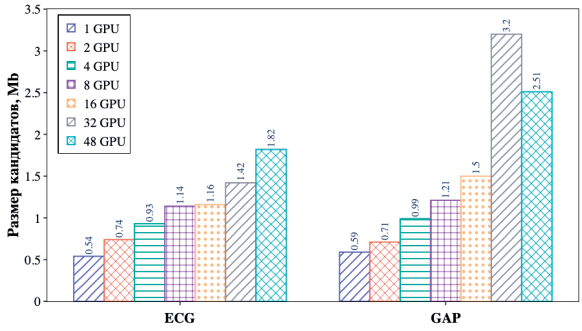
Рис. 2. Масштабируемость алгоритма на различных конфигурациях вычислительного кластера

Графики, представленные на рис. 2, отражают масштабируемость алгоритма в проведенных экспериментах. Можно видеть, что для конфигурации 16×K40 при поиске диссонансов в обоих рядах разработанный алгоритм демонстрирует близкие к линейным ускорение и эффективность (см. верхние графики рис. 2a и 2b соответственно). В случае конфигурации 48×P100 ускорение и эффективность для обоих рядов становятся сублинейными (см.

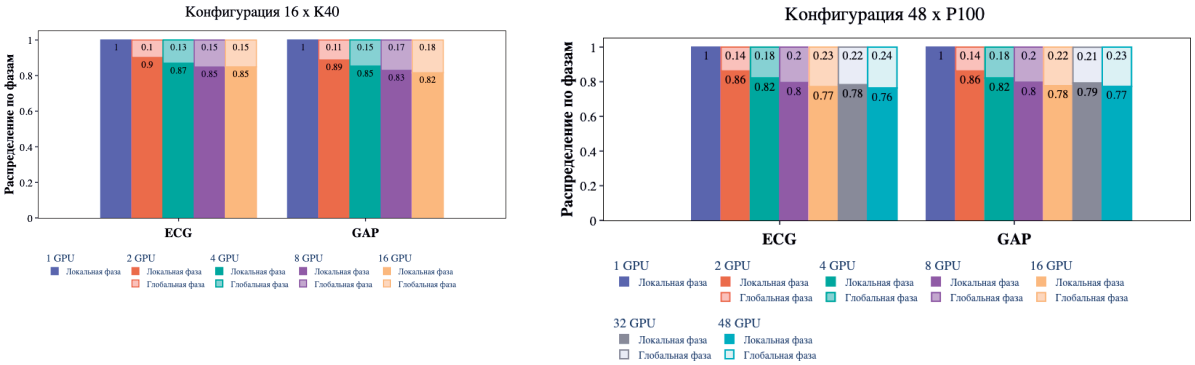
нижние графики рис. 2a и 2b соответственно). Производительность алгоритма в случае конфигурации 16×K40 ожидаемо ниже, чем для конфигурации 48×P100, которая задействует более мощные графические процессоры в бóльшем количестве (см. графики слева и справа рис. 2c соответственно).



(a) Количество найденных диссонансов



(b) Размер обработанных кандидатов



(c) Распределение времени по фазам алгоритма

Рис. 3. Статистические итоги работы алгоритма

Снижение масштабируемости алгоритма при переходе к конфигурации с бóльшим количеством узлов объясняется возрастающими при этом накладными расходами на обмены кандидатами между узлами. Тем не менее, масштабируемость алгоритма сохраняет линейный характер и ее деградация отсутствует. Отметим также, что для решения задачи поиска диссонансов на параллельных системах с распределенной памятью обмены кандидатами неизбежны, поскольку применение парадигмы MapReduce приведет к неадекватному результату: кандидаты в диссонансы, найденные в одном фрагменте ряда, но очищенные в рамках того же фрагмента, очевидно, не обязаны являться диссонансами ряда в смысле определения (3).

Статистические итоги поиска диссонансов (см. рис. 3) показывают следующее. В обоих рядах распределение найденных диссонансов по длинам примерно одинаковое (см. рис. 3a), а общее количество найденных диссонансов не превышает 0.08% от общего числа подпоследовательностей ряда с искомыми длинами, что согласуется с интуитивными представлениями об аномалиях.

Размер кандидатов, пересылаемых между узлами кластера (см. рис. 3b), определяет распределение времени работы алгоритма по его фазам. Локальная фаза подразумевает вычисления на графических процессорах узлов, необходимые для отбора и локальной очистки кандидатов. Глобальная фаза включает в себя пересылки кандидатов между узлами и вычисления на узле-мастере, необходимые для получения результирующего множества диссонансов. Бóльшее время выполнение глобальной фазы соответствует меньшей масштабируемости (см. рис. 3c и рис. 2a).

Заключение

В статье рассмотрена проблема обнаружения аномалий во временных рядах, которая в настоящее время является актуальной в широком спектре предметных областей: цифровая индустрия, здравоохранение, моделирование климата и прогноз погоды, финансовая аналитика и др. Для формализации понятия аномалии применяется концепция диссонанса [9] временного ряда. Подпоследовательность ряда является диссонансом, если ее ближайший сосед находится на расстоянии не менее чем заданный аналитиком порог r; ближайшим соседом называют подпоследовательность, которая не пересекается с данной и находится на минимальном расстоянии от нее. Алгоритм DRAG [9] реализует последовательный поиск диссонансов фиксированной длины. Алгоритм MERLIN [5] обеспечивает последовательный поиск диссонансов, имеющих длину в заданном диапазоне длин, и применяет многократные вызовы DRAG. В предыдущих исследованиях авторы настоящей статьи разработали параллельные версии вышеуказанных алгоритмов для графического процессора, соответственно PD3 [11] и PALMAD [6], где PALMAD применяет многократные вызовы PD3. Однако PALMAD ограничивает длину ряда размером оперативной памяти GPU. В настоящей статье предложен новый алгоритм поиска диссонансов временного ряда, имеющих длину в заданном диапазоне длин, на вычислительном кластере, каждый узел которого оснащен графическим процессором.

Предложенный алгоритм модифицирует схему вычислений PALMAD следующим образом. Временной ряд разбивается на фрагменты, распределяемые по узлам вычислительного кластера, и каждый узел выполняет многократные вызовы алгоритма PD3 для обработки собственного фрагмента на графическом процессоре. Алгоритм предусматривает цикл по диапазону длин диссонансов, где на каждом шаге подбор параметра r выполняется по следующей схеме. Сперва каждый узел выполняет локальный отбор: с помощью алгоритма PD3 находит во фрагменте множество локальных кандидатов в диссонансы. Далее с помощью технологии MPI узлы кластера осуществляют обмен полученными результатами по принципу “каждый с каждым”, и на каждом узле формируется множество глобальных кандидатов как объединение всех локальных кандидатов. Затем каждый узел выполняет глобальную очистку: из множества глобальных кандидатов удаляются ложноположительные диссонансы, формируя тем самым множество локальных диссонансов. Процедура глобальной очистки распараллеливается на основе блочного умножения матрицы подпоследовательностей-кандидатов и матрицы подпоследовательностей фрагмента. После этого один из узлов кластера объявляется мастером, остальные – рабочими. Каждый рабочий отправляет мастеру свое множество локальных диссонансов, после чего мастер формирует результирующее множество диссонансов рассматриваемой длины как пересечение множеств, полученных от рабочих и определяет порог как максимальное расстояние до ближайшего соседа среди только что найденных диссонансов.

Вычислительные эксперименты, проведенные на платформе суперкомпьютера Ломоносов-2 с временными рядами из реальных предметных областей, показывают близкие к линейным ускорение и эффективность разработанного алгоритма.

В будущих исследованиях мы планируем расширить предложенный алгоритм на случай, когда узел вычислительного кластера оснащается несколькими графическими процессорами.

Благодарности

Работа выполнена при финансовой поддержке Российского научного фонда (грант № 23-21-00465).

В исследованиях использовано оборудование Центра коллективного пользования сверхвысокопроизводительными вычислительными ресурсами МГУ имени М.В. Ломоносова.

Литература

1. Blázquez-Garcıa A., Conde A., Mori U., Lozano J.A. A review on outlier/anomaly detection in time series data // ACM Comput. Surv. 2021. Vol. 54, no. 3. 56:1–56:33. DOI: 10.1145/3444690.
2. Chandola V., Banerjee A., Kumar V. Anomaly detection: A survey // ACM Comput. Surv. 2009. Vol. 41, no. 3. 15:1–15:58. DOI: 10.1145/1541880.1541882.
3. Chandola V., Cheboli D., Kumar V. Detecting anomalies in a time series database. Retrieved from the University of Minnesota Digital Conservancy. 2009. Accessed: 2022-04-12. https://hdl.handle.net/11299/215791.
4. Lin J., Keogh E.J., Fu A.W., Herle H.V. Approximations to magic: Finding unusual medical time series // 18th IEEE Symposium on Computer-Based Medical Systems (CBMS 2005), 23-24 June 2005, Dublin, Ireland. IEEE Computer Society, 2005. P. 329–334. DOI: 10.1109/CBMS.2005.34.
5. Nakamura T., Imamura M., Mercer R., Keogh E.J. MERLIN: parameter-free discovery of arbitrary length anomalies in massive time series archives // 20th IEEE International Conference on Data Mining, ICDM 2020, Sorrento, Italy, November 17-20, 2020 / ed. by C. Plant, H. Wang, A. Cuzzocrea, *et al.* IEEE, 2020. P. 1190–1195. DOI: 10.1109/ICDM50108.2020.00147.
6. Zymbler M., Kraeva Y. High-performance Time Series Anomaly Discovery on Graphics Processors // CoRR. 2023. Vol. abs/2304.01660. arXiv: 2304.01660. URL: https://arxiv.org/abs/2304.01660.
7. Lin J., Keogh E.J., Lonardi S., Chiu B.Y. A symbolic representation of time series, with implications for streaming algorithms // Proceedings of the 8th ACM SIGMOD workshop on Research issues in data mining and knowledge discovery, DMKD 2003, San Diego, California, USA, June 13, 2003 / ed. by M.J. Zaki, C.C. Aggarwal. ACM, 2003. P. 2–11. DOI: 10.1145/882082.882086.
8. Zymbler M. A parallel discord discovery algorithm for time series on many-core accelerators // Numerical Methods and Programming. 2019. Vol. 20, no. 3. P. 211–223. (in Russian) DOI: 10.26089/NumMet.v20r320.
9. Yankov D., Keogh E.J., Rebbapragada U. Disk aware discord discovery: Finding unusual time series in terabyte sized datasets // Proceedings of the 7th IEEE International Conference on Data Mining (ICDM 2007), October 28-31, 2007, Omaha, Nebraska, USA. IEEE Computer Society, 2007. P. 381–390. DOI: 10.1109/ICDM.2007.61.
10. Yankov D., Keogh E.J., Rebbapragada U. Disk aware discord discovery: Finding unusual time series in terabyte sized datasets // Knowl. Inf. Syst. 2008. Vol. 17, no. 2. P. 241–262. DOI: 10.1007/s10115-008-0131-9.
11. Kraeva Y., Zymbler M. A parallel discord discovery algorithm for a graphics processor // Pattern Recognition and Image Analysis. 2023. Vol. 33, no. 2. P. 101–113. DOI: 10.1134/S1054661823020062.
12. Zymbler M., Grents A., Kraeva Y., Kumar S. A parallel approach to discords discovery in massive time series data // Computers, Materials & Continua. 2021. Vol. 66, no. 2. P. 1867–1878. DOI: 10.32604/cmc.2020.014232.

13. Wu Y., Zhu Y., Huang T., *et al.* Distributed discord discovery: Spark based anomaly detection in time series // 17th IEEE International Conference on High Performance Computing and Communications, HPCC 2015, 7th IEEE International Symposium on Cyberspace Safety and Security, CSS 2015, and 12th IEEE International Conference on Embedded Software and Systems, ICESS 2015, New York, NY, USA, August 24-26, 2015. IEEE, 2015. P. 154–159. DOI: 10.1109/HPCC-CSS-ICESS.2015.228.

14. Huang T., Zhu Y., Mao Y., *et al.* Parallel discord discovery // Advances in Knowledge Discovery and Data Mining - 20th Pacific-Asia Conference, PAKDD 2016, Auckland, New Zealand, April 19-22, 2016, Proceedings, Part II. Vol. 9652 / ed. by J. Bailey, L. Khan, T. Washio, *et al.* Springer, 2016. P. 233–244. Lecture Notes in Computer Science. DOI: 10.1007/978-3-319-31750-2_19.

15. Mueen A., Nath S., Liu J. Fast approximate correlation for massive time-series data // Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2010, Indianapolis, Indiana, USA, June 6-10, 2010 / ed. by A.K. Elmagarmid, D. Agrawal. ACM, 2010. P. 171–182. DOI: 10.1145/1807167.1807188.

16. Voevodin V.V., Antonov A.S., Nikitenko D.A., *et al.* Supercomputer Lomonosov-2: large scale, deep monitoring and fine analytics for the user community // Supercomput. Front. Innov. 2019. Vol. 6, no. 2. P. 4–11. DOI: 10.14529/jsfi190201.

17. Goldberger A.L., Amaral L.A.N., Glass L., *et al.* PhysioBank, PhysioToolkit, and PhysioNet components of a new research resource for complex physiologic signals // Circulation. 2000. Vol. 101, no. 23. P. 215–220. DOI: 10.1161/01.CIR.101.23.e215.

18. Individual household electric power consumption. 2023. Accessed: 2023-04-18. https://archive.ics.uci.edu/ml/datasets/individual+household+electric+power+consumption/.