Accelerating Medoids-based Clustering with the Intel Many Integrated Core Architecture

Timofey Rechkalov
South Ural State University
Chelyabinsk, Russia
Email: trechkalov@yandex.ru

Mikhail Zymbler
South Ural State University
Chelyabinsk, Russia
Email: mzym@susu.ru

***Abstract*—The Partition Around Medoids (PAM) is a variation of well known k-Means clustering algorithm where center of each cluster should be chosen as an object of clustered set of objects. PAM is used in a wide spectrum of applications, e.g. text analysis, bioinformatics, intelligent transportation systems, etc. There are approaches to speed up k-Means and PAM algorithms by means of graphic accelerators but there none for accelerators based on the Intel Many Integrated Core architecture. This paper presents a parallel version of PAM for the Intel Xeon Phi many-core coprocessor. Parallelization is based on the OpenMP technology. Loop operations are adapted to provide vectorization. Distance matrix is precomputed and stored in the coprocessor's memory. Experimental results are presented and confirm the efficiency of the algorithm.**

***Index Terms*—data mining, clustering, k-Means, Partition Around Medoids, parallel computing, OpenMP, Intel Many Integrated Core architecture, Intel Xeon Phi coprocessor.**

I. Introduction

Clustering is one of the basic problems of data mining aimed to organizing a set of data objects into subsets (clusters) such that objects in a cluster are similar to one another, yet dissimilar to objects in other clusters. Similarity is commonly defined in terms of how close the objects are, and is based on a specified distance metric [1].

The most fundamental method of clustering is partitioning, which organizes the objects of a set into several exclusive groups. More formally, given a set of n objects, a partitioning algorithm constructs k partitions of the data, where each partition represents a cluster and $k \leq n$. The algorithm divides the data objects into k clusters such that each cluster contains at least one object and each object is in the only cluster. An object is assigned to a closest cluster based on the distance measure between the object and the cluster center. Then algorithm iteratively improves the within-cluster variation by computing the new cluster center using the objects assigned to the cluster in the previous iteration. All the objects are then reassigned using the updated means as the new cluster centers. The iterations continue until the assignment is stable, that is, the clusters formed in the current round are the same as those formed in the previous round. Partitioning clustering algorithms differ in a way of calculation cluster centers, e.g. k-Means [2] and k-Modes [3] algorithms uses mean and mode values of clustered objects respectively, whereas k-Medoids algorithm uses an object of clustered data set (called medoid).

The Partition Around Medoids (PAM) [4] is a variation of k-Means, which is used in a wide spectrum of applications, e.g. text analysis [5], bioinformatics [6], intelligent transport systems [7], etc. The complexity of each iteration in the PAM algorithm is $O(k(n-k)^2)$. For large values of n and k computations are very costly. That is why there are approaches to speed up k-Means and PAM algorithms by means of graphic accelerators, e.g. [8], [9]. At the same time there none for modern accelerators based on the Intel Many Integrated Core (MIC) [10] architecture. In this paper we present a parallel version of PAM for MIC accelerators (what was done for the first time, to the best of our knowledge).

The remaining part of the paper is organized as follows. Section II gives an overview of serial PAM algorithm and briefly considers the Intel Xeon Phi many-core coprocessor's architecture and programming model. In section III we describe parallelization of PAM adapted for the Intel MIC architecture. The results of the experiments evaluating the algorithm are presented in section IV. Section V discusses related work. Section VI contains concluding remarks and directions for future research.

II. Background of the Research

A. Serial PAM Algorithm

To provide formal description of the PAM [11] algorithm we will use the following notation. Let $O = \{o_1, o_2, \ldots, o_n\}$ is a set of objects to be clustered where each object is a tuple consisting of p real-valued attributes. Let k is the number of clusters, $k \ll n$, and $C = \{c_1, c_2, \ldots, c_k\}$ is a set of medoids, $C \subset O$, and $\rho : O \times C \to R$ is a distance metric.

The algorithm takes the form of a steepest ascent hill climber, using a simple swap neighbourhood operation. In each iteration medoid object c_i and non-medoid object o_j are selected that produce the best clustering when their roles are switched. The objective function used is the sum of the distances from each object to the closest medoid:

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

Algorithm 1 depicts PAM pseudocode. PAM consists of two phases, namely BUILD and SWAP. In the first phase an initial clustering is obtained by the successive selection

Algorithm 1 PAM

Input: Set of objects O, number of clusters k
Output: Set of k clusters
  Init C {BUILD phase}
  repeat {SWAP phase}
    Calculate T_{min}
    Swap c_{min} and o_{min}
  until $T_{min} < 0$

of representative objects until k objects have been found. The first object c_1 is the one for which the sum of the distances to all other objects is as small as possible:

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

Object c_1 is the most centrally located in O set. Subsequently, at each step another object is selected, which decreases the objective function as much as possible. This object is the one for which the minimal distance to all selected medoids and distance to this object is as small as possible:

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

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

$$\dots$$

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

This process is continued until k objects have been found.

In the second phase of the algorithm, it is attempted to improve C (i.e. set of medoids) and therefore also to improve the clustering yielded by this set. Algorithm searches for a pair of objects (c_{min}, o_{min}), which minimizes the objective function. This is done by considering all pairs of objects (c_i, o_h) where c_i is a medoid and o_h is not a medoid. It is determined what effect is obtained on the objective function when a swap is carried out, i.e., when object c_i is no longer selected as a medoid but object o_h is. Let denote this effect as T_{ih}, then minimum value of T_{min} is achieved with (c_{min}, o_{min}) pair. If $T_{min} > 0$ then C set can not be improved so the algorithm stops.

Let us consider calculation of the T_{ih} effect using the following notation. Let $D = \{d_1, d_2, \dots, d_n\}$ is a set of distances from each object to the closest medoid. Let $S = \{s_1, s_2, \dots, s_n\}$ is a set of distances from each object to second closest medoid. Let C_{jih} is a contribution of non selected object o_j to the effect T_{ih} of a swap between c_i and o_h on the objective function. In this case T_{ih} is the sum of the contributions C_{jih}:

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

Algorithm 2 [11] depicts pseudocode of calculating C_{jih}.

Algorithm 2 Calculating C_{jih}

Input: o_j, c_i, o_h, d_j, s_j
Output: 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 if
  else if $\rho(o_j, o_h) < d_j$ **then**
    $C_{jih} \leftarrow \rho(o_j, o_h) - d_j$
  end if

B. The Intel Xeon Phi Architecture and Programming Model

The Intel Xeon Phi coprocessor is an x86 many-core coprocessor of 61 cores, connected by a high-performance on-die bidirectional interconnect where each core supports 4× hyperthreading and contains 512-bit wide vector processor unit (VPU). Each core has two levels of cache memory: a 32 Kb L1 data cache, a 32 Kb L1 instruction cache, and a core-private 512 Kb unified L2 cache. The Intel Xeon Phi coprocessor is to be connected to a host computer via a PCI Express system interface. Being based on Intel x86 architecture, the Intel Xeon Phi coprocessor supports the same programming tools and models as a regular Intel Xeon processor.

There are three programming modes to deal with the Intel Xeon Phi coprocessor: native, offload and symmetric. In native mode the application runs independently, on the coprocessor only. In offload mode the application is running on the host and offloads computationally intensive part of work to the coprocessor. The symmetric mode allows the coprocessor to communicate with other devices by means of Message Passing Interface (MPI).

III. Parallel PAM Algorithm for MIC Accelerators

In this section we describe an approach to implementation of PAM algorithm for the Intel Xeon Phi coprocessor. The approach is based on the following principles.

Data parallelism and vectorization. Using OpenMP technology we perform simultaneous execution on multiple cores of the same function across the elements of a dataset. Most loops of the original PAM algorithm with arithmetic operations were implemented to provide conversion of such operations from scalar form to vector form to be effectively computed by the coprocessor's VPUs.

Our implementation strives to provide *data locality* as much as possible, i.e. the program uses data close to recently accessed locations. Since the coprocessor loads a chunk of memory around an accessed location into the cache, locations

close to recently accessed locations are also likely to be in the cache so finally it increases algorithm's performance.

Algorithm 3 depicts PAM pseudocode adapted for use on the Intel Xeon Phi many-core coprocessor.

Algorithm 3 Parallel PAM for Intel Xeon Phi coprocessor

Input: Set of objects O, number of clusters k
Output: Set of k clusters
  Offload O, k from CPU to coprocessor
  $M \leftarrow PrepareDistanceMatrix(O)$
  $C \leftarrow BuildMedoids(M)$ {BUILD phase}
  repeat {SWAP phase}
    $T_{min} \leftarrow FindBestSwap(M, C)$
    Swap c_{min} and o_{min}
  until $T_{min} < 0$
  Offload C from coprocessor to CPU

The summary of parallel PAM subalgorithms is presented in Tab. I.

Table I
SUMMARY OF PARALLEL PAM SUBALGORITHMS

Name	Complexity	Parallelizing technique(s)
PrepareDistanceMatrix	$O(pn^2)$	OpenMP, vectorization
BuildMedoids	$O(kn^2)$	OpenMP, vectorization
FindBestSwap	$O(k(n-k)^2)$	OpenMP

To improve performance we use precomputing technique calculating distances between all objects of O set in advance. There is no need for repeated calculation of distances at each iteration, since distances simply can be looked up in M matrix.

The PAM algorithm deals with a lot of data arrays which are not fit into Intel Xeon Phi L2 memory cache. We process data by chunks of L bytes to satisfy data locality requirement. It is recommended [10] to set L to 16 and try multiplying or dividing by 2 and use n divisible by L. In our work we use $L = 32$.

The *PrepareDistanceMatrix* subalgorithm initializes distance matrix (see Algorithm 4). Unlike in [11] we store matrix in full form (not in upper triangular form) to provide better data locality for the rest of subalgorithms. To achieve better performance of this subalgorithm we use *tiling* technique [10].

The *BuildMedoids* subalgorithm implements BUILD phase (see Algorithm 5) according to formulas (2)–(5).

The *FindBestSwap* subalgorithm implements SWAP phase (see Algorithm 6). It checks all pairs of (c_i, o_h) objects where c_i is a medoid and o_h is not a medoid, calculates the effect for each T_{ih} swapping and returns the minimal one.

IV. EXPERIMENTAL EVALUATION

To evaluate the developed algorithm we performed experiments on the Tornado SUSU[1] supercomputer's node (see Tab. II for its specifications). Experiments were performed on single precision data, the coprocessor was used in offload mode. We measured PAM runtime while varying number of

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

Algorithm 4 Prepare Distance Matrix

Input: Set of objects O
Output: Distance matrix M
  allocate M
  parallel for o_i such that $1 \le i \le n$ **do**
    for $j = 1$ to n step L **do**
      for $k = 1$ to p **do**
        for l such that $j \le l \le j + L$ **do** {vectorized}
          {access to o_l is tiled}
          $m_{il} \leftarrow m_{il} + (o_i[k] - o_l[k])^2$
        end for
      end for
      for l such that $j \le l \le j + L$ **do** {vectorized}
        $m_{il} \leftarrow \sqrt{m_{il}}$
      end for
    end for
  end for

Algorithm 5 Build Medoids

Input: Distance matrix M
Output: Set of medoids C
  parallel for $i = 1$ to n **do**
    if $\sum_{j=1}^{n} m_{ij}$ is minimal **then** {sum is vectorized}
      $c_1 \leftarrow o_i$
    end if
  end for
  Init D distances to nearest medoid
  for $l = 2$ to k **do**
    parallel for $i = 1$ to n **do**
      {sum is vectorized}
      if $\sum_{j=1}^{n} \min(d_j, m_{ij})$ is minimal **then**
        $c_l \leftarrow o_i$
      end if
    end for
    Update D
  end for

Algorithm 6 Find Best Swap

Input: Distance matrix M, set of medoids C
Output: T_{min}
  Init T array of swap effects
  parallel for o_h such that $1 \le h \le n$ and o_h is not a medoid **do**
    for $l = 1$ to n step L **do**
      for $i = 1$ to k **do**
        $T_{ih} \leftarrow T_{ih} + \sum_{j=l}^{l+L} C_{jih}$
      end for
    end for
  end for
  $T_{min} \leftarrow \min_{1 \le h \le n, 1 \le i \le k} T_{ih}$

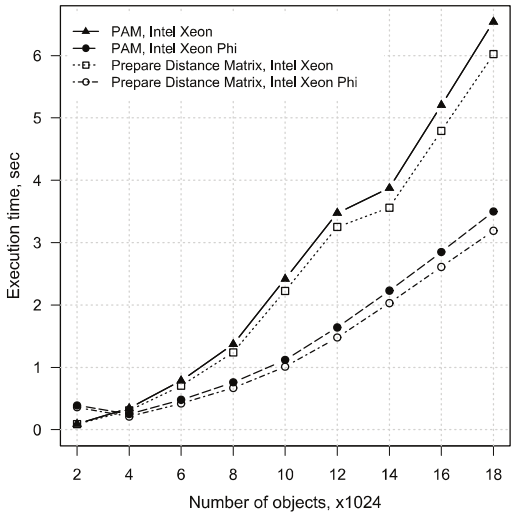

Figure 1. PAM performance on FCS Human dataset

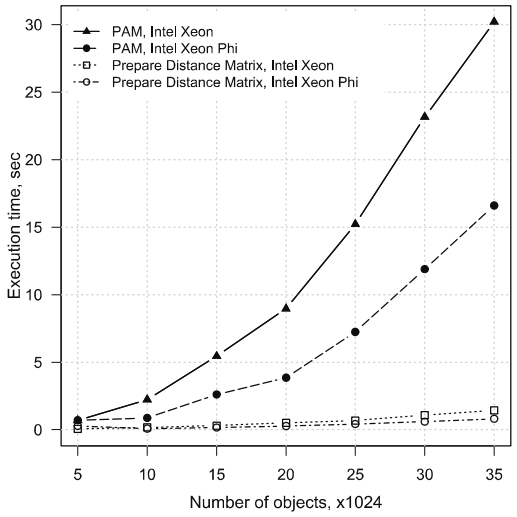

Figure 2. PAM performance on Corel Image Histogram dataset

clustered objects and investigated the influence of dataset properties on runtime of PAM subalgorithms.

Table II
SPECIFICATIONS OF TORNADO SUSU SUPERCOMPUTER NODE

Specifications	**Processor**	**Coprocessor**
Model	Xeon X5680	Xeon Phi SE10X
Cores	6	61
Frequency, GHz	3.33	1.1
Threads per core	2	4
Peak performance, TFLOPS	0.371	1.076

Datasets used in experiments are summarized in Tab. III.

Table III
DATASETS SUMMARY

Dataset	p	k	$n, \times 2^{10}$	
			min	max
FCS Human [12]	423	10	2	18
Corel Image Histogram [13]	32	120	5	35

Experimental results for FCS Human dataset are introduced in figure 1. FCS Human dataset has large dimension so the most time is taken by calculation of distance matrix. Calculation of distance matrix on the Intel Xeon Phi is two times faster then on the Intel Xeon.

Experimental results for Corel Image Histogram dataset are introduced in figure 2. Data dimension is small so preparing distance matrix does not require much time. The PAM algorithm is two times slower on the Intel Xeon than on the Intel Xeon Phi.

Experiments show that PAM performance depends on clustered data nature. The most complex thing for large dimension data is calculation of distance matrix. In case of small dimension data the rest of the PAM subalgorithms take significantly larger part of runtime than distance matrix calculation.

V. RELATED WORK

A significant amount of work has been done in the area of cluster analysis. The classical k-Means and k-Medoids algorithms was suggested in [2], [3]. The original PAM algorithm was proposed in [11].

The research devoted to accelerating clustering algorithms using parallel hardware includes the following. In [14] FPGA and GPU implementations of k-Means are compared. Authors of [15] describe improvements of k-Means reducing data transfers between CPU and GPU. In [16] a technique improving data distribution among GPU threads in k-Means is suggested. k-Means implementation for Hadoop framework with GPUs is described in [17]. In [8] several clustering methods on GPU including k-Medoids are implemented. A GPU-based framework for clustering genetic data using k-Medoids is described in [9] .

In our opinion currently the potential of the Intel MIC accelerators for cluster analysis is underestimated. To the best of our knowledge there is only paper [18] devoted to adaptation the DBSCAN density-based clustering algorithm for the Intel MIC architecture. The contribution of this paper is technique of acceleration of the Partitioning Around Medoids clustering algorithm with the Intel Xeon Phi many-core coprocessor.

VI. CONCLUSION

The paper has described a parallel version of Partitioning Around Medoids clustering algorithm for the Intel Xeon Phi many-core coprocessor. Parallelizing is based on OpenMP technology. Loop operations are adapted for vectorization. Algorithm uses a distance matrix calculated in advance and stored in the coprocessor's memory. Algorithm stores data in

continuous arrays and process data by chunks to achieve data locality for better performance.

Experimental results show effectiveness of suggested approach. Experiments show that PAM performance depends on clustered data nature. The most complex thing for large dimension data is calculation of distance matrix. In case of small dimension data the rest of the PAM subalgorithms take significantly larger part of runtime than distance matrix calculation.

As future work we plan to extend our research in the following directions: implement our algorithm for the cases of several coprocessors and cluster system based on nodes equipped with the Intel Xeon Phi coprocessor(s).

REFERENCES

[1] J. Han, M. Kamber, and J. Pei, *Data Mining: Concepts and Techniques, Third Edition (The Morgan Kaufmann Series in Data Management Systems)*, 3rd ed. Morgan Kaufmann, 7 2011.

[2] S. P. Lloyd, "Least squares quantization in PCM," *IEEE Transactions on Information Theory*, vol. 28, no. 2, pp. 129–136, 1982.

[3] Z. Huang, "Extensions to the k-means algorithm for clustering large data sets with categorical values," *Data Min. Knowl. Discov.*, vol. 2, no. 3, pp. 283–304, 1998.

[4] A. P. Reynolds, G. Richards, B. de la Iglesia, and V. J. Rayward-Smith, "Clustering rules: A comparison of partitioning and hierarchical clustering algorithms," *J. Math. Model. Algorithms*, vol. 5, no. 4, pp. 475–504, 2006.

[5] G. Wei, H. An, T. Dong, and H. Li, "A novel micro-blog sentiment analysis approach by longest common sequence and k-medoids," in *18th Pacific Asia Conference on Information Systems, PACIS 2014, Chengdu, China, June 24-28, 2014*, K. Siau, Q. Li, and X. Guo, Eds., 2014, p. 38.

[6] P. Ó. Broin, T. J. Smith, and A. Golden, "Alignment-free clustering of transcription factor binding motifs using a genetic-k-medoids approach," *BMC Bioinformatics*, vol. 16, p. 22, 2015.

[7] T. Zhang, Y. Xia, Q. Zhu, Y. Liu, and J. Shen, "Mining related information of traffic flows on lanes by k-medoids," in *11th International Conference on Fuzzy Systems and Knowledge Discovery, FSKD 2014, Xiamen, China, August 19-21, 2014*. IEEE, 2014, pp. 390–396.

[8] J. Espenshade, A. Pangborn, G. von Laszewski, D. Roberts, and J. S. Cavenaugh, "Accelerating partitional algorithms for flow cytometry on GPUs," in *IEEE International Symposium on Parallel and Distributed Processing with Applications, ISPA 2009, Chengdu, Sichuan, China, 10-12 August 2009*. IEEE, 2009, pp. 226–233.

[9] K. J. Kohlhoff, M. H. Sosnick, W. T. Hsu, V. S. Pande, and R. B. Altman, "CAMPAIGN: an open-source library of GPU-accelerated data clustering algorithms," *Bioinformatics*, vol. 27, no. 16, pp. 2321–2322, 2011.

[10] J. Jeffers and J. Reinders, *Intel Xeon Phi Coprocessor High-Performance Programming*, 1st ed. Morgan Kaufmann, 3 2013.

[11] L. Kaufman and P. J. Rousseeuw, *Finding Groups in Data: An Introduction to Cluster Analysis*. John Wiley, 1990.

[12] J. M. Engreitz, B. J. D. Jr., J. J. Marshall, and R. B. Altman, "Independent component analysis: Mining microarray data for fundamental human gene expression modules," *Journal of Biomedical Informatics*, vol. 43, no. 6, pp. 932–944, 2010.

[13] M. Ortega, Y. Rui, K. Chakrabarti, K. Porkaew, S. Mehrotra, and T. S. Huang, "Supporting ranked boolean similarity queries in MARS," *IEEE Trans. Knowl. Data Eng.*, vol. 10, no. 6, pp. 905–925, 1998.

[14] H. M. Hussain, K. Benkrid, A. Ebrahim, A. T. Erdogan, and H. Seker, "Novel dynamic partial reconfiguration implementation of k-means clustering on FPGAs: Comparative results with GPPs and GPUs," *Int. J. Reconfig. Comp.*, vol. 2012, pp. 135 926:1–135 926:15, 2012.

[15] Y. Xiao, R. Feng, C. Leung, and P. Sum, "GPU accelerated spherical k-means training," in *Neural Information Processing - 20th International Conference, ICONIP 2013, Daegu, Korea, November 3-7, 2013. Proceedings, Part II*, ser. Lecture Notes in Computer Science, M. Lee, A. Hirose, Z. Hou, and R. M. Kil, Eds., vol. 8227. Springer, 2013, pp. 392–399.

[16] B. Yan, Y. Zhang, Z. Yang, H. Su, and H. Zheng, "DVT-PKM: an improved GPU based parallel k-means algorithm," in *Intelligent Computing Methodologies - 10th International Conference, ICIC 2014, Taiyuan, China, August 3-6, 2014. Proceedings*, ser. Lecture Notes in Computer Science, D. Huang, K. Jo, and L. Wang, Eds., vol. 8589. Springer, 2014, pp. 591–601.

[17] H. Zheng and J. Wu, "Accelerate k-means algorithm by using GPU in the Hadoop framework," in *Web-Age Information Management - WAIM 2014 International Workshops: BigEM, HardBD, DaNoS, HRSUNE, BIDASYS, Macau, China, June 16-18, 2014 Revised Selected Papers*, ser. Lecture Notes in Computer Science, Y. Chen, W. Balke, J. Xu, W. Xu, P. Jin, X. Lin, T. Y. Tang, and E. Hwang, Eds., vol. 8597. Springer, 2014, pp. 177–186.

[18] M. M. A. Patwary, N. Satish, N. Sundaram, F. Manne, S. Habib, and P. Dubey, "Pardicle: Parallel approximate density-based clustering," in *International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2014, New Orleans, LA, USA, November 16-21, 2014*, T. Damkroger and J. Dongarra, Eds. IEEE, 2014, pp. 560–571.