УДК 004.272.25; 004.421; 004.032.24 doi 10.26089/NumMet.v20r211

ПАРАЛЛЕЛЬНЫЙ АЛГОРИТМ КЛАСТЕРИЗАЦИИ ДАННЫХ ДЛЯ МНОГОЯДЕРНЫХ УСКОРИТЕЛЕЙ INTEL MIC

Т. В. Речкалов[1], М. Л. Цымблер[2]

Алгоритм PAM (Partitioning Around Medoids) представляет собой разделительный алгоритм кластеризации, в котором в качестве центров кластеров выбираются только кластеризуемые объекты (медоиды). Кластеризация на основе техники медоидов применяется в широком спектре приложений: сегментирование медицинских и спутниковых изображений, анализ ДНК-микрочипов и текстов и др. На сегодня имеются параллельные реализации PAM для систем GPU и FPGA, но отсутствуют таковые для многоядерных ускорителей архитектуры Intel Many Integrated Core (MIC). В настоящей статье предлагается новый параллельный алгоритм кластеризации PhiPAM для ускорителей Intel MIC. Вычисления распараллеливаются с помощью технологии OpenMP. Алгоритм предполагает использование специализированной компоновки данных в памяти и техники тайлинга, позволяющих эффективно векторизовать вычисления на системах Intel MIC. Эксперименты, проведенные на реальных наборах данных, показали хорошую масштабируемость алгоритма.

Ключевые слова: кластеризация, медоид, параллельный алгоритм, OpenMP, Intel Xeon Phi, представление данных в памяти, векторизация вычислений.

1. Введение. Кластеризация является одной из базовых задач интеллектуального анализа данных и предполагает разбиение множества объектов сходной структуры на заранее неизвестные группы (*кластеры*) в зависимости от похожести свойств объектов.

Классический и относительно простой алгоритм кластеризации k-Means [17] выполняет итеративное разбиение объектов на кластеры, используя представление каждого кластера соответствующим *центроидом* (точка, получаемая посредством покоординатного усреднения объектов кластера). Качество кластеризации k-Means, однако, может ухудшаться из-за шумов и выбросов в исходных данных. Алгоритмы k-Medoids и PAM (Partitioning Around Medoids) [12] решают данную проблему путем представления кластера соответствующим *медоидом* — объектом исходного множества, который находится ближе остальных к центроиду. Алгоритмы на базе техники медоидов более устойчивы к выбросам и шумам в данных и применяются в широком спектре приложений, связанных с кластеризацией данных (анализ изображений [14, 20], ДНК-микрочипов [19], текстов [21], сетевого трафика [7] и др.). В настоящее время разработаны многочисленные параллельные реализации указанных алгоритмов для платформ NVIDIA GPU (Graphic Processing Units) [9, 13, 14] и FPGA (Field Programmable Gate Arrays) [20], но отсутствуют таковые для многоядерных ускорителей архитектуры Intel MIC. Целью данного исследования является ускорение алгоритма PAM на платформе Intel MIC.

Многоядерные ускорители архитектуры Intel MIC [6] являются конкурентоспособной альтернативой более распространенным системам GPU и FPGA. Системы Intel 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 ускорителей MIC, имеющих базовое имя Xeon Phi: сопроцессор Knights Corner (KNC) [5] с 57–61 ядрами и процессор Knights Landing (KNL) [23] с 64–72 ядрами. Выгода от применения ускорителей MIC, как правило, проявляется в приложениях, где обработка больших объемов данных (от сотен тысяч элементов) может быть организована в виде циклов, 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 [4].

[1] Южно-Уральский государственный университет, Высшая школа электроники и компьютерных наук, просп. Ленина, 76, 454080, Челябинск; аспирант, e-mail: trechkalov@yandex.ru

[2] Южно-Уральский государственный университет, Высшая школа электроники и компьютерных наук, просп. Ленина, 76, 454080, Челябинск; начальник отдела, e-mail: mzym@susu.ru

В настоящей статье предлагается новый параллельный алгоритм *PhiPAM* для кластеризации данных на основе техники медоидов для многоядерных ускорителей архитектуры Intel MIC. Статья организована следующим образом. Раздел 2 содержит описание последовательного алгоритма PAM и краткий обзор работ по тематике исследования. Предлагаемый параллельный алгоритм *PhiPAM* представлен в разделе 3. В разделе 4 описаны вычислительные эксперименты, исследующие эффективность предложенного алгоритма. Заключение резюмирует результаты, полученные в рамках исследования.

2. Постановка задачи и обзор работ. Задача *кластеризации* заключается в разбиении множества объектов сходной структуры на заранее неизвестные группы в зависимости от похожести свойств объектов и формально может быть определена следующим образом. Пусть заданы конечные множества $X = \{x_1, x_2, \ldots, x_n\}$, где $n > 1$ — множество объектов d-мерного метрического пространства, для которых задана функция расстояния $\rho(x_i, x_j)$, и $C = \{c_1, c_2, \ldots, c_k\}$, где $k \ll n$ — набор уникальных идентификаторов (номеров, имен, меток) кластеров.

Алгоритм кластеризации определяется как функция $\alpha : X \to C$, которая каждому объекту назначает уникальный идентификатор кластера. Алгоритм кластеризации выполняет разбиение множества X на непересекающиеся непустые подмножества (*кластеры*) таким образом, чтобы каждый кластер состоял из объектов, близких по метрике ρ, а объекты разных кластеров существенно отличались. В нашей работе в качестве метрики используется расстояние Евклида.

Разделительный алгоритм кластеризации предполагает два этапа работы. На первом этапе объекты исходного множества распределяются по кластерам (возможно, случайным образом) так, что в каждом кластере имеется по крайней мере один объект, и каждый объект принадлежит в точности одному кластеру. На втором этапе итеративно выполняется перемещение объектов между кластерами с целью улучшить начальное разбиение (чтобы объекты из одного кластера были более "близкими", а из разных кластеров — более "далекими" друг другу) до достижения определенного критерия. При этом могут использоваться эвристики, поскольку поиск всех возможных разбиений может привести к большим накладным расходам. В классическом разделительном алгоритме k-Means [17] при улучшении разбиения каждый кластер представляется своим *центроидом*, получаемым посредством покоординатного усреднения объектов кластера. Алгоритм *PAM (Partitioning Around Medoids)* [12] в качестве представления кластера использует *медоид* — объект исходного множества, который находится ближе остальных к центроиду. Техника медоидов обеспечивает повышение *робастности* алгоритма (устойчивости к выбросам и шумам в данных).

Алгоритм 1. PAM(in X, k; out C)

▷ Фаза Build

1: Инициализировать C

▷ Фаза Swap

2: **repeat**
3: Найти $\{T_{\min}, c_{\min}, x_{\min}\}$
4: Поменять местами $c_{\min}$ и $x_{\min}$
5: **until** $T_{\min} < 0$
6: **return** C

Алгоритм PAM (см. алгоритм 1) состоит из двух фаз: Build и Swap. На *фазе Build* выполняется инициализация множества медоидов, которые затем уточняются на *фазе Swap*. На фазе Build в качестве первого медоида c_1 выбирается объект, сумма расстояний от которого до всех остальных объектов является наименьшей:

$$c_1 = \arg \min_{1 \leqslant h \leqslant n} \sum_{j=1}^{n} \rho(x_h, x_j).$$

Для выбора остальных медоидов используется целевая функция $E : X \times X \to \mathbb{R}$, определяемая как сумма расстояний от каждого объекта до ближайшего к нему медоида:

$$E = \sum_{j=1}^{n} \min_{1 \leqslant i \leqslant k} \rho(c_i, x_j).$$

При выборе каждого последующего медоида производится вычисление целевой функции относительно объектов, ранее выбранных в качестве медоидов, и каждого из еще не выбранных объектов для мини-

мизации целевой функции:

$$
\begin{aligned}
c_2 &= \arg\min_{1\leqslant h\leqslant n}\sum_{j=1}^{n}\min\bigl(\rho(c_1,x_j),\ \rho(x_h,x_j)\bigr),\\
c_3 &= \arg\min_{1\leqslant h\leqslant n}\sum_{j=1}^{n}\min\left(\min_{1\leqslant \ell\leqslant 2}\bigl(\rho(c_\ell,x_j)\bigr),\ \rho(x_h,x_j)\right),\\
&\cdots\cdots\cdots\\
c_k &= \arg\min_{1\leqslant h\leqslant n}\sum_{j=1}^{n}\min\left(\min_{1\leqslant \ell\leqslant k-1}\bigl(\rho(c_\ell,x_j)\bigr),\ \rho(x_h,x_j)\right).
\end{aligned}
$$

На фазе Swap алгоритм пытается изменить множество медоидов C таким образом, чтобы улучшить значение целевой функции E. PAM выполняет поиск пары объектов $(c_{\min}, x_{\min})$, минимизирующих целевую функцию. Для этого перебираются все пары объектов (c_i, x_h), где c_i — медоид, а x_h не является медоидом, и вычисляется изменение целевой функции при исключении c_i из множества медоидов и включении вместо него x_h. Указанное изменение обозначается как T_{ih}, а его минимальное значение, достигаемое на паре $(c_{\min}, x_{\min})$, как $T_{\min}$. Если $T_{\min} > 0$, то множество C не может быть улучшено и фаза Swap и алгоритм в целом завершаются. Объект исходного множества принадлежит тому кластеру, медоид которого является ближайшим к данному объекту.

Для вычисления T_{ih} используются множества D, $S \in \mathbb{R}^n$, определяемые следующим образом:

$$
\begin{aligned}
D &= \left\{ d_i \middle| d_i = \rho(x_i, c_{\min_1}),\ c_{\min_1} := \arg\min_{c_j\in C}\rho(x_i,c_j)\right\},\\
S &= \left\{ s_i \middle| s_i = \rho(x_i, c_{\min_2}),\ c_{\min_2} := \arg\min_{c_j\in C\setminus c_{\min_1}}\rho(x_i,c_j)\right\}.
\end{aligned}
$$

Иными словами, D — это множество расстояний от каждого объекта до ближайшего медоида, S — множество расстояний от каждого объекта до второго ближайшего медоида. Вклад не-медоида x_j в вычисление T_{ih} при замене c_i на x_h обозначается как δ_{jih}, и T_{ih} вычисляется следующим образом:

$$
T_{ih} = \sum_{j=1}^{n}\delta_{jih}.
$$

Величина δ_{jih} вычисляется как показано в алгоритме 2.

Алгоритмы кластеризации на базе техники медоидов применяются в широком спектре приложений: сегментирование медицинских [20] и спутниковых [14] изображений, анализ ДНК-микрочипов [19], автоматизированная рубрикация текстов [21], компьютерная безопасность [7] и др.

В работах [9, 13] предложены параллельные реализации алгоритма k-Medoids для GPU. Для многоядерных ускорителей архитектуры Intel MIC предложены параллельные версии алгоритма k-Means в работах [10, 15, 26].

В работе [20] описана архитектура распараллеливания алгоритма PAM для FPGA и расчеты потенциального достижимого ускорения, однако соответствующая реализация не представлена. В работе [14] представлен алгоритм *GPUPAM* — параллельная реализация алгоритма PAM для графического процессора. Алгоритм GPUPAM предполагает создание двумерного массива нитей CUDA размера $k\times(n-k)$ для подсчета целевой функции, где каждая нить выполняет задачу обмена местами медоида и не-медоида.

3. Параллельный алгоритм PhiPAM. Алгоритм *PhiPAM* [22] (см. алгоритм 3) представляет собой параллельную версию описанного выше алгоритма PAM для многоядерных ускорителей архитектуры Intel MIC. В отличие от последовательного алгоритма, PhiPAM выполняет предварительное вычисление расстояний между каждой парой объектов кластеризуемого множества, чтобы далее использовать эти расстояния в вычислениях на фазах Build и Swap. Предвычисления в данном случае могут быть реализованы в виде распараллеливаемых и векторизуемых компилятором циклов, что повышает общую производительность алгоритма (за счет избыточного пространства оперативной памяти, необходимого для хранения расстояний). Вычисление расстояний, равно как и каждая из фаз Build и Swap, распараллеливается с помощью технологии OpenMP [18].

Множество кластеризуемых объектов X представляется в виде матрицы $\mathbf{X} \in \mathbb{R}^{n\times d}$, множество медоидов C — в виде массива $\mathbf{C} \in \mathbb{N}^k$, содержащего индексы медоидов в множестве X. Предварительно вычисляемые расстояния между объектами множества X сохраняются в *матрице расстояний* $\mathbf{M} \in \mathbb{R}^{n\times n}$, где

Алгоритм 2. δ_{jih}(in x_j, c_i, x_h, d_j, s_j; out δ)

```
1:  if ρ(x_j, c_i) > d_j and ρ(x_j, x_h) > d_j  then
2:      δ ← 0
3:  else if ρ(x_j, c_i) = d_j then
4:      if ρ(x_j, x_h) < s_j then
5:          δ ← ρ(x_j, x_h) − d_j
6:      else
7:          δ ← s_j − d_j
8:  else if ρ(x_j, x_h) < d_j then
9:      δ ← ρ(x_j, x_h) − d_j
10: return δ
```

Алгоритм 3. PhiPAM(in $\mathbf{X}$, k, $block$, L; out $\mathbf{C}$)

```
                                          ▷ Вычисление матрицы расстояний
1:  X_ASA ← Permute(X, block)
2:  M ← PhiBlockwise(X, X_ASA, block)
                                          ▷ Фаза Build
3:  C ← PhiBuild(M, k, L)
                                          ▷ Фаза Swap
4:  repeat
5:      {T_min, c_min, x_min} ← PhiSwap(C, M, L)
6:      C_{c_min} ← x_min
7:  until T_min < 0
8:  return C
```

Алгоритм 4. PhiBlockwise(in $\mathbf{X}$, $\mathbf{X}_{ASA}$, $block$; out $\mathbf{M}$)

```
1:  #pragma omp parallel for
2:  for i from 1 to n
3:      for j from 1 to ⌈n / block⌉ do
4:          sum ← 0̄
5:          for k from 1 to d do
6:              for ℓ from 1 to block  do
7:                  sum_ℓ ← sum_ℓ + (X(i,k) − X_ASA(j+k, ℓ))^2
8:              for k from 1 to block do
9:                  M(i, j·block + k) ← sum_k
10: return M
```

$\mathbf{M}(i,j) = \rho^2\big(\mathbf{X}(i,\cdot), \mathbf{X}(j,\cdot)\big)$. Переменные $block$ и L являются параметрами распараллеливания вычисления матрицы расстояний и фаз Build и Swap соответственно и рассматриваются ниже.

3.1. Распараллеливание вычисления матрицы расстояний. Для вычисления матрицы расстояний используется алгоритм *PhiBlockwise* [3]. Данный алгоритм (см. алгоритм 4) предполагает использование вспомогательной матрицы $\mathbf{X}_{ASA} \in \mathbb{R}^{d\cdot\left\lceil\frac{n}{block}\right\rceil\times block}$, которая представляет собой матрицу $\mathbf{X}$, размещенную в памяти в соответствии со схемой *ASA (Array of Structures of Arrays)* [11].

Штатным способом компоновки матрицы в памяти является схема *AoS (Array of Structures)* (рис. 1а), предполагающая хранение матрицы в виде массива, элементами которого являются структуры (значения комбинированного типа данных). Однако схема вычисления евклидовых расстояний при использовании компоновки AoS приводит к высокому количеству кэш-промахов ввиду низкой локальности обрабаты-

```
1  type AoS_element = record
2  float x ;
3  float y ;
4  float z ;
5  end ;
```

```
1  type ASA_element = record
2  x : array [1 .. block] of float ;
3  y : array [1 .. block] of float ;
4  z : array [1 .. block] of float ;
5  end ;
```

var X_{AoS} : **array** [1 .. n] of AoS_element ; **var** X_{ASA} : **array** $\left[1 .. \left\lceil \frac{n}{block} \right\rceil\right]$ of ASA_element ;

a) b)

Рис. 1. Способы компоновки матрицы $\mathbf{X}$ в памяти (для d=3): a) массив из структур, b) массив структур из массивов

ваемых в памяти данных. В отличие от схемы AoS, усложненная схема ASA (см. рис. 1b) предполагает хранение матрицы в виде массива из $\left\lceil \frac{n}{block} \right\rceil$ элементов, где элемент состоит из d массивов, каждый из которых содержит $block$ вещественных чисел. Параметр $block$ подбирается с учетом двух следующих условий. Мощность множества кластеризуемых объектов X и n должна быть кратна $block$. Если это не так, то необходимо дополнить матрицу $\mathbf{X}$ фиктивными нулевыми строками (соответствующие объекты впоследствии исключаются из результатов кластеризации). Кроме того, параметр $block$ должен быть кратен значению ширины векторного регистра ускорителя Intel MIC. *Шириной векторного регистра* называют максимальное количество элементов данных, которые одномоментно можно поместить в регистр. Для ускорителя Intel MIC размер векторного регистра составляет 512 бит, что позволяет разместить в нем 16 вещественных чисел с одинарной точностью. В экспериментах с PhiBlockwise на различных наборах данных наилучшие результаты производительности алгоритма были достигнуты при значении параметра $block$=512 [22], которое используется в настоящем исследовании. Усложненная компоновка данных матрицы $\mathbf{X}$ способствует уменьшению кэш-промахов процессора и, соответственно, повышает производительность алгоритма PhiBlockwise. Для получения матрицы $\mathbf{X}_{ASA}$ используется параллельный алгоритм $Permute$ (см. алгоритм 5).

Алгоритм 5. Permute(in $\mathbf{X}_{SoA}$, $block$; out $\mathbf{X}_{ASA}$)

```
1: #pragma omp parallel for
2: for j from 1 to ⌈n/block⌉ do
3:     for i from 1 to d do
4:         for k from 1 to block do
5:             X_ASA(j·d + i, k) ← X_SoA(j·block + k, i)
6: return X_ASA
```

3.2. Распараллеливание фаз Build и Swap. В параллельной реализации фаз алгоритма Build и Swap используется техника *майлинга (tiling)* [4]. Тайлинг предполагает разбиение множества итераций счетчика исходного цикла на непересекающиеся подмножества меньшей мощности. Это приводит к разбиению исходного обрабатываемого массива на блоки *(тайлы)* меньшего размера, которые могут быть помещены в кэш-память процессора. Хранение данных в кэш-памяти для их неоднократного использования в процессе обработки тайла снижает количество кэш-промахов и увеличивает общую эффективность алгоритма. Указанная техника также дает компилятору больше информации о контексте исполнения и позволяет выполнить векторизацию вычислений в теле цикла или развертку цикла [4]. Размер тайла является параметром алгоритма, обозначим его как L. В реализации используется рекомендуемый в руководстве системного программиста Intel MIC [11] размер тайла $L = 32$. Для компактной записи циклов алгоритма, к которым применена техника тайлинга, введем сокращенную запись оператора цикла, представленную на рис. 2.

Параллельная реализация фазы Build представлена в алгоритме 6. Множества D и S представляются

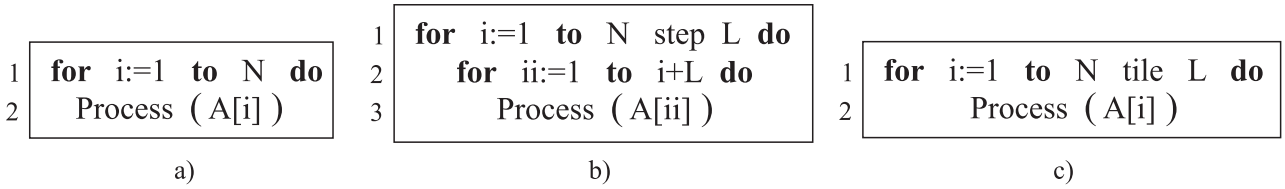


Рис. 2. Тайлинг цикла: a) исходный цикл, b) цикл с тайлингом, c) краткая запись цикла с тайлингом

как одномерные массивы $\mathbf{D}, \mathbf{S} \in \mathbb{R}^n$ соответственно. Алгоритм действует следующим образом. Массив $\mathbf{D}$ расстояний от точек до ближайшего медоида инициализируется значением ∞. Далее выполняется цикл по всем кластерам, в котором осуществляется поиск медоида, минимизирующего значение целевой функции.

Алгоритм 6. PhiBuild(in $\mathbf{M}$, k, L; out $\mathbf{C}$)

```
1:  D ← ∞̄
2:  for ℓ from 1 to k do
3:      min ← ∞
4:      #pragma omp parallel for reduction(func_min(sum) : {min, C_ℓ})
5:      for i from 1 to n tile L do
6:          sum ← 0
7:          for j from 1 to n tile L do
8:              if D_j is ∞ then
9:                  sum ← sum + M(i, j)
10:             else if D_j > M(i, j) then
11:                 sum ← sum + M(i, j) − D_j
12:         if sum < min then
13:             min ← sum; C_ℓ ← i
14:     #pragma omp parallel for
15:     for i from 1 to n do
16:         D_i ← min(D_i, M(C_ℓ, i))
17: return C
```

Тело указанного цикла организовано следующим образом. Выполняется сканирование всех кластеризуемых объектов, и для каждого объекта вычисляется значение целевой функции относительно данного объекта и ранее найденных медоидов. Объект, на котором достигнут минимум целевой функции, добавляется в множество медоидов. Данный цикл распараллеливается с помощью стандартной директивы компилятора OpenMP `#pragma omp parallel for`. Конструкция `reduction` этой директивы обеспечивает *свертку* операции поиска минимума функции: каждая нить выполняет поиск локального минимума в назначенном ей диапазоне обработки массива, затем находится глобальный минимум как наименьшее значение среди локальных минимумов. В качестве результата свертки выдается глобальный минимум и индекс медоида, на котором найденное значение достигнуто. Сканирование кластеризуемых объектов осуществляется с помощью описанной выше техники тайлинга. Этот же прием применяется во вложенном цикле, где выполняется вычисление значений целевой функции. В качестве завершающей тело цикла операции алгоритм обновляет значения вектора расстояний от каждой точки до ближайшего к ней медоида. Алгоритм возвращает начальную расстановку медоидов, используемую далее в фазе Swap.

Параллельная реализация фазы Swap представлена в алгоритме 7. Алгоритм действует следующим образом. Выполняется сканирование кластеризуемых объектов, и выполняются следующие действия: замена каждого медоида на кластеризуемый объект и вычисление соответствующего изменения значения целевой функции. При этом осуществляется поиск минимального значения целевой функции и индексов медоида и кластеризуемого объекта, на которых найденный минимум достигается. Цикл, выполняющий сканирование, распараллеливается с помощью стандартной директивы компилятора OpenMP `pragma omp parallel for` с использованием конструкции `reduction`, рассмотренной выше.

Алгоритм 7. PhiSwap(in $\mathbf{C}$, $\mathbf{M}$, L; out $T_{\min}$, $c_{\min}$, $x_{\min}$)

```
1:  D ← ∞; S ← ∞
2:  #pragma omp parallel for
3:  for h from 1 to n tile L do
4:      for i from 1 to k do
5:          if D_h > M(h, C_i) then
6:              S_h ← D_h
7:              D_h ← M(h, C_i)
8:          else if S_h > M(h, C_i) then
9:              S_h ← M(h, C_i)
10: #pragma omp parallel for reduction(func_{min(T)} : {T_min, c_min, o_min})
11: for h from 1 to n tile L do
12:     for i from 1 to k do
13:         T ← 0
14:         for ℓ from 1 to n tile L do
15:             T ← T + Delta(M, ℓ, C_i, h, D_ℓ, S_ℓ)
16:         if T < T_min then
17:             T_min ← T; c_min ← i; x_min ← h
18: return {T_min, c_min, x_min}
```

Таблица 1

Аппаратная платформа экспериментов

Характеристика	Процессор Intel Xeon		Ускоритель Intel MIC
Модель	E5-2630v4 (Host)	E5-2697v4 (Broadwell)	Xeon Phi 7290 (KNL)
Количество физ. ядер	2×10	2×16	72
Гиперпоточность	2×	2×	4×
Количество лог. ядер	40	64	288
Частота, ГГц	2.2	2.6	1.5
Размер вект. регистра, бит	256	256	512
Пик. произв-ть, TFLOPS	0.390	0.600	3.456

Таблица 2

Наборы данных для экспериментов

Набор	d	n	k_{elbow}	Семантика
Power	3	35×2^{10}	4	Информация о домашнем энергопотреблении [16]
FCS Human	423	18×2^{10}	10	Агрегированная информация о геноме человека [8]
Haiti	3	45×2^{10}	4	Спутниковое фото высокого разрешения (землетрясение в Гаити 2010 г.) [14]

4. Вычислительные эксперименты. В рамках исследования предложенного алгоритма PhiPAM были проведены вычислительные эксперименты, в которых измерялись быстродействие, ускорение и параллельная эффективность алгоритма. Под быстродействием подразумевается время работы алгоритма без учета времени загрузки данных в память и вывода результатов. На основе полученных значений вычислялись ускорение и параллельная эффективность, определяемые следующим образом [1]. *Ускорение* и *параллельная эффективность* параллельного алгоритма, запускаемого на p нитях, вычисляется как $s(p) = \dfrac{t_1}{t_p}$ и $e(p) = \dfrac{s(p)}{p}$ соответственно, где t_1 и t_p — время работы алгоритма на одной и p нитях соответственно. Кроме того, было произведено сравнение быстродействия алгоритмов PhiPAM и GPUPAM [14].

Эксперименты были проведены на многоядерных системах Intel, установленных в Сибирском суперкомпьютерном центре ИВМиМГ СО РАН [2], технические характеристики которых резюмированы в табл. 1. В экспериментах использовались наборы данных, резюмированные в табл. 2. Количество кластеров k_{elbow}, близкое к оптимальному для набора данных, было предварительно найдено для каждого набора с помощью метода локтя [25].

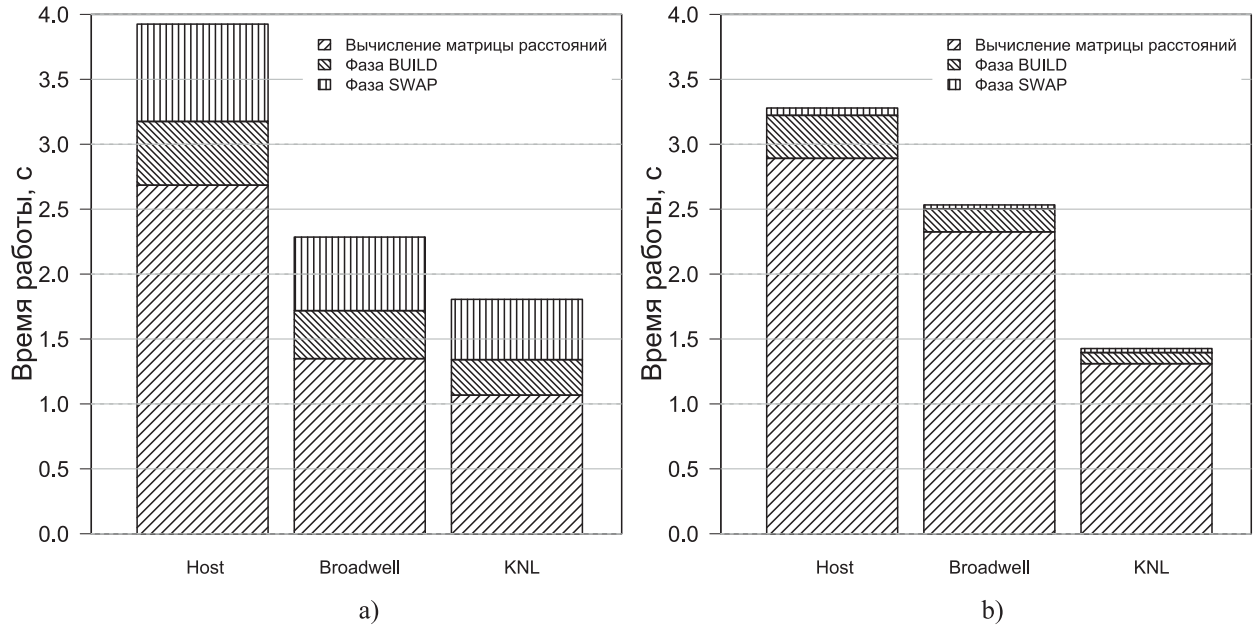


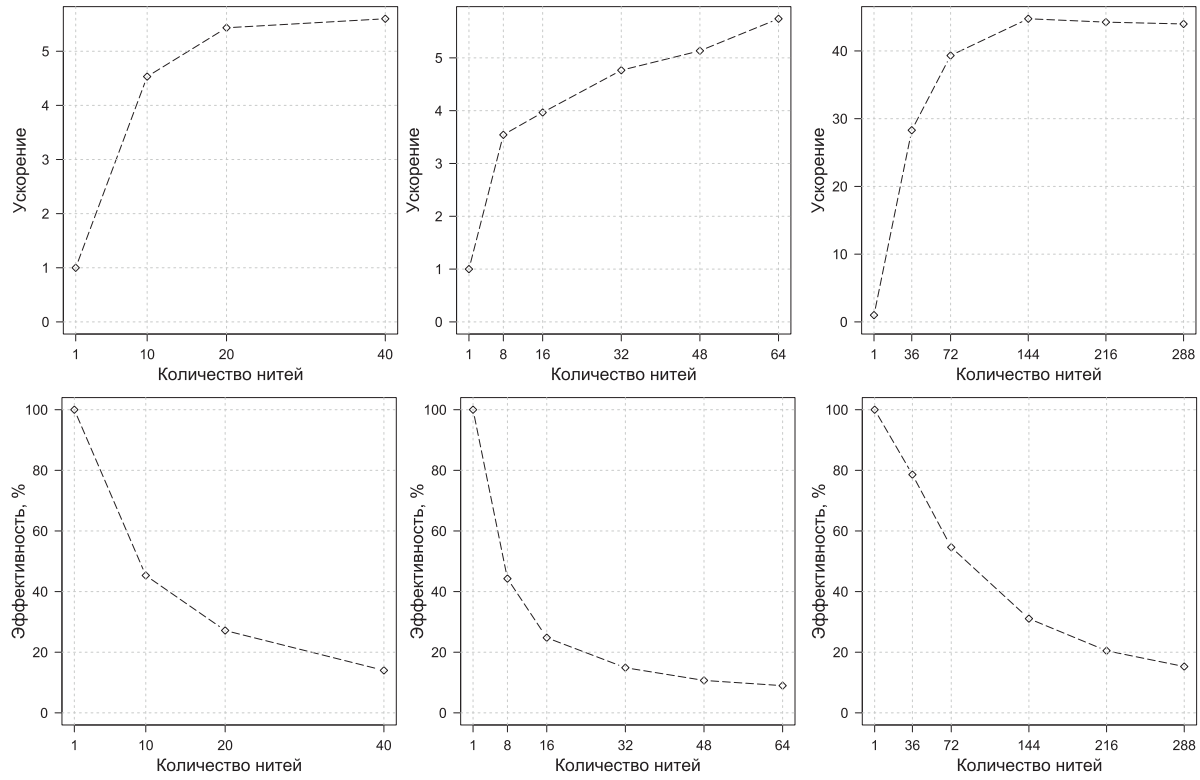
Рис. 3. Быстродействие PhiPAM: a) набор Power, b) набор FCS Human

Таблица 3
Сравнение быстродействия алгоритмов GPUPAM и PhiPAM

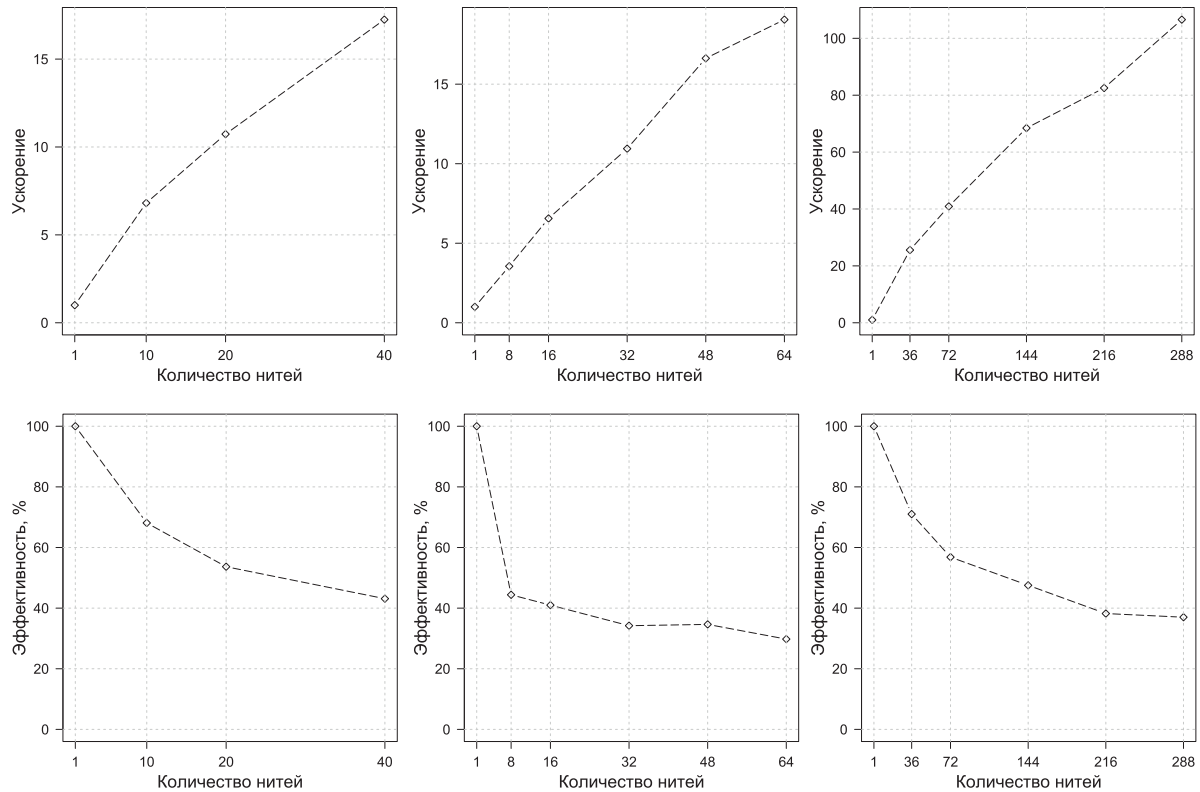
Алгоритм	GPUPAM	PhiPAM
Платформа	NVIDIA GeForce410	Intel Xeon Phi 7290
Кол-во физ. ядер	48	2
Частота, ГГц	1.15	1.5
Пик. произв-ть, GFLOPS	73	96
Время работы, с	197	91

На рис. 3 представлено быстродействие алгоритма PhiPAM на различных платформах. Можно видеть, что алгоритм показывает наибольшее быстродействие на платформе многоядерного ускорителя, опережая двухпроцессорные системы Intel. Ускорение и параллельная эффективность алгоритма представлены на рис. 4. Можно видеть, что ускоритель Intel MIC является лучшей среди рассмотренных платформой для масштабирования алгоритма: при количестве нитей, совпадающем с количеством физических ядер системы, алгоритм обеспечивает ускорение 40 раз и параллельную эффективность 60%. В табл. 3 представлены результаты сравнения быстродействия алгоритмов PhiPAM и GPUPAM на основе набора данных и результатов экспериментов из работы [14]. PhiPAM запускается на двух ядрах ускорителя Intel MIC, обеспечивающих пиковую производительность ускорителя, относительно близкую к производительности той платформы, на которой запускался алгоритм GPUPAM. Результаты показывают, что алгоритм PhiPAM, использующий предвычисление матрицы расстояний и технику тайлинга циклов в реализации фаз Build и Swap, в итоге опережает конкурента.

5. Заключение. В статье рассмотрена задача ускорения алгоритма кластеризации PAM, использующего технику медоидов, на многоядерных вычислительных системах архитектуры Intel Many Integrated Core (MIC). Кластеризация на основе техники медоидов направлена на повышение устойчивости алго-



a)



b)

Рис. 4. Ускорение и эффективность PhiPAM (слева направо платформы: Host, Broadwell, KNL): a) набор Power, b) набор FCS Human

ритма к шумам и выбросам в исходных данных и применяется в широком спектре приложений: сегментирование медицинских и спутниковых изображений, анализ ДНК-микрочипов и текстов и др. Ускорители Intel MIC обеспечивают большое количество энергоэффективных вычислительных ядер, поддерживающих векторную обработку данных, и являются конкурентоспособной альтернативой более распространенным системам GPU и FPGA.

Предложен новый параллельный алгоритм PhiPAM, использующий технологию OpenMP. Алгоритм предполагает предварительное вычисление матрицы расстояний между кластеризуемыми объектами, в котором используется специализированная компоновка данных в памяти, позволяющая сократить количество кэш-промахов и векторизовать вычисления. Параллельная реализация стандартных фаз алгоритма PAM использует технику тайлинга циклов, которая позволяет эффективно векторизовать вычисления на ускорителях MIC. Эксперименты, проведенные на реальных наборах данных, показали хорошую масштабируемость алгоритма.

Работа выполнена при финансовой поддержке РФФИ (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). В работе использованы ресурсы ЦКП Сибирского суперкомпьютерного центра ИВМиМГ СО РАН.

СПИСОК ЛИТЕРАТУРЫ

1. *Воеводин В.В., Воеводин Вл.В.* Параллельные вычисления. СПб.: БХВ-Петербург, 2002.
2. Перечень оборудования Центра коллективного пользования Сибирского суперкомпьютерного центра ИВМиМГ СО РАН. http://www.sscc.icmmg.nsc.ru/hardware.html.
3. *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. **7**, №. 3. 65–82.
4. *Bacon D.F., Graham S.L., Sharp O.J.* Compiler transformations for high-performance computing // ACM Computing Surveys. 1994. **26**, N 4. 345–420.
5. *Chrysos G.* Intel Xeon Phi coprocessor (codename Knights Corner) // Proc. of the 2012 IEEE Hot Chips 24th Symposium (HCS). New York: IEEE Press, 2012. doi 10.1109/HOTCHIPS.2012.7476487.
6. *Duran A., Klemm M.* The Intel many integrated core architecture // Proc. of the 2012 Int. Conf. on High Performance Computing and Simulation. New York: IEEE Press, 2012. 365–366.
7. *El-Alfy E.-S.M.* Detection of phishing websites based on probabilistic neural networks and K-Medoids clustering // The Computer Journal. 2017. **60**, N 12. 1745–1759.
8. *Engreitz J.M., Daigle B.J., Marshall J.J., Altman R.B.* Independent component analysis: mining microarray data for fundamental human gene expression modules // Journal of Biomedical Informatics. 2010. **43**, N 6. 932–944.
9. *Espenshade J., Pangborn A., von Laszewski G., et al.* Accelerating partitional algorithms for flow cytometry on GPUs // Proc. of the IEEE Int. Symp. on Parallel and Distributed Processing with Applications. New York: IEEE Press, 226–233. 2009.
10. *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. **103**. 21–28.
11. *Jeffers J., Reinders J.* Intel Xeon Phi coprocessor high performance programming. Boston: Morgan Kaufmann, 2013.
12. *Kaufman L., Rousseeuw P.J.* Finding groups in data: an introduction to cluster analysis. New York: Wiley, 1990.
13. *Kohlhoff K.J., Sosnick M.H., Hsu W.T., et al.* CAMPAIGN: an open-source library of GPU-accelerated data clustering algorithms // Bioinformatics. 2011. **27**, N 16. 2321–2322.
14. *Kurte K.R., Durbha S.S.* High resolution disaster data clustering using Graphics Processing Units // Proc. of the 2013 IEEE Int. Geoscience and Remote Sensing Symposium. New York: IEEE Press, 2013. 1696–1699.
15. *Lee S., Liao W.-K., Agrawal A., et al.* Evaluation of K-means data clustering algorithm on Intel Xeon Phi // Proc. of the 2016 IEEE Int. Conf. on Big Data. New York: IEEE Press, 2016. 2251–2260.
16. *Bache K., Lichman M.* Individual household electric power consumption dataset. Irvine: University of California, 2013.
17. *Lloyd S.P.* Least squares quantization in PCM // IEEE Transactions on Information Theory. 1982. **28**, N 2. 129–136.
18. *Mattson T.* Introduction to OpenMP // Proc. of the 2006 ACM/IEEE Conf. on Supercomputing. New York: ACM Press, 2006. doi 10.1145/1188455.1188673.
19. *Mohammed N.N., Abdulazeez A.M.* Evaluation of partitioning around medoids algorithm with various distances on microarray data // Proc. of the 2017 IEEE Int. Conf. on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData). New York: IEEE Press, 2017. 1011–1016.
20. *Mushtaq H., Khawaja S.G., Akram M.U., et al.* A parallel architecture for the Partitioning Around Medoids (PAM) algorithm for scalable multi-core processor implementation with applications in healthcare // Sensors. 2018. **18**, N 12.

doi 10.3390/s18124129.
21. *Nguyen P.T., Eckert K., Ragone A., Noia T.D.* Modification to K-Medoids and CLARA for effective document clustering // Lecture Notes in Computer Science. Vol. 10352. Cham: Springer, 2017. 481–491.
22. *Rechkalov T., Zymbler M.* Accelerating medoids-based clustering with the Intel Many Integrated Core architecture // Proc. of the 9th Int. Conf. on Application of Information and Communication Technologies. New York: IEEE Press, 2015. 413–417.
23. *Sodani A.* Knights Landing (KNL): 2nd generation Intel Xeon Phi processor // Proc. of the 2015 IEEE Hot Chips 27th Symposium. New York: IEEE Press, 2015. doi 10.1109/HOTCHIPS.2015.7477467.
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. *Thorndike R.L.* Who belongs in the family? // Psychometrika. 1953. **18**, N 4. 267–276.
26. *Wu F., Wu Q., Tan Y., et al.* A vectorized K-Means algorithm for Intel Many Integrated Core architecture // Lecture Notes in Computer Science. Vol. 8299. Heidelberg: Springer, 2013. 277–294.

Поступила в редакцию
26.02.2019

A Parallel Data Clustering Algorithm for Intel MIC Accelerators

T. V. Rechkalov [1] **and M. L. Zymbler** [2]

[1] *South Ural State University, School of Electronic Engineering and Computer Science; prospekt Lenina 76, Chelyabinsk, 454080, Russia; Graduate Student, e-mail: trechkalov@yandex.ru*

[2] *South Ural State University, School of Electronic Engineering and Computer Science; prospekt Lenina 76, Chelyabinsk, 454080, Russia; Ph.D., Associate Professor, Head of Department, e-mail: mzym@susu.ru*

Received February 26, 2019

Abstract: The PAM (Partitioning Around Medoids) is a partitioning clustering algorithm where each cluster is represented by an object from the input dataset (called a medoid). The medoid-based clustering is used in a wide range of applications: the segmentation of medical and satellite images, the analysis of DNA microarrays and texts, etc. Currently, there are parallel implementations of PAM for GPU and FPGA systems, but not for Intel Many Integrated Core (MIC) accelerators. In this paper, we propose a novel parallel PhiPAM clustering algorithm for Intel MIC systems. Computations are parallelized by the OpenMP technology. The algorithm exploits a sophisticated memory data layout and loop tiling technique, which allows one to efficiently vectorize computations with Intel MIC. Experiments performed on real data sets show a good scalability of the algorithm.

Keywords: clustering, medoid, parallel algorithm, OpenMP, Intel Xeon Phi, data layout, vectorization of computations.

References

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

2. Hardware Specifications of the Siberian Supercomputing Center.
http://www.sscc.icmmg.nsc.ru/hardware.html. Cited April 5, 2019.

3. T. V. Rechkalov and M. L. Zymbler, "A Parallel Algorithm of Euclidean Distance Matrix Computation for the Intel Xeon Phi Knights Landing Many-Core Processor," Vestn. Yuzhn. Ural. Gos. Univ. Ser. Vychisl. Mat. Inf. **7** (3), 65–82 (2018).

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

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. A. Duran and M. Klemm, "The Intel Many Integrated Core Architecture," in *Proc. 2012 Int. Conf. on High Performance Computing and Simulation, Madrid, Spain, July 2–6, 2012* (IEEE Press, New York, 2012), pp. 365–366.

7. E.-S.M. El-Alfy, "Detection of Phishing Websites Based on Probabilistic Neural Networks and K-Medoids Clustering," Comput. J. **60** (12), 1745–1759 (2017).

8. J. M. Engreitz, B. J. Daigle, J. J. Marshall, and R. B. Altman, "Independent Component Analysis: Mining Microarray Data for Fundamental Human Gene Expression Modules," J. Biomed. Inform. **43** (6), 932–944 (2010).

9. J. Espenshade, A. Pangborn, G. von Laszewski, et al., "Accelerating Partitional Algorithms for Flow Cytometry on GPUs," in *Proc. IEEE Int. Symp. on Parallel and Distributed Processing with Applications, Chengdu, Sichuan, China, August 10–12, 2009* (IEEE Press, New York, 2009), pp. 226–233.

10. M. Jaros, P. Strakos, T. Karásek, et al., "Implementation of K-means Segmentation Algorithm on Intel Xeon Phi and GPU: Application in Medical Imaging," Adv. Eng. Software **103**, 21–28 (2017).

11. J. Jeffers and J. Reinders, *Intel Xeon Phi Coprocessor High Performance Programming* (Morgan Kaufmann, Boston, 2013).

12. L. Kaufman and P. J. Rousseeuw, *Finding Groups in Data: An introduction to Cluster Analysis* (Wiley, New York, 1990).

13. K. J. Kohlhoff, M. H. Sosnick, W. T. Hsu, et al., "CAMPAIGN: An Open-Source Library of GPU-Accelerated Data Clustering Algorithms," Bioinformatics **27** (16), 2321–2322 (2011).

14. K. R. Kurte and S. S. Durbha, "High Resolution Disaster Data Clustering Using Graphics Processing Units," in *Proc. 2013 IEEE Int. Geoscience and Remote Sensing Symposium, Melbourne, Australia, July 21–26, 2013* (IEEE Press, New York, 2013), pp. 1696–1699.

15. S. Lee, W.-K. Liao, A. Agrawal, et al., "Evaluation of K-Means Data Clustering Algorithm on Intel Xeon Phi," in *Proc. 2016 IEEE Int. Conf. on Big Data, Washington DC, USA, December 5–8, 2016* (IEEE Press, New York, 2016), pp. 2251–2260.

16. K. Bache and M. Lichman, *Individual Household Electric Power Consumption Dataset* (Univ. of California, Irvine, 2013).

17. S. P. Lloyd, "Least Squares Quantization in PCM," IEEE Trans. Inform. Theory **28** (2), 129–136 (1982).

18. 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

19. N. N. Mohammed and A. M. Abdulazeez, "Evaluation of Partitioning Around Medoids Algorithm with Various Distances on Microarray Data," in *Proc. of the 2017 IEEE Int. Conf. on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData), Exeter, United Kingdom, June 21–23, 2017* (IEEE Press, New York, 2017), pp. 1011–1016.

20. H. Mushtaq, S. G. Khawaja, M. U. Akram, et al., "A Parallel Architecture for the Partitioning Around Medoids (PAM) Algorithm for Scalable Multi-Core Processor Implementation with Applications in Healthcare," Sensors **18** (2018). doi 10.3390/s18124129.

21. P. T. Nguyen, K. Eckert, A. Ragone, and T. D. Noia, "Modification to K-Medoids and CLARA for Effective Document Clustering," in *Lecture Notes in Computer Science* (Springer, Cham, 2017), Vol. 10352, pp. 481–491.

22. T. Rechkalov and M. Zymbler, "Accelerating Medoids-Based Clustering with the Intel Many Integrated Core Architecture," in *Proc. 9th Int. Conf. on Application of Information and Communication Technologies, Rostov-on-Don, Russia, October 14–16, 2015* (IEEE Press, New York, 2015), pp. 413–417.

23. A. Sodani, "Knights Landing (KNL): 2nd Generation Intel Xeon Phi Processor," in *Proc. 2015 IEEE Hot Chips 27th Symposium (HCS), Cupertino, USA, August 22–25, 2015* (IEEE Press, New York, 2015), doi 10.1109/HOTCHIPS.2015.7477467

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. R. L. Thorndike, "Who Belongs in the Family?," Psychometrika **18** (4), 267–276 (1953).

26. F. Wu, Q. Wu, Y. Tan, et al., "A Vectorized K-Means Algorithm for Intel Many Integrated Core Architecture," in *Lecture Notes in Computer Science* (Springer, Heidelberg, 2013), Vol. 8299, pp. 277–294.