УДК 004.272.25, 004.421, 004.032.24 DOI: 10.14529/cmse180305

ПАРАЛЛЕЛЬНЫЙ АЛГОРИТМ ВЫЧИСЛЕНИЯ МАТРИЦЫ ЕВКЛИДОВЫХ РАССТОЯНИЙ ДЛЯ МНОГОЯДЕРНОГО ПРОЦЕССОРА INTEL XEON PHI ПОКОЛЕНИЯ KNIGHTS LANDING

© 2018 Т.В. Речкалов, М.Л. Цымблер

Южно-Уральский государственный университет
(454080 Челябинск, пр. им. В.И. Ленина, д. 76)
E-mail: trechkalov@yandex.ru, mzym@susu.ru

Поступила в редакцию: 06.05.2018

Вычисление матрицы Евклидовых расстояний требуется в широком спектре задач, связанных с интеллектуальным анализом данных. В настоящее время большое количество параллельных алгоритмов решения этой задачи реализовано для графических процессоров. Однако данные разработки не могут быть просто перенесены на многоядерные системы архитектуры Intel Many Integrated Core. В статье предлагается параллельный алгоритм вычисления матрицы Евклидовых расстояний на многоядерном процессоре Intel Xeon Phi поколения Knights Landing для случая, когда входные данные могут быть размещены в оперативной памяти. Данный алгоритм использует блочно-ориентированную схему организации вычислений, которая позволяет эффективно использовать возможности векторизации вычислений Intel Xeon Phi. В алгоритме применена нетривиальная компоновка данных в оперативной памяти для уменьшения количества кэш-промахов процессора во время вычислений. Эксперименты на реальных и синтетических наборах данных показали, что предложенный алгоритм хорошо масштабируется и опережает аналоги в случае прямоугольных матриц с данными малой размерности.

Ключевые слова: матрица Евклидовых расстояний, OpenMP, Intel Xeon Phi, Knights Landing, компоновка данных в памяти, векторизация вычислений.

ОБРАЗЕЦ ЦИТИРОВАНИЯ

Речкалов Т.В., Цымблер М.Л. Параллельный алгоритм вычисления матрицы Евклидовых расстояний для многоядерного процессора Intel Xeon Phi Knights Landing // Вестник ЮУрГУ. Серия: Вычислительная математика и информатика. 2018. Т. 7, № 3. С. 65–82. DOI: 10.14529/cmse180305.

Введение

Вычисление матрицы Евклидовых расстояний (МЕР) требуется в широком спектре научных и практических задач, связанных с интеллектуальным анализом данных [5]. Элементами МЕР являются квадраты расстояний[1], которые могут быть интерпретированы как расстояния между точками одного множества или расстояния между точками двух множеств в Евклидовом пространстве. Указанные случаи порождают, соответственно, квадратные и прямоугольные матрицы. Квадратные МЕР используются в задачах извлечения аудио и видео информации [5], в обработке сигналов [5] и др. Прямоугольные МЕР играют важную роль в приложениях, связанных с кластеризацией данных, где необходимо вычислять расстояние между центрами кластеров и точек данных, подвергаемых кластеризации, например, в сегментировании медицинских изображений [12, 21], иерархической [2] нечеткой [4] кластеризации данных ДНК-микрочипов и др.

[1]Строго говоря, МЕР должны содержать Евклидовы расстояния, но не их квадраты. Однако, здесь и далее будет использоваться определение с квадратом расстояния, поскольку большинство авторов работ о МЕР придерживается данного соглашения [5].

В данной статье рассматривается задача вычисления как квадратной, так и прямоугольной МЕР, определяемая следующим образом. Рассмотрим два непустых конечных множества из n и m точек в d-мерном Евклидовом пространстве. Точки первого множества рассматриваются как строки матрицы $\mathbf{A} \in \mathbb{R}^{n \times d}$, точки второго множества — как строки матрицы $\mathbf{B} \in \mathbb{R}^{m \times d}$. Обозначим как $a_{1,\cdot}, \ \ldots, \ a_{n,\cdot}$ и $b_{1,\cdot}, \ \ldots, \ b_{m,\cdot}$, где $a_{i,\cdot}, b_{j,\cdot} \in \mathbb{R}^d$, строки матриц $\mathbf{A}$ и $\mathbf{B}$, соответственно. Тогда матрица Евклидовых расстояний $\mathbf{D} \in \mathbb{R}^{n \times m}$ состоит из строк $d_{1,\cdot}, \ \ldots, \ d_{n,\cdot}$, где $d_{i,\cdot} \in \mathbb{R}^m$, $d_{i,j} = \|a_{i,\cdot} - b_{j,\cdot}\|^2$, и $\|\cdot\|$ означает Евклидову норму[2].

Поскольку вычисление МЕР имеет кубическую временную сложность $O(nmd)$, данная задача зачастую представляет собой наиболее времяемкий этап решаемой аналитической проблемы и поэтому является предметом разработки параллельных алгоритмов для различных аппаратных архитектур.

В настоящее время большое количество параллельных алгоритмов вычисления МЕР разработано для графических процессоров [1, 2, 10, 13]. Эти разработки, однако, не могут быть напрямую перенесены на многоядерные системы Intel Xeon Phi [3, 18]. Intel Xeon Phi представляет собой семейство систем на основе архитектуры Intel Many Integrated Core (MIC), которая обеспечивает большое количество вычислительных ядер с высокой пропускной способностью локальной памяти и 512-битными векторными регистрами. Будучи основанными на архитектуре Intel x86, системы Intel Xeon Phi поддерживают параллелизм на уровне нитей и те же программные инструменты, что и обычный процессор Intel Xeon, что делает их конкурентоспособной альтернативой графическим процессорам. На сегодня компания Intel предлагает два поколения продуктов архитектуры MIC: Knights Corner (KNC) [3] и Knights Landing (KNL) [18]. KNC представляет собой сопроцессор с количеством ядер до 61, поддерживающий как нативные приложения, так и выгрузку (*offloading*) вычислений и данных с управляющего процессора. KNL обеспечивает до 72 ядер и, в отличие от предшественника, является самостоятельно загружаемым устройством, которое исполняет приложения только в нативном режиме.

В данной статье рассматривается задача параллельного вычисления МЕР на многоядерном процессоре Intel Xeon Phi поколения Knights Landing для случая, когда данные, задействованные в вычислениях, могут быть размещены в оперативной памяти. Описаны следующие основные результаты. Предложен параллельный алгоритм, основанный на новой блочно-ориентированной схеме вычислений, которая позволяет более эффективно использовать возможности процессора Intel Xeon Phi KNL по векторизации вычислений, чем прямолинейные техники на основе выравнивания данных и автоматической векторизации. Проведены вычислительные эксперименты на реальных и синтетических наборах данных, показавшие высокую масштабируемость разработанного алгоритма и его более высокую производительность, чем у аналогов, для случая прямоугольных матриц с данными малой размерности.

Остаток статьи организован следующим образом. В разделе 1 приведен обзор работ по тематике исследования. Раздел 2 содержит описание предложенного параллельного алгоритма вычисления МЕР. Результаты экспериментального исследования разработанного алгоритма приведены в разделе 3. В заключении суммируются полученные результаты и предлагаются направления будущих исследований.

[2]Заметим, что данное определение покрывает случай $\mathbf{A} \equiv \mathbf{B}$.

1. Обзор работ

В работе [2] Чанг (Chang) и др. предложили параллельный алгоритм вычисления MEP на GPU с помощью CUDA. Этот алгоритм предполагает, что матрица квадратная ($n = m$) и что размер матрицы n и размерность пространства d кратны 16. Алгоритмическое требование кратности 16 следует из архитектуры NVIDIA GPU. В соответствии с технологией CUDA нити группируются в двумерные блоки 16×16, которые, в свою очередь, формируют двумерную сетку $\frac{n}{16} \times \frac{n}{16}$. Таким образом, каждая нить идентифицируется четверкой (b_x, b_y, t_x, t_y), где пара (b_x, b_y) задает положение блока, а пара (t_x, t_y) — положение в сетке. В этой координатной системе нить вычисляет $d_{16 \cdot b_y + t_y, 16 \cdot b_x + t_x}$ элемент MEP. На каждой итерации все нити загружают две подматрицы размера 16×16 в разделяемую память. Каждая нить после синхронизации вычисляет частичное Евклидово расстояние. Далее нити должны быть снова синхронизированы, прежде чем загрузить следующие две подматрицы для расчета. Эксперименты авторов на системе NVIDIA Tesla C870 (с пиковой производительностью 0,5 GFLOPS) показали, что алгоритм обеспечивает ускорение до 44 раз по сравнению с процессором.

Ли (Li) и др. [13] предложили блочный метод вычисления MEP для сверхбольших наборов данных в распределенной среде с GPU. Данный метод предполагает реализацию алгоритма вычисления MEP на GPU для вычисления подматриц MEP. Затем авторы используют программную систему на основе парадигмы MapReduce для разбиения исходной задачи вычисления MEP на ряд маленьких независимых подзадач для вычисления подматриц MEP. Программная система динамически выделяет ресурсы GPU подзадачам для достижения максимальной производительности. В экспериментах на трех системах NVIDIA Tesla 1060 (0,9 GFLOPS каждая) алгоритм показывает ускорение до 15 раз.

Ким (Kim) и др. [10] предложили добавлять фиктивные нулевые данные при использовании алгоритма из работы [2]. Матрицы входных данных дополняются нулевыми строками и столбцами для обработки вычислительным ядром CUDA. В экспериментах на NVIDIA Tesla C2050 (0,5 GFLOPS) алгоритм показал ускорение до 47 раз по сравнению с процессором.

В работе [1] Арефин (Arefin) и др. улучшили подходы, предложенные в [2, 10, 13]. Вместе с MEP входные данные разбиваются на блоки. Данная операция выполняется во внешней памяти, из-за чего данное решение работает до 30 раз медленнее, чем в исходных работах. Тем не менее, данный подход применим при обработке данных, не помещающихся в оперативную память процессора или GPU.

В работах By (Wu) и др. [20], Ли (Lee) и др. [12] и Ярош (Jaros) и др. [9] косвенно затронута задача вычисления MEP на ускорителях Intel MIC. Авторы этих работ ускоряли на сопроцессорах Intel Xeon Phi KNC алгоритм кластеризации k-means, в котором вычисление MEP является подзадачей.

В работе [20] авторы предложили гетерогенный подход к распараллеливанию алгоритма k-Means, задействующий как процессор, так и сопроцессор Intel Xeon Phi KNC. В соответствии с данным подходом процессор выполняет назначение точек данных по кластерам и затем выгружает данные и центроиды на сопроцессор. Таким образом, Xeon Phi KNC на каждой итерации вычисляет MEP между исходными данными и центроидами. Для увеличения пропускной способности памяти и кэша алгоритм хранит данные в виде массива структур. В экспериментах авторами получено ускорение до 24 раз, однако

масштабируемость алгоритма существенно снижалась при использовании более чем 56 нитей.

Авторы работы [9] использовали похожий подход и выносили вычисления на Intel Xeon Phi KNC. Следует отметить, что в работах-предшественниках [9, 20] используются многоядерные системы архитектуры Intel MIC предыдущего поколения и выгрузку данных и вычислений, однако в описании экспериментального исследования алгоритмов стадия вычисления MEP не отделяется от других вычислений, затрудняя сравнение производительности и масштабируемости вычисления MEP с другими решениями.

В работе [12] авторы для вычисления MEP на процессоре Intel Xeon Phi KNL используют параллельный алгоритм, представленный в Алг. 1. В данном алгоритме применяются относительно простые техники ускорения вычислений — выравнивание данных и автовекторизация, — поэтому далее данная реализация будет упоминаться как STRAIGHTFORWARD.

Algorithm 1 STRAIGHTFORWARD(IN **A**, **B**; OUT **D**)

```
1:  #pragma omp parallel for
2:  for i from 1 to n do
3:      sum ← 0
4:      for j from 1 to m do
5:          __assume_aligned(a_{i,·}, 64)
6:          __assume_aligned(b_{j,·}, 64)
7:          for k from 1 to d do
8:              sum ← sum + (a_{i,k} − b_{j,k})^2
9:          end for
10:         d_{i,j} ← sum
11:     end for
12: end for
```

Строки 5–6 представляют собой подсказку компилятору Intel, что память, отведенная под каждую из матриц **A** и **B**, выровнена на указанное количество байтов. Начальные адреса исходных данных выравниваются таким образом, чтобы быть кратными ширине векторного регистра (VPU, Vector Processing Unit) системы Intel Xeon Phi[3]. Данный прием позволяет избежать эффекта разбиения цикла (*loop peeling*) в строке 7, существенно снижающего производительность векторизованного цикла. Действительно, в случае отсутствия подсказки компилятор предполагает, что данные не выровнены и разделит цикл на три части. Первая небольшая часть итераций цикла осуществляет доступ к исходным данным от начального адреса до первого выровненного адреса, векторизуется отдельно. Так же компилятором отдельно будет векторизована небольшая часть итераций цикла, которые обрабатывают исходные данные от последнего выровненного адреса до завершающего адреса данных.

Далее, в теле цикла в строке 7, несмотря на выравнивание начального адреса первой точки данных, начальный адрес второй точки данных не будет выровнен, если размерность d не будет кратна ширине векторного регистра, что снова приведет к

[3]Шириной векторного регистра называют максимальное количество элементов данных, которое можно поместить в регистр. Для Intel Xeon Phi размер векторного регистра составляет 512 бит, что позволяет разместить в нем 16 вещественных чисел с одинарной точностью.

разбиению цикла и неэффективной векторизации. Для решения этой проблемы авторы дополняют точки входных данных фиктивными нулевыми координатами таким образом, чтобы размерность точек была кратна ширине векторного регистра, в силу чего цикл разрешается компилятором как две векторные операции.

Тем не менее, в высокопроизводительных вычислениях, помимо выравнивания данных, способ компоновки данных оказывает существенное влияние на эффективность операций доступа к памяти [8]. В следующем разделе будет рассмотрено применение компоновки данных в памяти для ускорения вычисления MEP.

2. Ускорение вычислений на процессоре Intel Xeon Phi KNL

Предлагаемый в данной работе подход к вычислению MEP на процессоре Intel Xeon Phi KNL имеет два следующих основных отличия от алгоритма Straightforward. Во-первых, предлагается новая схема организации вычислений для эффективного использования возможностей векторизации Intel Xeon Phi. Во-вторых, используется более сложная компоновка данных в оперативной памяти. Указанные особенности рассматриваются ниже в разделах 2.1 и 2.2 соответственно.

2.1. Схема вычислений

Основная идея предлагаемого подхода заключается в изменении схемы вычислений таким образом, чтобы по сравнению с прямолинейным подходом было векторизовано большее количество операций. В алгоритме Straightforward в цикле вычисляется одно расстояние между двумя точками. Следовательно, внутренний цикл (см. Алг. 1, строка 7) будет скомпилирован в две векторные операции: поэлементные разность и произведение векторов.

Предлагаемый алгоритм, получивший название Blockwise, итеративно вычисляет несколько расстояний от одной точки из первого множества до *block* точек из второго множества, где *block* — параметр алгоритма (см. Алг. 2).

В строках 1–7 осуществляется изменение компоновки данных второго множества точек в памяти (детальное описание приведено в разделе 2.2) и формирование их копии для дальнейших вычислений. Внешний цикл (строка 9) перебирает первое множество точек и распараллеливается. Внутренний цикл в строке 10 перебирает блоки второго множества точек. Цикл в строке 12 выполняет вычисления по координатам блока точек. Цикл в строке 15 вычисляет расстояния и компилируется в две векторные операции. Строки 13 и 14 подсказывают компилятору о выравнивании в памяти точки из первого множества и блока точек из второго множества соответственно. Наконец, цикл в строке 20 сохраняет вычисленные расстояния в матрицу расстояний, причем эта операция компилируется в одну векторную операцию. Данному циклу предшествует подсказка для компилятора (строка 19) о выравнивании результирующей матрицы расстояний в памяти.

Описанный алгоритм предполагает, что мощность второго множества точек m кратна количеству блоков *block*. Если это не так, необходимо увеличить m до ближайшего целого числа, кратного *block*, путем дополнения матрицы $\mathbf{B}$ фиктивными нулевыми строками.

Кроме того, для гарантии эффективной векторизации операций над матрицей $\mathbf{B}$ параметр *block* должен быть кратен значению ширины векторного регистра $width_{VPU}$. Также, для получения большей выгоды от векторизации вычислений в качестве матрицы

Algorithm 2 BLOCKWISE(IN **A**, **B**, *layout*, *block*; OUT **D**)

1: **if** *layout* **is** SoA **then**
2: PERMUTE(B, m, $\tilde{B}$)
3: **else if** *layout* **is** ASA **then**
4: PERMUTE(B, *block*, $\tilde{B}$)
5: **else**
6: ▷ Компоновка AoS, перестановка элементов не нужна
7: **end if**
8: #pragma omp parallel for
9: **for** i **from** 1 **to** n **do**
10: **for** j **from** 1 **to** $\lceil \frac{m}{block} \rceil$ **do**
11: $sum \leftarrow \overline{0}$
12: **for** k **from** 1 **to** d **do**
13: __assume_aligned($a_{i,\cdot}$, 64)
14: __assume_aligned($\tilde{b}_{j+k,\cdot}$, 64)
15: **for** ℓ **from** 1 **to** *block* **do**
16: $sum_\ell \leftarrow sum_\ell + (a_{i,k} - \tilde{b}_{j+k,\ell})^2$
17: **end for**
18: **end for**
19: __assume_aligned($d_{i,\cdot}$, 64)
20: **for** k **from** 1 **to** *block* **do**
21: $d_{i,j \cdot block+k} \leftarrow sum_k$
22: **end for**
23: **end for**
24: **end for**

B необходимо взять матрицу, которая хранит наибольшее по мощности множество точек из двух заданных.

Отметим, что предложенный алгоритм предполагает эмпирический выбор параметра *block* в соответствии с вышеизложенными требованиями (детали выбора данного параметра рассмотрены далее в разделе 3).

2.2. Компоновка данных

На рис. 1 представлены основные способы компоновки массивов в памяти (в виде объявлений типов данных языка Си) [8].

```
typedef struct {
    float x;
    float y;
    float z;
} AoS;
```

AoS **B**[m];

a) Массив из структур

```
typedef struct {
    float x[m];
    float y[m];
    float z[m];
} SoA;
```

SoA **B**;

b) Структура из массивов

```
typedef struct {
    float x[block];
    float y[block];
    float z[block];
} ASA;
```

ASA **B**[$\lceil \frac{m}{block} \rceil$];

c) Массив структур из массивов

Рис. 1. Основные способы компоновки массивов в памяти

Компоновка AoS (Array of Structures) предполагает хранение множества точек в виде массива, элементами которого являются структуры, и часто используется как компоновка данных по умолчанию. В случае компоновки данных SoA (Structure of Arrays) для хранения каждой координаты точек используются отдельные массивы. Это может приводить к конфликтам совместного доступа нитей к памяти, если сценарий доступа к данным предполагает обработку смежных элементов. Компоновка ASA (Array of Structures of Arrays) разбивает данные на блоки в соответствии с параметром *block*. ASA-*block* обобщает компоновки данных, рассмотренные выше: ASA-1 соответствует AoS, а ASA-m соответствует SoA. Такая усложненная компоновка данных способствует уменьшению кэш-промахов процессора при вычислении матрицы расстояний.

Algorithm 3 PERMUTE(IN $\mathbf{B}$, *block*; OUT $\tilde{\mathbf{B}}$)

```
#pragma omp parallel for
for j from 1 to ⌈m/block⌉ do
    for i from 1 to d do
        for k from 1 to block do
            b̃_{j·d+i,k} ← b_{j·block+k,i}
        end for
    end for
end for
```

Алгоритм 3 параллельно преобразует матрицу исходных данных из одной компоновки данных в другую путем перестановки элементов. Для заданного размера *block* и матрицы $\mathbf{B} \in \mathbb{R}^{m \times d}$ с компоновкой данных AoS, алгоритм создает матрицу $\tilde{\mathbf{B}} \in \mathbb{R}^{d \cdot \lceil \frac{m}{block} \rceil \times block}$ с компоновкой данных ASA-*block* (или SoA, если $block = m$).

3. Экспериментальное исследование

3.1. Описание экспериментов

Цели. В экспериментах были исследованы производительность и масштабируемость предложенного алгоритма по сравнению с алгоритмом STRAIGHTFORWARD [12] и реализацией вычисления MEP с помощью функции из библиотеки Intel Math Kernel Library (MKL)[4], оптимизированной для Intel Xeon Phi. Алгоритм BLOCKWISE был реализован для компоновок данных AoS, SoA и ASA-512. Все алгоритмы запускались на Intel MIC для разных наборов данных. Измерялось время работы алгоритма без учета времени загрузки данных и записи результата. На основе полученных значений вычислялось ускорение и параллельная эффективность алгоритмов, определяемые следующим образом.

Ускорение и параллельная эффективность параллельного алгоритма, запускаемого на k нитях, вычисляется как $s(k) = \frac{t_1}{t_k}$ и $e(k) = \frac{s(k)}{k}$ соответственно, где t_1 и t_k — время работы алгоритма на одной и k нитях соответственно.

Было проведено исследование производительности и масштабируемости предложенного алгоритма для квадратных и прямоугольных MEP. В случае прямоугольных матриц использовались те же тестовые данные, что и в работе [12].

Чтобы убедиться, что предложенная вычислительная схема дает выигрыш от векторизации вычислений на MIC системах, было проведено сравнение производительности

[4]Intel Math Kernel Library 2018 Release Notes.

алгоритмов Blockwise, Straightforward и Intel MKL на Intel Xeon и Intel Xeon Phi на одних и тех же наборах данных.

Далее, было выполнено сравнение производительности алгоритма вычисления MEP, предложенного в работе [10], выполняющегося на системе NVIDIA Tesla C2050[5], и алгоритма Blockwise, выполняющегося на Intel Xeon Phi (поскольку указанные системы имеют примерно одинаковую пиковую производительность).

Наконец, приведены экспериментальные результаты, которые обосновывают выбор значения параметра *block*.

Наборы данных. В экспериментах использовались наборы данных, описанные в табл. 1. Наборы данных Census [14] и FCS Human [6] взяты из реальных приложений. Наборы данных MixSim и ADS являются синтетическими и получены с помощью программ-генераторов, описанных в работах [15] и [16] соответственно. Группа наборов данных ADS (Aligned Data Set) использовалась для экспериментальной оценки алгоритма Straightforward в работе [12]. Группа наборов данных PRND (Pseudo Random Numbers) использовалась в экспериментах в работе [10].

Таблица 1

Наборы данных для экспериментов

Набор	d	n	m	**Вид**	**Семантика**
MixSim	5	$35 \cdot 2^{10}$	$35 \cdot 2^{10}$	Синтетический	Получен генератором из [15]
Census	67	$35 \cdot 2^{10}$	$35 \cdot 2^{10}$	Реальный	Результаты переписи населения США [14]
FCS Human	423	$18 \cdot 2^{10}$	$18 \cdot 2^{10}$	Реальный	Агрегированная информация о геноме человека [6]
ADS-16	16	10^6	10^3	Синтетический	Наборы данных из [12]
ADS-32	32				
ADS-64	64				
ADS-256	256				
PRND-50	50	$15 \cdot 10^3$	$15 \cdot 10^3$	Синтетический	Наборы данных из [10]
PRND-100	100				
PRND-150	150				
PRND-200	200				

В экспериментах с наборами данных MixSim, Census и FCS Human брались подмножества этих наборов, чтобы количество точек было кратно значению параметра $block = 512$ (см. раздел 2.1).

Для оценки алгоритма Straightforward на наборах данных, в которых размерность точек d не кратна ширине векторного регистра $width_{VPU} = 16$, значение d увеличено до ближайшего целого, кратного 16, путем дополнения исходных данных фиктивными нулевыми координатами. Для оценки алгоритма Straightforward на наборах данных, использованных в работах [12] и [10], значения n и m были увеличены до ближайшего целого, кратного значению параметра $block = 512$, путем дополнения указанных множеств фиктивными нулевыми точками.

[5]NVIDIA Tesla C2050/C2070 Data sheet.

Аппаратное обеспечение. Эксперименты проведены на узле вычислительного кластера Торнадо ЮУрГУ [11]. Характеристики хост и MIC системы узла приведены в табл. 2.

Таблица 2

Характеристики аппаратной платформы

Характеристика	**Хост**	**Сопроцессор MIC**
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

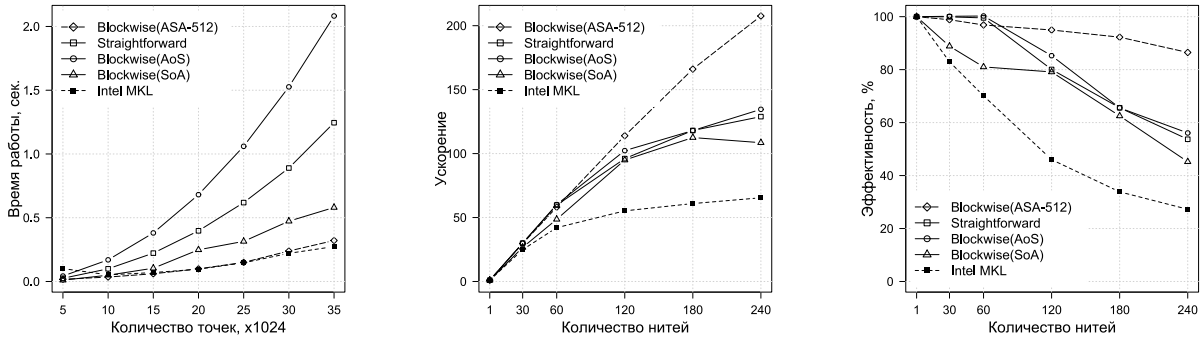
3.2. Результаты и обсуждение

Масштабируемость. На рис. 2 и рис. 3 показаны время работы, масштабируемость и параллельная эффективность алгоритмов на квадратных и прямоугольных MEP соответственно.

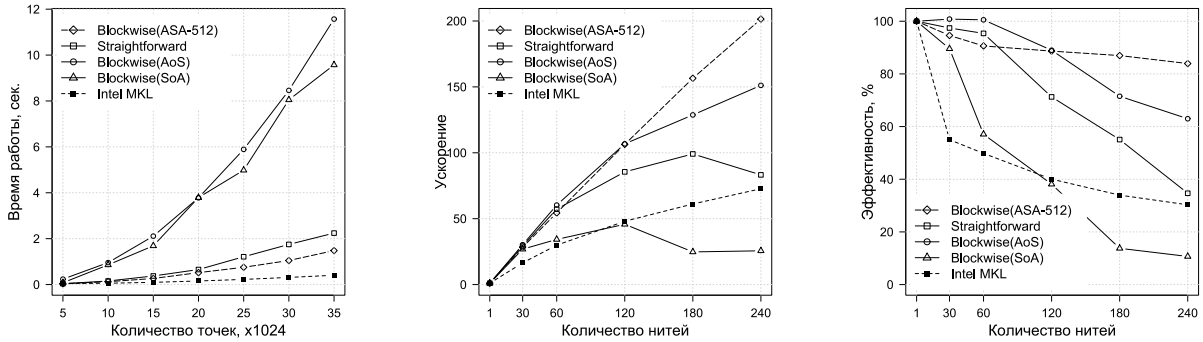
По результатам вычислений квадратных MEP мы видим, что наилучший результат показал алгоритм Intel MKL, а алгоритм Blockwise(ASA-512) занимает второе место и почти сравнивается с Intel MKL на наборе данных MixSim, когда значение d выровнено на 16. В то же время Intel MKL показывает почти худшую масштабируемость и параллельную эффективность среди исследуемых алгоритмов. Все алгоритмы, за исключением Intel MKL и Blockwise(SoA) показывают близкое к линейному ускорение и эффективность примерно 80%, когда количество нитей равно количеству физических ядер аппаратной платформы. При этом, если количество нитей превышает количество ядер, то только алгоритм Blockwise(ASA-512) сохраняет ранее описанное поведение, показывая ускорение до 200 и эффективность как минимум 80%. В то же время ускорение и параллельная эффективность других алгоритмов перестает увеличиваться или даже резко уменьшается.

Эксперименты с вычислением прямоугольных матриц используют наборы данных большего объема и показывают следующие результаты. Алгоритм Blockwise(ASA-512) превосходит по скорости работы все прочие алгоритмы на наборах данных ADS-16 и ADS-32 и показывает сравнимый результат с алгоритмом Intel MKL на наборе ADS-64. На наборе данных ADS-256 алгоритм Intel MKL превосходит все прочие алгоритмы. Что касается масштабируемости, можно видеть примерно ту же картину, что и для квадратных матриц. Алгоритм Blockwise(ASA-512) показывает ускорение, близкое к линейному, и параллельную эффективность 90%, если количество нитей совпадает с количеством физических ядер. В диапазоне от 60 до 240 нитей наш алгоритм показывает лучшую масштабируемость, ускоряясь до 160 раз и показывая параллельную эффективность 70%. Можно заключить, что алгоритм Blockwise(ASA-512) показывает лучшие результаты на прямоугольных матрицах малой размерности (примерно когда $d \leqslant 32$).

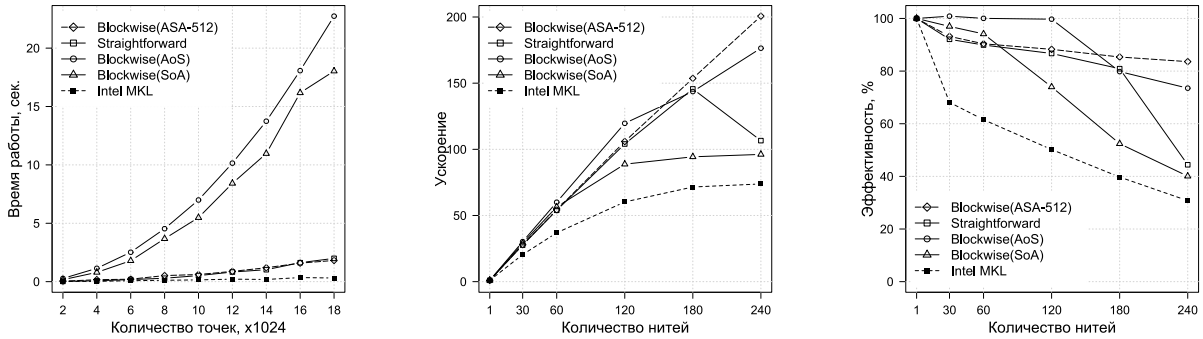
Преимущества векторизации. Табл. 3 показывает производительность алгоритма Blockwise(ASA-512) на системах Intel Xeon и Intel Xeon Phi в сравнении с алгоритмом



a) Набор данных MixSim (d выровнено до 16): время работы, ускорение и эффективность



b) Набор данных Census (d выровнено до 80): время работы, ускорение и эффективность

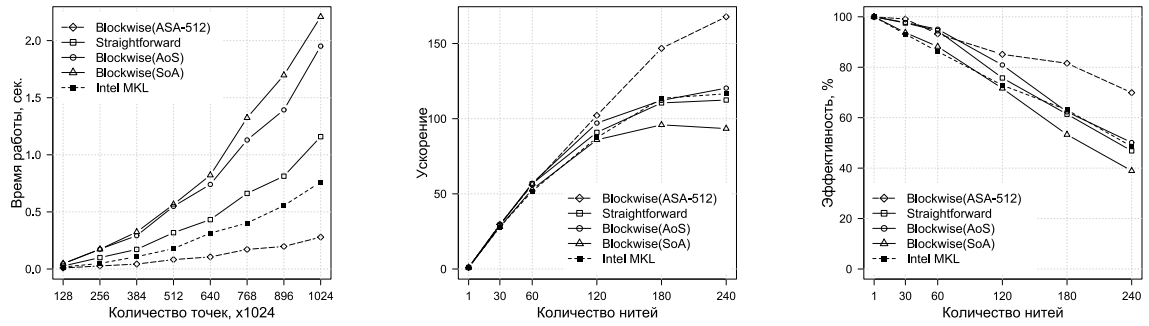


c) Набор данных FCS Human (d выровнено до 432): время работы, ускорение и эффективность

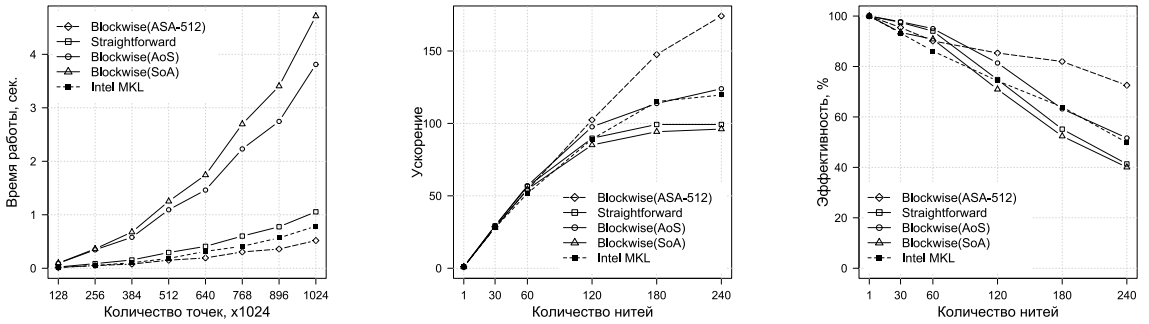
Рис. 2. Время работы и масштабируемость алгоритма на квадратных матрицах

Straightforward. Как можно увидеть, алгоритм Blockwise(ASA-512) от $3,5$ до 8 раз быстрее на Intel Xeon Phi, чем на хосте с двумя процессорами Intel Xeon. Алгоритм Straightforward, также как и Blockwise(ASA-512), быстрее работает на Intel Xeon Phi, чем на Intel Xeon. В то же время предложенный в данной работе алгоритм показывает лучшее время на указанных платформах. Отметим также, что алгоритм Intel MKL превосходит Blockwise(ASA-512) на данных большой размерности (примерно при $d > 32$) на обеих платформах.

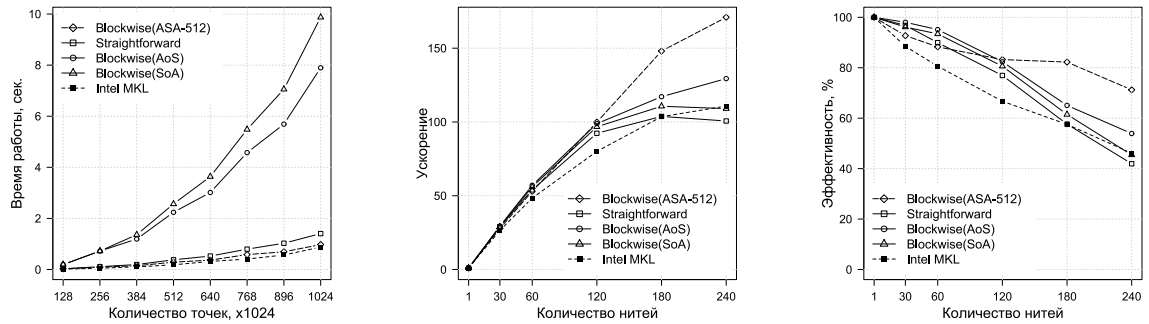
Сравнение с GPU. Сравнение производительности предложенного в данной работе алгоритма с алгоритмом из работы [10] приведено в табл. 4. Как видно, алгоритм



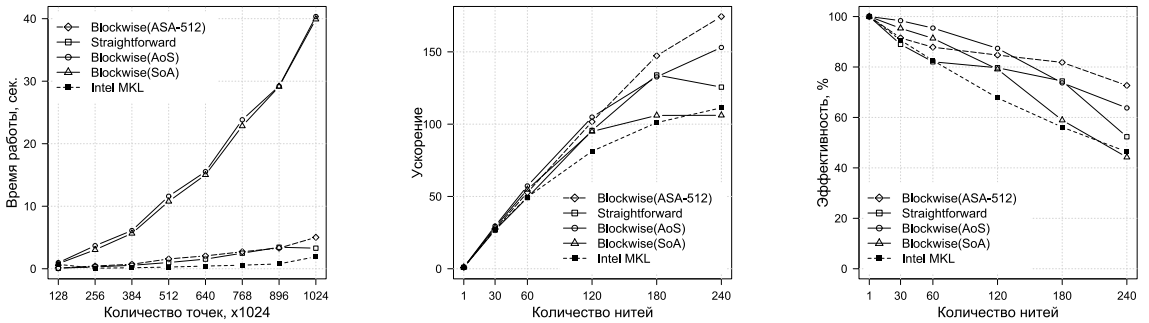
a) Набор данных ADS-16: время работы, ускорение и эффективность



b) Набор данных ADS-32: время работы, ускорение и эффективность



c) Набор данных ADS-64: время работы, ускорение и эффективность



d) Набор данных ADS-256: время работы, ускорение и эффективность

Рис. 3. Время работы и масштабируемость алгоритма на прямоугольных матрицах

Таблица 3

Время работы на наборах данных ADS, с

Набор	**Intel Xeon Phi (KNC)** 1,076 TFLOPS			2×**Intel Xeon CPU** 0,371 TFLOPS			**Соотношение** 2×**CPU**/**Phi**	
	Blockwise (ASA-512)	Intel MKL	Straight-forward	Blockwise (ASA-512)	Intel MKL	Straight-forward	Blockwise (ASA-512)	Straight-forward
ADS-16	0,28	0,76	1,05	1,04	3,02	1,00	3,7×	1,0×
ADS-32	0,51	0,78	1,15	1,76	3,14	1,79	3,5×	1,6×
ADS-64	0,98	0,88	1,36	3,78	3,81	4,25	3,9×	3,1×
ADS-256	3,71	1,92	3,79	30,32	5,14	31,41	8,2×	8,3×

Blockwise(ASA-512) до двух раз быстрее на Intel Xeon Phi, чем алгоритм для NVIDIA Tesla C2050. При этом алгоритм Intel MKL работает быстрее, чем Blockwise(ASA-512) на этих наборах данных.

Таблица 4

Время работы на наборах данных PRND, с

Набор	**Intel Xeon Phi** 1,076 TFLOPS		2×**Intel Xeon** 0,371 TFLOPS		**NVIDIA Tesla** 1,03 TFLOPS
	Blockwise (ASA-512)	Intel MKL	Blockwise (ASA-512)	Intel MKL	Ким (Kim) и др. [10]
PRND-50	0,19	0,07	0,35	0,74	0,82
PRND-100	0,32	0,08	0,59	0,89	1,01
PRND-150	0,45	0,10	0,78	1,01	1,21
PRND-200	0,58	0,12	1,60	1,16	1,41

Выбор параметра *block***.** Приведенные выше результаты экспериментов были получены после эмпирического выбора параметра *block*. Значение $block = 512$ было подобрано следующим образом. Алгоритм Blockwise(ASA-*block*) был запущен на Intel Xeon Phi на серии искусственных наборов данных из $n = m = 2^{15}$ случайных точек с размерностью d, равной 3, 5, 67 и 129 с различными значениями *block* (см. рис. 4). После этого значение $block = 512$ было выбрано в качестве значения, дающего наилучшую производительность для большинства значений размерности d.

Обсуждение. В заключении представления результатов экспериментов с предложенным алгоритмом обсудим его накладные расходы на оперативную память и вычисления.

Накладные расходы на оперативную память возникают по следующим причинам. Во-первых, для эффективного использования возможностей векторизации вычислений процессором Intel Xeon Phi алгоритм Blockwise требует, чтобы мощность второго множества точек была кратна параметру *block*. Если это не так, то значение m необходимо увеличить до ближайшего целого, кратного *block*, с помощью дополнения исходного набора данных фиктивными нулевыми точками. Таким образом, в худшем случае будет добавлено $d \cdot (block - 1)$ избыточных нулевых элементов. Во-вторых, до вычисления матрицы расстояний создается копия матрицы, представляющей второе множество точек,

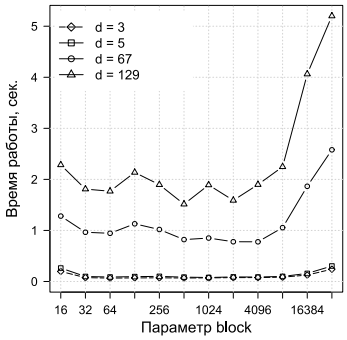


Рис. 4. Производительность алгоритма Blockwise(ASA-*block*) для различных значений параметра *block*

которая заполнятся с помощью операции перестановки элементов исходной матрицы. Поэтому требуется дополнительно $d \cdot \max(n, m)$ элементов в памяти. Здесь функция max используется для увеличения эффективности векторизации при вычислении матрицы расстояний, поскольку матрица **B** должна представлять наибольший из двух наборов точек. В итоге, в худшем случае потребуется $d \cdot (block - 1 + \max(n, m))$ избыточных элементов данных.

Алгоритм Straightforward, в отличие от предложенного решения, требует, чтобы значение размерности пространства d было кратно ширине векторного регистра $width_{VPU}$. Если это условие не выполнено, то точки исходных данных должны быть дополнены фиктивными нулевыми координатами. Таким образом, в худшем случае будет добавлено $(width_{VPU} - 1) \cdot (m + n)$ нулевых элементов. Возвращаясь к результатам экспериментов, сравнивающих алгоритмы Blockwise(ASA-512) и Straightforward, мы видим, что при вычислении прямоугольных MEP для данных малой размерности предложенный алгоритм имеет меньшие накладные расходы по памяти, чем алгоритм Straightforward.

Накладные расходы по времени представляют собой перестановку элементов исходной матрицы для изменения компоновки данных. Эксперименты показали, что время, затрачиваемое на изменение компоновки, пренебрежимо мало, и по сравнению со временем работы алгоритма составляет менее 1%.

В завершении обсуждения следует напомнить, что производительность алгоритма Blockwise(ASA-*block*) зависит от параметра *block*, значение которого определяется эмпирическим путем.

Заключение

В данной работе исследована проблема вычисления матрицы Евклидовых расстояний (MEP), часто возникающая как подзадача в большом количестве практических и научных задач, связанных с интеллектуальным анализом данных. К настоящему времени разработано большое количество алгоритмов вычисление MEP на графических процессорах, однако эти разработки не могут быть напрямую перенесены на современные многоядерные системы Intel Xeon Phi, являющиеся перспективной альтернативой GPU. В статье исследована проблема вычисления MEP на платформе Intel Xeon Phi Knights Landing (KNL) для случая, когда данные могут быть размещены в оперативной памяти.

В работе предложен новый параллельный алгоритм для вычисления MEP, названный Blockwise и имеющий два ключевых отличия от прямолинейной реализации,

использующей выравнивание данных и автовекторизацию. Во-первых, алгоритм использует блочно-ориентированную схему организации вычислений, которая обеспечивает эффективное использование векторных операций Intel Xeon Phi. Во-вторых, применена нетривиальная компоновка данных в оперативной памяти для уменьшения количества кэш-промахов процессора во время вычислений.

Проведено экспериментальное исследование предложенного алгоритма на синтетических и реальных наборах данных для вычисления квадратных и прямоугольных матриц, и выполнено сравнение с аналогами. Алгоритм Blockwise показал близкое к линейному ускорение и эффективность не ниже 80%, когда количество нитей совпадает с количеством физических ядер на используемой платформе. Когда алгоритм Blockwise задействует более одной нити на физическое ядро, его ускорение и параллельная эффективность становятся сублинейными, превосходя, однако, при этом указанные характеристики алгоритмов-конкурентов. Предложенный алгоритм превосходит прямолинейный подход и алгоритм из Intel Math Kernel Library (MKL) в случае прямоугольной матрицы и точками малой размерности (примерно $d \leqslant 32$). На точках бóльшей размерности ($d > 32$) алгоритм Intel MKL превосходит другие алгоритмы как на квадратных,так и на прямоугольных матрицах, в то время как алгоритм Blockwise показывает примерно ту же производительность, что и прямолинейный подход.

Исследование вычисления матрицы Евклидовых расстояний на процессорах Intel MIC может быть продолжено в следующих направлениях: приложение предложенного алгоритма к различным алгоритмам кластеризации (например, k-means [12], PAM [17] и др.), разработка аналитической модели, которая будет предсказывать производительность алгоритма Blockwise и определять значение параметра $block$, обеспечивающее наилучшую производительность алгоритма.

Работа выполнена при финансовой поддержке Российского фонда фундаментальных исследований (грант № 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).

Литература

1. Arefin A.S., Riveros C., Berretta R., Moscato P. Computing Large-scale Distance Matrices on GPU // Proceedings of the 7th International Conference on Computer Science and Education, ICCSE 2012, July 14–17, 2012, Melbourne, Australia. IEEE Computer Society, 2012. P. 576–580. DOI: 10.1109/ICCSE.2012.6295141.

2. Chang D., Jones N.A., Li D., Ouyang M., Ragade R.K. Compute Pairwise Euclidean Distances of Data Points with GPUs // Proceedings of the IASTED International Symposium on Computational Biology and Bioinformatics, CBB'2008, November 16–18, 2008, Orlando, Florida, USA. IASTED, 2008. P. 278–283.

3. Chrysos G. Intel Xeon Phi Coprocessor (Codename Knights Corner) // Proceedings of the 2012 IEEE Hot Chips 24th Symposium (HCS), Cupertino, CA, USA, August 27–29, 2012. P. 1–31. DOI: 10.1109/HOTCHIPS.2012.7476487

4. Dembélé D., Kastner P. Fuzzy c-Means Method for Clustering Microarray Data // Bioinformatics. 2003. Vol. 19, No. 8. P. 973–980. DOI: 10.1093/bioinformatics/btg119

5. Dokmanic I., Parhizkar R., Ranieri J., Vetterli M. Euclidean Distance Matrices: Essential Theory, Algorithms, and Applications // IEEE Signal Processing Magazine. 2015. Vol. 32, No. 6. P. 12–30. DOI: 10.1109/MSP.2015.2398954

6. Engreitz J.M., Daigle B.Jr., Marshall J.J., Altman R.B. Independent Component Analysis: Mining Microarray Data for Fundamental Human Gene Expression Modules // Journal of Biomedical Informatics. 2010. Vol. 43, No. 6. P. 932–944. DOI: 0.1016/j.jbi.2010.07.001

7. Foote J. An Overview of Audio Information Retrieval // Multimedia Systems. 1999. Vol. 7, No. 1. P. 2–10. DOI: 10.1007/s005300050106

8. Hassan Q.F. Innovative Research and Applications in Next-generation High Performance computing. IGI Global, 2016. DOI: 10.4018/978-1-5225-0287-6.

9. Jaros M., Strakos P., Karásek T., et al. Implementation of k-Means Segmentation Algorithm on Intel Xeon Phi and GPU: Application in Medical Imaging // Advances in Engineering Software. 2017. Vol. 103. P. 21–28. DOI: 10.1016/j.advengsoft.2016.05.008

10. Kim S., Ouyang M. Compute Distance Matrices with GPU // Proceedings of the 3rd Annual International Conference on Advances in Distributed and Parallel Computing, ADPC'2012, 17–18 September, 2012, Bali, Indonesia. DOI: 10.5176/2251-1652_ADPC12.07

11. Kostenetskiy P., Safonov A. SUSU Supercomputer Resources // PCT'2016, International Scientific Conference on Parallel Computational Technologies, Arkhangelsk, Russia, March 29–31, 2016. CEUR Workshop Proceedings. Vol. 1576, CEUR-WS.org, 2016. P. 561–573.

12. Lee S., Liao W., Agrawal A., Hardavellas N., Choudhary A.N. Evaluation of k-Means Data Clustering Algorithm on Intel Xeon Phi // Proceedings of the 2016 IEEE International Conference on Big Data, BigData 2016, Washington DC, USA, December 5–8, 2016. IEEE Computer Society, 2016. P. 2251–2260. DOI: 10.1109/BigData.2016.7840856

13. Li Q., Kecman V., Salman R. A Chunking Method for Euclidean Distance Matrix Calculation on Large Dataset Using Multi-GPU // Proceedings of the 9th International Conference on Machine Learning and Applications, ICMLA 2010, Washington, DC, USA, 12–14 December 2010. IEEE Computer Society, 2010. P. 208–213. DOI: 10.1109/ICMLA.2010.38

14. Meek C., Thiesson B., Heckerman D. The Learning-Curve Sampling Method Applied to Model-Based Clustering // Journal of Machine Learning Research. 2002. Vol. 2. P. 397–418.

15. Melnykov V., Chen W.C., Maitra R. MixSim: An R Package for Simulating Data to Study Performance of Clustering Algorithms // Journal of Statistical Software. 2012. Vol. 51, No. 12. P. 1–25. DOI: 10.18637/jss.v051.i12

16. Narayanan R., Ozisikyilmaz B., Zambreno J., Memik G., Choudhary A.N. MineBench: A Benchmark Suite for Data Mining Workloads // Proceedings of the 2006 IEEE International Symposium on Workload Characterization, IISWC 2006, October 25–27, 2006, San Jose, California, USA. IEEE Computer Society, 2006. P. 182–188. DOI: 10.1109/IISWC.2006.302743

17. Rechkalov T.V., Zymbler M.L. Accelerating Medoids-based Clustering with the Intel Many Integrated Core Architecture // Proceedings of the 9th International Conference on Application of Information and Communication Technologies (AICT'2015), October 14–16, 2015, Rostov-on-Don, Russia. IEEE Computer Society, 2015. P. 413–417. DOI: 10.1109/ICAICT.2015.7338591

18. Sodani A. Knights Landing (KNL): 2nd Generation Intel Xeon Phi Processor // Proceedings of the 2015 IEEE Hot Chips 27th Symposium (HCS), Cupertino, CA, USA, August 22–25, 2015. IEEE Computer Society, 2015. P. 1–24. DOI: 10.1109/HOTCHIPS.2015.7477467

19. Valenzise G., Gerosa L., Tagliasacchi M., Antonacci F., Sarti A. Scream and Gunshot Detection and Localization for Audio-surveillance Systems // Proceedings of the 4th IEEE International Conference on Advanced Video and Signal Based Surveillance, AVSS 2007, 5–7 September, 2007, Queen Mary, University of London, London, United Kingdom. IEEE Computer Society, 2007. P. 21–26. DOI: 10.1109/AVSS.2007.4425280

20. Wu F., Wu Q., Tan Y., Wei L., Shao L., Gao L. A Vectorized k-Means Algorithm for Intel Many Integrated Core Architecture // Proceedings of the 10th International Symposium on Advanced Parallel Processing Technologies, APPT 2013, Stockholm, Sweden, August 27–28, 2013, Revised Selected Papers. Lecture Notes in Computer Science. Springer, 2013. Vol. 8299. P. 277–294. DOI: 10.1007/978-3-642-45293-2_21

21. Zou J., Chen L., Chen C.L.P. Ensemble Fuzzy c-Means Clustering Algorithms Based on KL-Divergence for Medical Image Segmentation // Proceedings of the 2013 IEEE International Conference on Bioinformatics and Biomedicine, Shanghai, China, December 18–21, 2013. IEEE Computer Society, 2013. P. 291–296. DOI: 10.1109/BIBM.2013.6732505

Речкалов Тимофей Валерьевич, аспирант, кафедра системного программирования, Южно-Уральский государственный университет (национальный исследовательский университет) (Челябинск, Российская Федерация)

Цымблер Михаил Леонидович, к.ф.-м.н., доцент, кафедра системного программирования, Южно-Уральский государственный университет (национальный исследовательский университет) (Челябинск, Российская Федерация)

DOI: 10.14529/cmse180305

A PARALLEL ALGORITHM OF EUCLIDEAN DISTANCE MATRIX COMPUTATION FOR THE INTEL XEON PHI KNIGHTS LANDING MANY-CORE PROCESSOR

© 2018 T.V. Rechkalov, M.L. Zymbler

South Ural State University (pr. Lenina 76, Chelyabinsk, 454080 Russia)

E-mail: trechkalov@yandex.ru, mzym@susu.ru

Received: 06.05.2018

Computation of a Euclidean distance matrix (EDM) is a typical task in a wide spectrum of problems connected with data mining. Currently, many parallel algorithms for this task have been developed for graphical processors. These developments, however, cannot be directly applied to the Intel Many Integrated Core systems. In this paper, we suggest a parallel algorithm for EDM computation on Intel Xeon Phi Knights Landing processor in the case when the input data fit into the main memory. The algorithm exploits block-oriented scheme of computations that allows for the efficient utilization of Intel Xeon Phi vectorization abilities. In the algorithm, we also apply apply a sophisticated data layout to store data points in main memory so as to reduce the number of processor cache misses during EDM computations. Experimental evaluation of the algorithm on real-world and synthetic datasets shows that it is highly scalable and outruns analogues in the case of rectangular matrices with low-dimensional data points.

Keywords: Euclidean distance matrix, OpenMP, Intel Xeon Phi, Knights Landing, data layout, vectorization.

FOR CITATION

Rechkalov T.V., Zymbler M.L. A Parallel Algorithm of Euclidean Distance Matrix Computation for the Intel Xeon Phi Knights Landing Many-core Processor. *Bulletin of the South Ural State University. Series: Computational Mathematics and Software Engineering.* 2018. vol. 7, no. 3. pp. 65–82. (in Russian) DOI: 10.14529/cmse180305.

References

1. Arefin A.S., Riveros C., Berretta R., Moscato P. Computing Large-scale Distance Matrices on GPU. Proceedings of the 7th International Conference on Computer Science and Education, ICCSE 2012, July 14–17, 2012, Melbourne, Australia. IEEE Computer Society, 2012. pp. 576–580. DOI: 10.1109/ICCSE.2012.6295141.
2. Chang D., Jones N.A., Li D., Ouyang M., Ragade R.K. Compute Pairwise Euclidean Distances of Data Points with GPUs. Proceedings of the IASTED International Symposium on Computational Biology and Bioinformatics, CBB'2008, November 16–18, 2008, Orlando, Florida, USA. IASTED, 2008. pp. 278–283.
3. Chrysos G. Intel Xeon Phi Coprocessor (Codename Knights Corner). Proceedings of the 2012 IEEE Hot Chips 24th Symposium (HCS), Cupertino, CA, USA, August 27–29, 2012. pp. 1–31. DOI: 10.1109/HOTCHIPS.2012.7476487
4. Dembélé D., Kastner P. Fuzzy c-Means Method for Clustering Microarray Data. Bioinformatics. 2003. vol. 19, no. 8. pp. 973–980. DOI: 10.1093/bioinformatics/btg119
5. Dokmanic I., Parhizkar R., Ranieri J., Vetterli M. Euclidean Distance Matrices: Essential Theory, Algorithms, and Applications. IEEE Signal Processing Magazine. 2015. vol. 32, no. 6. pp. 12–30. DOI: 10.1109/MSP.2015.2398954
6. Engreitz J.M., Daigle B.Jr., Marshall J.J., Altman R.B. Independent Component Analysis: Mining Microarray Data for Fundamental Human Gene Expression Modules. Journal of Biomedical Informatics. 2010. vol. 43, no. 6. pp. 932–944. DOI: 0.1016/j.jbi.2010.07.001
7. Foote J. An Overview of Audio Information Retrieval. Multimedia Systems. 1999. vol. 7, no. 1. pp. 2–10. DOI: 10.1007/s005300050106
8. Hassan Q.F. Innovative Research and Applications in Next-generation High Performance computing. IGI Global, 2016. DOI: 10.4018/978-1-5225-0287-6.
9. Jaros M., Strakos P., Karásek T., et al. Implementation of k-Means Segmentation Algorithm on Intel Xeon Phi and GPU: Application in Medical Imaging. Advances in Engineering Software. 2017. vol. 103. pp. 21–28. DOI: 10.1016/j.advengsoft.2016.05.008
10. Kim S., Ouyang M. Compute Distance Matrices with GPU. Proceedings of the 3rd Annual International Conference on Advances in Distributed and Parallel Computing, ADPC'2012, 17–18 September, 2012, Bali, Indonesia. DOI: 10.5176/2251-1652_ADPC12.07
11. Kostenetskiy P., Safonov A. SUSU Supercomputer Resources. PCT'2016, International Scientific Conference on Parallel Computational Technologies, Arkhangelsk, Russia,

March 29–31, 2016. CEUR Workshop Proceedings. vol. 1576, CEUR-WS.org, 2016. pp. 561–573.

12. Lee S., Liao W., Agrawal A., Hardavellas N., Choudhary A.N. Evaluation of k-Means Data Clustering Algorithm on Intel Xeon Phi. Proceedings of the 2016 IEEE International Conference on Big Data, BigData 2016, Washington DC, USA, December 5–8, 2016. IEEE Computer Society, 2016. pp. 2251–2260. DOI: 10.1109/BigData.2016.7840856

13. Li Q., Kecman V., Salman R. A Chunking Method for Euclidean Distance Matrix Calculation on Large Dataset Using Multi-GPU. Proceedings of the 9th International Conference on Machine Learning and Applications, ICMLA 2010, Washington, DC, USA, 12–14 December 2010. IEEE Computer Society, 2010. pp. 208–213. DOI: 10.1109/ICMLA.2010.38

14. Meek C., Thiesson B., Heckerman D. The Learning-Curve Sampling Method Applied to Model-Based Clustering. Journal of Machine Learning Research. 2002. vol. 2. pp. 397–418.

15. Melnykov V., Chen W.C., Maitra R. MixSim: An R Package for Simulating Data to Study Performance of Clustering Algorithms. Journal of Statistical Software. 2012. vol. 51, no. 12. pp. 1–25. DOI: 10.18637/jss.v051.i12

16. Narayanan R., Ozisikyilmaz B., Zambreno J., Memik G., Choudhary A.N. MineBench: A Benchmark Suite for Data Mining Workloads. Proceedings of the 2006 IEEE International Symposium on Workload Characterization, IISWC 2006, October 25–27, 2006, San Jose, California, USA. IEEE Computer Society, 2006. pp. 182–188. DOI: 10.1109/IISWC.2006.302743

17. Rechkalov T.V., Zymbler M.L. Accelerating Medoids-based Clustering with the Intel Many Integrated Core Architecture. Proceedings of the 9th International Conference on Application of Information and Communication Technologies (AICT'2015), October 14–16, 2015, Rostov-on-Don, Russia. IEEE Computer Society, 2015. pp. 413–417. DOI: 10.1109/ICAICT.2015.7338591

18. Sodani A. Knights Landing (KNL): 2nd Generation Intel Xeon Phi Processor. Proceedings of the 2015 IEEE Hot Chips 27th Symposium (HCS), Cupertino, CA, USA, August 22–25, 2015. IEEE Computer Society, 2015. pp. 1–24. DOI: 10.1109/HOTCHIPS.2015.7477467

19. Valenzise G., Gerosa L., Tagliasacchi M., Antonacci F., Sarti A. Scream and Gunshot Detection and Localization for Audio-surveillance Systems. Proceedings of the 4th IEEE International Conference on Advanced Video and Signal Based Surveillance, AVSS 2007, 5–7 September, 2007, Queen Mary, University of London, London, United Kingdom. IEEE Computer Society, 2007. pp. 21–26. DOI: 10.1109/AVSS.2007.4425280

20. Wu F., Wu Q., Tan Y., Wei L., Shao L., Gao L. A Vectorized k-Means Algorithm for Intel Many Integrated Core Architecture. Proceedings of the 10th International Symposium on Advanced Parallel Processing Technologies, APPT 2013, Stockholm, Sweden, August 27–28, 2013, Revised Selected Papers. Lecture Notes in Computer Science. Springer, 2013. vol. 8299. pp. 277–294. DOI: 10.1007/978-3-642-45293-2_21

21. Zou J., Chen L., Chen C.L.P. Ensemble Fuzzy c-Means Clustering Algorithms Based on KL-Divergence for Medical Image Segmentation. Proceedings of the 2013 IEEE International Conference on Bioinformatics and Biomedicine, Shanghai, China, December 18–21, 2013. IEEE Computer Society, 2013. pp. 291–296. DOI: 10.1109/BIBM.2013.6732505