Параллельный алгоритм кластеризации для многоядерного сопроцессора Intel Xeon Phi*

Т.В. Речкалов

Южно-Уральский государственный университет

В работе описана параллельная версия алгоритма кластеризации Partitioning Around Medoids для сопроцессора Intel Xeon Phi. Распараллеливание выполнено на основе OpenMP. Циклические операции преобразованы для выполнения векторизации. Алгоритм использует матрицу предвычисленных расстояний, которая хранится в памяти сопроцессора. Представлены результаты вычислительных экспериментов, подтверждающие эффективность разработанного алгоритма.

1. Введение

Кластеризация (clustering) представляет собой одну из базовых задач интеллектуального анализа данных и заключается в разбиении заданного конечного множества объектов на непересекающиеся подмножества (*кластеры*) таким образом, чтобы каждый кластер состоял из схожих объектов, а объекты разных кластеров существенно различались; при этом семантика кластеров заранее не известна [4].

Семейство разделительных (partitioning) алгоритмов кластеризации предполагает начальное разбиение исходного множества объектов мощности n на k кластеров ($k \leq n$), где кластеры совокупно удовлетворяют следующим условиям: (1) в каждом кластере имеется, по крайней мере, один объект и (2) каждый объект принадлежит в точности одному кластеру. После выполнения начального разбиения разделительный алгоритм итеративно осуществляет перемещения объектов между кластерами с целью улучшить начальное разбиение (чтобы объекты из одного кластера были более «близкими», а из разных кластеров — более «далекими» друг другу). Принадлежность объекта кластеру определяется на основе вычисления расстояния от данного объекта до центра данного кластера; по окончании итерации алгоритм перевычисляет центры кластеров.

Алгоритм *PAM (Partitioning Around Medoids)* [13] представляет собой один из классических разделительных алгоритмов кластеризации, в котором в качестве центров кластеров могут выбираться только кластеризуемые объекты (называемые *медоидами*). Алгоритм PAM применяется в широком спектре предметных областей: анализ текстов [14], биоинформатика [1], интеллектуальные транспортные системы [17] и др. Вычислительная сложность одной итерации алгоритма составляет $O(k(n-k)^2)$, что дает неприемлемо низкую эффективность при больших значениях n и k. Имеющиеся попытки увеличения эффективности PAM связаны с использованием графических ускорителей [3, 9] и оставляют без внимания многоядерные ускорители на базе архитектуры Intel Many Integrated Core [7].

В данной статье предлагается параллельный алгоритм кластеризации PAM для многоядерного сопроцессора Intel Xeon Phi. Статья организована следующим образом. Раздел 2 содержит обзор работ и краткий обзор архитектуры и модели программирования многоядерного сопроцессора Intel Xeon Phi. В разделе 3 вводятся базовые определения и приводится обзор алгоритма PAM. Раздел 4 описывает распараллеленную реализацию алгоритма PAM. Вычислительные эксперименты представлены в разделе 5. В разделе 6 суммируются полученные результаты и указываются направления будущих исследований.

*Исследование выполнено при поддержке программы «Участник молодежного научно-инновационного конкурса» («УМНИК»).

2. Контекст исследования и обзор работ

2.1. Обзор работ

Существует большое количество работ по кластерному анализу. Классические алгоритмы кластеризации k-Means и k-Medoids были описаны в [5, 10]. Оригинальный алгоритм PAM был предложен в [8].

Следующие работы посвящены ускорению алгоритмов кластеризации с помощью параллельных аппаратных средств. В статье [6] сравниваются реализации k-Means для FPGA и GPU. Авторы [15] описывают улучшения алгоритма k-Means для уменьшения передачи данных между CPU и GPU. В [16] предложена техника для улучшения распределения данных по потокам GPU в алгоритме k-Means. Реализация алгоритма k-Means для фреймворка Hadoop с применением графических ускорителей описана в [18]. В работе [3] описана реализация нескольких алгоритмов кластеризации для GPU, включая k-Medoids. Фреймворк для кластеризации генетических данных на GPU с помощью алгоритма k-Medoids описан в [9].

По нашему мнению потенциал ускорителей на архитектуре Intel MIC для решения задач кластеризации недооценен. Насколько нам известно существует только одна работа [12], посвященная адаптации алгоритма плотностной кластеризации DBSCAN для архитектуры Intel MIC. В данной работе описана техника ускорения алгоритма кластеризации Partitioning Around Medoids с помощью многоядерного сопроцессора Intel Xeon Phi.

2.2. Архитектура и модель программирования сопроцессора Intel Xeon Phi

Многоядерный сопроцессор Intel Xeon Phi состоит из 61 ядра на базе архитектуры x86, соединенных высокоскоростной двунаправленной шиной, где каждое ядро поддерживает 4× гипертрединг и содержит 512-битный векторный процессор. Каждое ядро имеет собственный кэш 1 и 2 уровня, при этом обеспечивается когерентность кэшей всех ядер. Сопроцессор соединяется с хост-компьютером посредством интерфейса PCI Express. Поскольку сопроцессор Intel Xeon Phi основан на архитектуре Intel x86, он поддерживает те же программные инструменты и модели программирования, что и ординарный процессор Intel Xeon.

Сопроцессор поддерживает следующие режимы запуска приложений: *native*, *offload* и *symmetric*. В режиме *native* приложение выполняется независимо, исключительно на сопроцессоре. В режиме *offload* приложение запускается на процессоре и выгружает вычислительно интенсивную часть работы (код и данные) на сопроцессор. Режим *symmetric* позволяет сопроцессору и процессору взаимодействовать в рамках модели обмена сообщениями (Message Passing Interface).

3. Описание алгоритма Partitioning Around Medoids

Введем следующие обозначения для формального описания алгоритма PAM [8]. Пусть $O = \{o_1, o_2, \ldots, o_n\}$ — это множество кластеризуемых объектов, где каждый объект — это кортеж, состоящий из p вещественных чисел. Пусть k количество кластеров, $k \ll n$, $C = \{c_1, c_2, \ldots, c_k\}$ множество медоидов, $C \subset O$, и $\rho : O \times C \to R$ — это метрика расстояния.

Алгоритм PAM является разновидностью метода наискорейшего подъема. На каждой итерации выбирается пара медоид c_i и не-медоид o_j такая, что замена медоида на не-медоид дает лучшую кластеризацию из возможных. Оценка кластеризации выполняется с помощью целевой функции, вычисляемой как сумма расстояний от каждого объекта до ближайшего медоида:

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

Вход : Множество объектов O, количество кластеров k
Выход: Множество кластеров C
1 Инициализировать C ; `// фаза BUILD`
2 **repeat**`// фаза SWAP`
3 | Вычислить T_{min} ;
4 | Поменять местами c_{min} и o_{min};
5 **until** $T_{min} < 0$;

Рис. 1. Псевдокод алгоритма PAM

Псевдокод алгоритма PAM представлен на рис. 1. PAM состоит из двух фаз: BUILD и SWAP. В фазе BUILD выполняется первичная кластеризация, в которой последовательно выбирается k объектов в качестве медоидов. Первый объект c_1 — это объект, сумма расстояний от которого до всех остальных объектов является наименьшей:

$$c_1 = \arg\min_{1 \le h \le n} \sum_{j=1}^{n} \rho(o_h, o_j). \quad (2)$$

Затем выбирается следующий объект, минимизирующий целевую функцию. Для этого производится вычисление целевой функции относительно ранее выбранных объектов c и каждого из невыбранных объектов o:

$$c_2 = \arg\min_{1 \le h \le n} \sum_{j=1}^{n} \min(\rho(c_1, o_j), \rho(o_h, o_j)), \quad (3)$$

$$c_3 = \arg\min_{1 \le h \le n} \sum_{j=1}^{n} \min(\min_{1 \le l \le 2}(\rho(c_l, o_j)), \rho(o_h, o_j)), \quad (4)$$

...

$$c_k = \arg\min_{1 \le h \le n} \sum_{j=1}^{n} \min(\min_{1 \le l \le k-1}(\rho(c_l, o_j)), \rho(o_h, o_j)). \quad (5)$$

Эта процедура повторяется, пока не будет выбрано k объектов.

В фазе SWAP алгоритм PAM пытается улучшить множество медоидов C. Алгоритм выполняет поиск пары объектов (c_{min}, o_{min}), минимизирующих целевую функцию. Для этого перебираются все пары объектов (c_i, o_h), где c_i — это медоид, а o_h не-медоид. Вычисляется изменение целевой функции при исключении c_i из множества медоидов и включении o_h вместо него. Обозначим это изменение как T_{ih}, а минимальное значение T_{min} достигается на паре (c_{min}, o_{min}). Если $T_{min} > 0$, тогда множество C не может быть улучшено, и алгоритм завершается.

Для описания вычисления T_{ih} введем следующие обозначения. Пусть $D = \{d_1, d_2, \ldots, d_n\}$ — это множество расстояний от каждого объекта до ближайшего медоида. Пусть $S = \{s_1, s_2, \ldots, s_n\}$ — это множество расстояний от каждого объекта до второго ближайшего медоида. Пусть C_{jih} — это вклад не-медоида o_j в T_{ih} при замене c_i на o_h. В этом случае T_{ih} определяется как сумма C_{jih}:

$$T_{ih} = \sum_{j=1}^{n} C_{jih}. \quad (6)$$

Псевдокод вычисления C_{jih} представлен на рис. 2 [8].

Вход : o_j, c_i, o_h, d_j, s_j
Выход: C_{jih}
if $\rho(o_j, c_i) > d_j$ **and** $\rho(o_j, o_h) > d_j$ **then**
  $C_{jih} \leftarrow 0$
else if $\rho(o_j, c_i) = d_j$ **then**
  if $\rho(o_j, o_h) < s_j$ **then**
    $C_{jih} \leftarrow \rho(o_j, o_h) - d_j$
  else
    $C_{jih} \leftarrow s_j - d_j$
  end
else if $\rho(o_j, o_h) < d_j$ **then**
  $C_{jih} \leftarrow \rho(o_j, o_h) - d_j$
end

Рис. 2. Вычисление C_{jih}

4. Параллельный алгоритм

В данном разделе мы описываем подход к реализации алгоритма PAM на сопроцессоре Intel Xeon Phi. Подход основан на следующих принципах.

Параллелизм по данным и векторизация. С помощью технологии OpenMP мы обеспечиваем одновременное исполнение одной и той же функции над элементами исходного набора данных. Большинство циклов алгоритма PAM с арифметическими операциями были реорганизованы из скалярной формы в векторную, для эффективного исполнения на векторных арифметических устройствах сопроцессора.

В нашей реализации используется несколько механизмов для достижения *локальности данных*, то есть программа обращается к данным, расположенным близко к недавно запрошенным областям памяти. Так как сопроцессор загружает данные в кэш блоками, то области памяти близкие к ранее загруженным областям так же попадут в кэш, что приведет к росту производительности алгоритма.

На рис. 3 представлен псевдокод алгоритма PAM, адаптированного для сопроцессора Intel Xeon Phi.

Вход : Множество объектов O, количество кластеров k
Выход: Множество кластеров C
Выгрузить (offload) O, k из памяти CPU на сопроцессор;
$M \leftarrow PrepareDistanceMatrix(O)$;
$C \leftarrow BuildMedoids(M)$; `// фаза BUILD`
repeat`// фаза SWAP`
  $T_{min} \leftarrow FindBestSwap(M, C)$;
  Поменять местами c_{min} и o_{min};
until $T_{min} < 0$;
Выгрузить (offload) C из памяти сопроцессора в память CPU;

Рис. 3. Распараллеленный алгоритм PAM для сопроцессора Intel Xeon Phi

Сводная информация о подалгоритмах PAM представлена в таблице 1.

Для улучшения производительности мы используем технику предвычисления расстоя-

Таблица 1. Сводная информация о подалгоритмах PAM

Название	**Временная сложность**	**Техники распараллеливания**
PrepareDistanceMatrix	$O(pn^2)$	OpenMP, векторизация
BuildMedoids	$O(kn^2)$	OpenMP, векторизация
FindBestSwap	$O(k(n-k)^2)$	OpenMP

ний между всеми объектами множества O. В этом случае нет необходимости в вычислении расстояний на каждой итерации алгоритма PAM, так как все расстояния сохранены в матрице расстояний M.

Алгоритм PAM оперирует большим количеством массивов данных, не помещающихся в кэш-памяти L2 сопроцессора Intel Xeon Phi. Мы обрабатываем данные блоками по L элементов для обеспечения локальности данных. Рекомендуемое [7] значение для L равно 16. Кроме того, в нашей задаче n должно быть кратно L. Мы использовали значение $L = 32$.

Подалгоритм *PrepareDistanceMatrix* инициализирует матрицу расстояний (см. рис. 4). В отличие от [8] мы храним матрицу расстояний в полной форме, а не в верхнетреугольной форме для достижения лучшей локальности данных во всех оставшихся подалгоритмах. Для достижения лучшей производительности мы использовали *тайлинг* [7].

```
   Вход  : Множество объектов O
   Выход: Матрица расстояний M
 1 forall the o_i таких, что 1 ≤ i ≤ n do // parallelized
 2     for j = 1 до n шаг L do
 3         for k = 1 to p do
 4             for l такое, что j ≤ l ≤ j + L do // vectorized
                   // доступ к o_l организован с применением тайлинга
 5                 m_il ← m_il + (o_i[k] − o_l[k])^2;
 6             end
 7         end
 8         for l такое, что j ≤ l ≤ j + L do // vectorized
 9             m_il ← √m_il;
10         end
11     end
12 end
```

Рис. 4. Вычисление матрицы расстояний

Подалгоритм *BuildMedoids* реализует фазу BUILD (см. рис. 5) в соответствии с формулами (2)–(5).

Подалгоритм *FindBestSwap* реализует фазу SWAP (см. рис. 6). Он перебирает все пары (c_i, o_h) объектов, где c_i медоид, а o_h не-медоид, вычисляет величину T_{ih} для каждой пары и возвращает минимальное значение T_{min}.

5. Вычислительные эксперименты

Для оценки разработанного алгоритма мы выполнили эксперименты на узле суперкомпьютера «Торнадо ЮУрГУ»[1], спецификации которого представлены в таблице 2. Экспе-

[1] supercomputer.susu.ru/en/computers/tornado/

Вход : Матрица расстояний M
Выход: Множество медоидов C
forall the $i = 1$ *to* n **do** `// parallelized`
  if $\sum_{j=1}^{n} m_{ij}$ *минимальна* **then** `// вычисление суммы векторизовано`
    $c_1 \leftarrow o_i$;
  end
end
Инициализировать вектор расстояний до ближайшего медоида D
for $l = 2$ *to* k **do**
  forall the $i = 1$ *to* n **do** `// parallelized`
    `// вычисление суммы векторизовано`
    if $\sum_{j=1}^{n} \min(d_j, m_{ij})$ *минимальна* **then**
      $c_l \leftarrow o_i$;
    end
  end
  Обновить D;
end

Рис. 5. Фаза BUILD

Вход : Матрица расстояний M, множество медоидов C
Выход: T_{min}
Инициализировать массив величин переключения объектов T
forall the o_h *таких, что* $1 \leq h \leq n$ *и* o_h *не медоид* **do** `// parallelized`
  for $l = 1$ *до* n *шаг* L **do**
    for $i = 1$ *до* k **do**
      $T_{ih} \leftarrow T_{ih} + \sum_{j=l}^{l+L} C_{jih}$;
    end
  end
end
$T_{min} \leftarrow \min_{1 \leq h \leq n, 1 \leq i \leq k} T_{ih}$;

Рис. 6. Фаза SWAP

рименты проводились над данными одинарной точности. Сопроцессор Intel Xeon Phi использован в режиме *offload*. Мы измеряли время работы алгоритма PAM в зависимости от количества обрабатываемых данных и исследовали влияние свойств наборов данных на время работы подалгоритмов PAM.

Характеристики наборов данных, использованных в экспериментах, представлены в таблице 3.

Результаты экспериментов на данных FCS Human представлены на рис. 7(a). Данные из набора FCS Human имеют большую размерность, поэтому наибольшее время работы занимает процесс вычисления матрицы расстояний. Вычисление матрицы расстояний на Intel Xeon Phi в два раза эффективнее, чем на процессоре Intel Xeon.

Результаты экспериментов на данных гистограмм фотобазы Corel представлены на рис. 7(b). Размерность данных небольшая, поэтому подготовка матрицы расстояний не за-

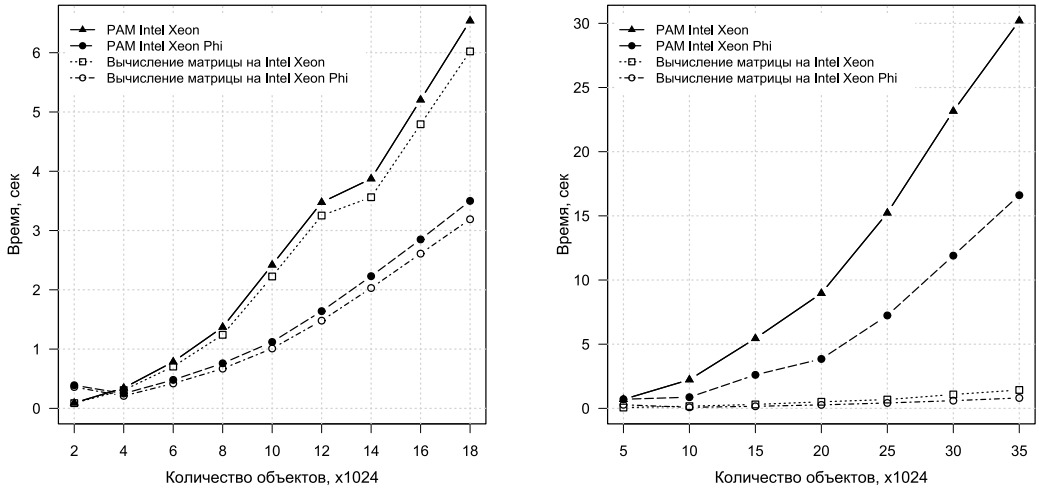
Таблица 2. Спецификация узла суперкомпьютера «Торнадо ЮУрГУ»

Спецификации	Процессор	Сопроцессор
Модель	Intel Xeon X5680	Intel Xeon Phi SE10X
Количество ядер	6	61
Тактовая частота, ГГц	3,33	1,1
Количество нитей на ядро	2	4
Пиковая производительность, TFLOPS	0,371	1,076

Таблица 3. Характеристики экспериментальных наборов данных

Набор данных	p	k	$n, \times 2^{10}$	
			min	max
FCS Human [2]	423	10	2	18
Corel Image Histogram [11]	32	10	5	35

нимает много времени. Алгоритм PAM в два раза медленнее на процессоре Intel Xeon, чем на сопроцессоре Intel Xeon Phi.



(a) Производительность на данных FCS Human (b) Производительность на данных гистограмм фотобазы Corel

Рис. 7. Результаты экспериментов

Эксперименты показали, что скорость работы алгоритма PAM определяется природой кластеризуемых данных. Наиболее трудоемким этапом для обработки данных большой размерности является вычисление матрицы расстояний. При работе с данными небольшой размерности время работы подалгоритмов PAM значительно превосходит время вычисления матрицы расстояний.

6. Заключение

В работе описана параллельная версия алгоритма кластеризации Partitioning Around Medoids для сопроцессора Intel Xeon Phi. Распараллеливание выполнено на основе OpenMP. Циклические операции преобразованы для выполнения векторизации. Алгоритм использует матрицу предвычисленных расстояний, которая хранится в памяти сопроцессора. Алгоритм хранит данные в непрерывных массивах и обрабатывает данные блоками для обеспечения локальности данных и, как следствие, лучшей производительности.

Представлены результаты вычислительных экспериментов, подтверждающие эффективность разработанного алгоритма. Эксперименты показали, что скорость работы алгоритма PAM определяется природой кластеризуемых данных. Наиболее трудоемким этапом для обработки данных большой размерности является вычисление матрицы расстояний. При работе с данными небольшой размерности время работы подалгоритмов PAM значительно превосходит время вычисления матрицы расстояний.

В качестве возможного направления дальнейших исследований интересными представляются следующие направления: модернизация разработанного алгоритма для случая вычислительного узла с несколькими сопроцессорами Intel Xeon Phi и расширение данного алгоритма для кластерной системы, вычислительные узлы которой оснащены сопроцессорами Intel Xeon Phi.

Литература

1. Broin P. O., Smith T. J., Golden A. A. J. Alignment-free Clustering of Transcription Factor Binding Motifs using a Genetic-k-Medoids Approach // BMC Bioinformatics. 2015. Vol. 16, N. 1. P. 22–33.

2. 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. Vol. 43, N. 6. P. 932–944.

3. Espenshade J., Pangborn A., Laszewski G., Roberts D., Cavenaugh J. S. Accelerating Partitional Algorithms for Flow Cytometry on GPUs // International Symposium on Parallel and Distributed Processing with Applications, August 10–12, 2009, Chengdu, Sichuan, China, Proceedings. IEEE. P. 226–233.

4. Han J., Kamber M. Data Mining: Concepts and Techniques, 3rd Edition. Morgan Kaufmann Publishers Inc., 2011. 744 p.

5. Huang Z. Extensions to the k-Means Algorithm for Clustering Large Data Sets with Categorical Values // Data Min. Knowl. Discov. 1998. Vol. 2, N. 3. P. 283–304.

6. Hussain H. M., Benkrid K., Ebrahim A., Erdogan A. T., Seker H. Novel Dynamic Partial Reconfiguration Implementation of K-Means Clustering on FPGAs: Comparative Results with GPPs and GPUs // Int. J. Reconfig. Comp. 2012. Vol. 2012, P. 135 926:1–135 926:15.

7. Jeffers J., Reinders O. Intel Xeon Phi Coprocessor High Performance Programming Morgan Kaufmann Publishers Inc., 2013. 432 p.

8. Kaufman L., Rousseeuw P. J. Finding Groups in Data: An Introduction to Cluster Analysis John Wiley, 1990.

9. Kohlhoff K., Sosnick M.H., Hsu W., Pande V., Altman R. CAMPAIGN: an Open-Source Library of GPU-Accelerated Data Clustering Algorithms // Bioinformatics. 2011. Vol. 27, N. 16. P. 2321–2322.

10. Lloyd S. P. Least squares quantization in PCM // IEEE Transactions on Information Theory. 1982. Vol. 28, N. 2. P. 129–136.

11. Ortega M., Rui Y., Chakrabarti K., Porkaew K., Mehrotra S., Huang T. S. Supporting Ranked Boolean Similarity Queries in MARS // IEEE Trans. Knowl. Data Eng. 1998. Vol. 10, N. 6. P. 905–925.

12. Patwary M. M. A., Satish N., Sundaram N., Manne F., Habib S., Dubey P. Pardicle: Parallel Approximate Density-Based Clustering // International Conference for High Performance Computing, Networking, Storage and Analysis, November 16–21, 2014, New Orleans, LA, USA, Proceedings. IEEE. P. 560–571.

13. Reynolds A. P., Richards G., de la Iglesia B., Rayward-Smith V. J. Clustering Rules: a Comparison of Partitioning and Hierarchical Clustering Algorithms // Journal of Mathematical Modelling and Algorithms. 2006. Vol. 5, N. 4. P. 475–504.

14. Wei G., An H., Dong T., Li H. A Novel micro-Blog Sentiment Analysis Approach by Longest Common sequence and k-Medoids // 18th Pacific Asia Conference on Information Systems, June 24–28, 2014, Chengdu, China, Proceedings. P. 38–47.

15. Xiao Y., Feng R., Leung C., Sum P. GPU Accelerated Spherical K-Means Training // 20th International Conference on Neural Information Processing, November 3–7, 2013, Daegu, Korea, Proceedings. Springer. P. 392–399.

16. Yan B., Zhang Y., Yang Z., Su H., Zheng H. DVT-PKM: An Improved GPU Based Parallel K-Means Algorithm // 10th International Conference on Intelligent Computing Methodologies, August 3–6, 2014, Taiyuan, China, Proceedings. Springer. P. 591–601.

17. Zhang T., Xia Y., Zhu Q., Liu Y., Shen J. Mining Related Information of Traffic Flows on Lanes by k-Medoids // 11th International Conference on Fuzzy Systems and Knowledge Discovery, August 19–21, 2014, Xiamen, China, Proceedings. P. 390–396.

18. Zheng H., Wu J. Accelerate K-means Algorithm by Using GPU in the Hadoop Framework // Web-Age Information Management, June 16–18, 2014, Macau, China, Proceedings. Springer. P. 177–186.

Parallel clustering algorithm for Intel Xeon Phi coprocessor

Timofey Rechkalov

Keywords: clustering, PAM, Intel Xeon Phi, OpenMP

Article describes parallel version of Partitioning Around Medoids algorithm for Intel Xeon Phi coprocessor. It is based on OpenMP technology. Loop operations adopted for vectorization. Algorithm uses distance matrix in coprocessor memory. Experiment results show effectiveness of suggested approach.