Partition Around Medoids Clustering on the Intel Xeon Phi Many-Core Coprocessor

Timofey V. Rechkalov

South Ural State University, Chelyabinsk, Russia
trechkalov@yandex.ru

Abstract. The paper touches upon the problem of implementation Partition Around Medoids (PAM) clustering algorithm for the Intel Many Integrated Core architecture. PAM is a form of well-known k-Medoids clustering algorithm and is applied in various subject domains, e.g. bioinformatics, text analysis, intelligent transportation systems, etc. An optimized version of PAM for the Intel Xeon Phi coprocessor is introduced where OpenMP parallelizing technology, loop vectorization, tiling technique and efficient distance matrix computation for Euclidean metric are used. Experimental results for different data sets confirm the efficiency of the proposed algorithm.

Keywords: data mining · clustering · k-Medoids · Partition Around Medoids · Intel Many Integrated Core architecture · Intel Xeon Phi coprocessor · parallel computing · tiling · vectorization · OpenMP

1 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.

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. 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. After

This work was financially supported by the Ministry of education and science of the Russian Federation ("Research and development on priority directions of scientific-technological complex of Russia for 2014–2020" Federal Program, contract No. 14.574.21.0035).

this cluster centers are updated and all the objects are then reassigned using 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 [11] and k-Modes [5] 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) [18] is a variation of k-Means, which is used in a wide spectrum of applications, e.g. text analysis, bioinformatics, intelligent transport systems, 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 GPU [3, 10]. At the same time there none for modern accelerators based on the Intel Many Integrated Core (MIC) [8] architecture. In spite of many recent developments for manycore platforms in data mining [13, 15, 23] and databases [1, 7] there are few ones for the Intel MIC architecture.

In this paper we present a parallel version of PAM for MIC accelerators. The remaining part of the paper is organized as follows. Section 2 gives an overview of serial PAM algorithm and discusses related work. In section 3 we describe parallelization of PAM adapted for the Intel MIC architecture. The results of the experiments evaluating the algorithm are presented in section 4. Section 5 contains summary and directions for future research.

2 Background of the Research

2.1 Serial PAM Algorithm

To provide formal description of the PAM [9] 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:

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

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 of representative objects until k objects have been found.

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} o_{min};
until $T_{min} < 0$;

Fig. 1. PAM

The first object c_1 is the one for which the sum of the distances to all other objects is as small as possible:

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

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:

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

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

$$\dots$$

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

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}:

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

Algorithm 2 [9] depicts pseudocode of 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
else if $\rho(o_j, o_h) < d_j$ **then**
    $C_{jih} \leftarrow \rho(o_j, o_h) - d_j$
end

Fig. 2. Calculating C_{jih}

2.2 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 [5, 11]. The original PAM algorithm was proposed in [9].

The research devoted to accelerating clustering algorithms using parallel hardware includes the following. In [6] FPGA and GPU implementations of k-Means are compared. Authors of [20] describe improvements of k-Means reducing data transfers between CPU and GPU. In [21] a technique improving data distribution among GPU threads in k-Means is suggested. k-Means implementation for Hadoop framework with GPUs is described in [22]. In [3] several clustering methods on GPU including k-Medoids are implemented. A GPU-based framework for clustering genetic data using k-Medoids described in [10].

In our opinion currently the potential of the Intel MIC accelerators for cluster analysis is underestimated. Paper [16] proposes modification of the DBSCAN density-based clustering algorithm for the Intel MIC architecture. In [19] a version of k-Means for CPU and Intel MIC heterogeneous architecture is presented, where authors used vectorization and sophisticated layout scheme to improve data locality. The contribution of this paper is technique of acceleration of the Partitioning Around Medoids clustering algorithm with the Intel Xeon Phi many-core coprocessor.

3 Parallel PAM Algorithm for MIC Accelerators

In this section we describe an approach to implementation of PAM algorithm for the Intel Xeon Phi coprocessor [17]. The Intel Xeon Phi coprocessor is an x86-based SMP-on-a-chip with over fifty cores. It supports 4× hardware threads per core 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 connected to other devices via the PCIe bus. Intel Xeon Phi coprocessor is based on Intel x86 architecture and it supports the same programming tools and models as a regular Intel Xeon processor. Our 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.

Input : Set of objects O, number of clusters k
Output: Set of C 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;

Fig. 3. Parallel PAM for Intel Xeon Phi coprocessor

The summary of parallel PAM subalgorithms is presented in Tab. 1.

To improve performance we use precomputing technique by means of calculating distances between all objects of O set in advance. There is no need for

Table 1. 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

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 [8] 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 [9] 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 [8].

Input : Set of objects O
Output: Distance matrix 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 */`
        $m_{il} \leftarrow m_{il} + (o_i[k] - o_l[k])^2$; `/* access to` o_l `is tiled */`
      end
    end
    for l *such that* $j \le l \le j + L$ **do** `/* vectorized */`
      $m_{il} \leftarrow \sqrt{m_{il}}$;
    end
  end
endfor

Fig. 4. Prepare Distance Matrix

Tiling is a technique for improving data reuse in cache architectures. Cache architectures generally employ least recently used (LRU) methods to determine which data is evicted from the cache as new data is requested. Therefore, the longer data remains unused, the more likely it will be evicted from the cache and no longer available immediately when needed. Tiling the access pattern can exploit data that remains in the cache from recent, previous iterations.

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

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
endfor
Init D distances to nearest medoid;
for $l = 2$ *to* k **do**
    parallel for $i = 1$ *to* n **do**
        if $\sum_{j=1}^{n} \min(d_j, m_{ij})$ *is minimal* **then** `/* sum is vectorized */`
            $c_l \leftarrow o_i$;
        end
    endfor
    Update D;
end

Fig. 5. BUILD phase

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

Fig. 6. SWAP phase

SWAP phase executes many logical operations in (see Alg. 2). By this reason two versions of the PAM algorithm were implemented. PAM-1 executes more logical operations with lesser temporary data. PAM-2 executes lesser logical operations but stores more temporary data. Preparing matrix function and BUILD phase are the same in both versions.

4 Experimental Evaluation

To evaluate the developed algorithm we performed experiments on the hardware specified in Tab. 2. Experiments were performed on single precision data, the coprocessor was used in offload mode. We measured PAM runtime while varying number of clustered objects and investigated the influence of dataset properties on runtime of PAM subalgorithms.

Table 2. Specifications of hardware

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. 3.

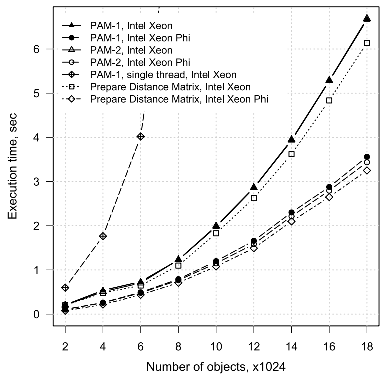
Table 3. Datasets Summary

Dataset	p	k	$n, \times 2^{10}$		Max data size, Mb	Time to transfer to coprocessor, sec
			min	max		
FCS Human [2]	423	10	2	18	29.74	0.005
Corel Image Histogram [14]	32	15	5	35	4.38	0.001
MixSim [12]	5	10	5	35	0.68	0.001
Letter Recognition [4]	16	26	2	18	1.13	0.001

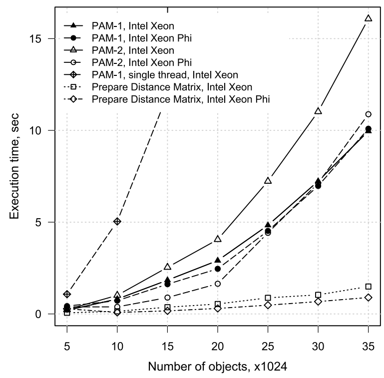
Experimental results for FCS Human dataset are introduced in Fig. 7(a). FCS Human dataset has large dimension so the most time is taken by calculation of distance matrix. Calculation of distance matrix on the Intel Xeon Phi is two times faster then on the Intel Xeon. There is no significant difference between PAM-1 and PAM-2 for this dataset.

Experimental results for Corel Image Histogram dataset are introduced in Fig. 7(b). Data dimension is small so preparing distance matrix does not require much time. PAM-1 shows similar performance on both CPU and Intel Xeon Phi. The PAM-2 algorithm is two times slower on the Intel Xeon than on the Intel Xeon Phi.

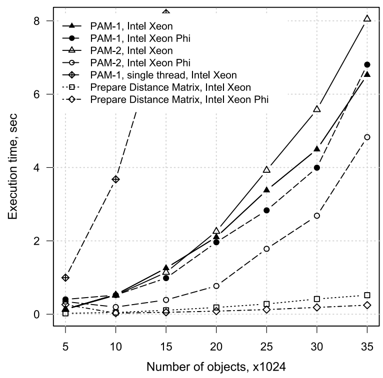
Experimental results for MixSim dataset are introduced in Fig. 7(c). Again PAM-1 shows similar performance on both CPU and the Intel Xeon Phi. PAM-2 on the Intel Xeon Phi shows best result on this dataset.



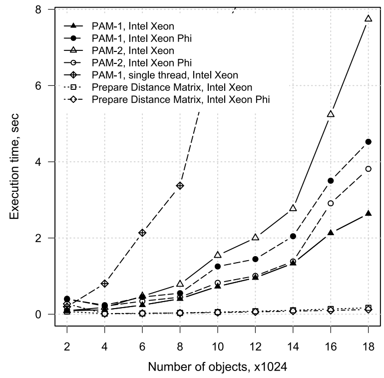
(a) FCS Human dataset



(b) Corel Image Histogram dataset



(c) MixSim dataset



(d) Letter Recognition dataset

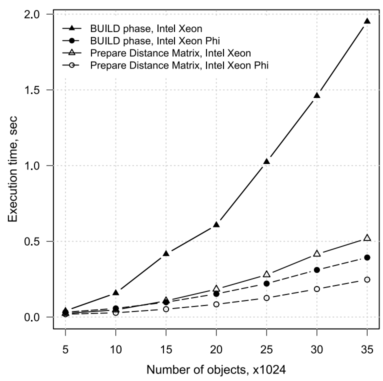
Fig. 7. Performance of the PAM algorithm

Experimental results for Letter Recognition dataset are introduced in Fig. 7(d). PAM-1 shows the best result on the Intel Xeon. Both PAM-1 and PAM-2 shows similar results on the Intel Xeon Phi.

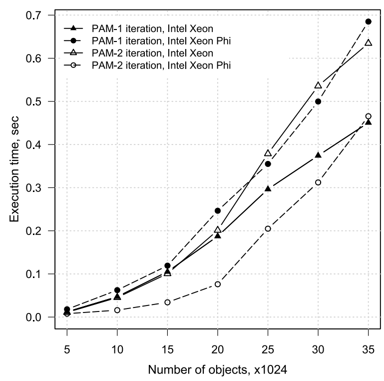
Intuitively PAM-2 is a better implementation for the Intel Xeon Phi. This suggestion is confirmed by experiments. In all tests PAM-2 is twice better on the Intel Xeon Phi than the Intel Xeon. In the same time PAM-1 is the best with the Intel Xeon only once. In other tests there is no significant difference.

To investigate this fact deeper we made more experiments to see contribution of every PAM subalgorithm in Fig. 8. Figures 8(a) and 8(c) show time of matrix calculation and BUILD phase. The Intel Xeon Phi outperforms the Intel Xeon in both subalgorithms. Figures 8(b) and 8(d) show average time of one iteration in SWAP phase. In these figures we can see that Intel Xeon Phi performance degraded faster then Intel Xeon. PAM-2 implementation looses to PAM-1 in

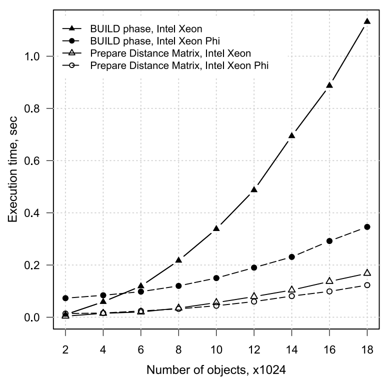
SWAP phase for big datasets so we need to continue PAM-2 improvements for Intel Xeon Phi.



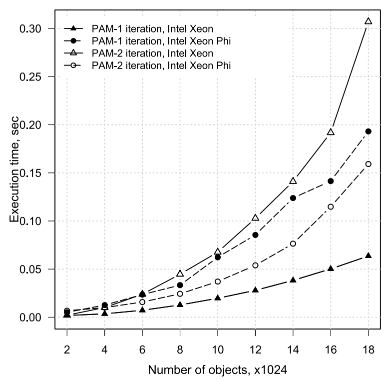
(a) MixSim: BUILD phase and prepare distance matrix timings



(b) MixSim: PAM-1 and PAM-2 iteration timings



(c) Letter Recognition: BUILD phase and prepare distance matrix timings



(d) Letter Recognition: PAM-1 and PAM-2 iteration timings

Fig. 8. Deep comparison of the PAM algorithm implementations

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. BUILD phase is more effective on the Intel Xeon Phi. SWAP phase perform better on the Intel Xeon. PAM execution on Letter Recognition dataset requires more iterations than MixSim experiment. By this reason PAM-1 shows best result with Letter Recognition and PAM-2 shows best result with MixSim.

5 Conclusion

The paper has described a parallel version of Partitioning Around Medoids clustering algorithm for the Intel Xeon Phi many-core coprocessor. An optimized version of PAM for the Intel Xeon Phi coprocessor is introduced where OpenMP parallelizing technology, loop vectorization, tiling technique and efficient distance matrix computation for Euclidean metric are used. 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. BUILD phase is more effective on the Intel Xeon Phi. SWAP phase perform better on the Intel Xeon. PAM-1 shows best result with Letter Recognition dataset and PAM-2 shows best result with MixSim.

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. Besedin, K.Y., Kostenetskiy, P.S., Prikazchikov, S.O.: Increasing efficiency of data transfer between main memory and intel xeon phi coprocessor or NVIDIA GPUS with data compression. In: Malyshkin, V. (ed.) Parallel Computing Technologies - 13th International Conference, PaCT 2015, Petrozavodsk, Russia, August 31 - September 4, 2015, Proceedings. LNCS, vol. 9251, pp. 319–323. Springer (2015)
2. Engreitz, J.M., Jr., B.J.D., Marshall, J.J., Altman, R.B.: Independent component analysis: Mining microarray data for fundamental human gene expression modules. Journal of Biomedical Informatics 43(6), 932–944 (2010)
3. Espenshade, J., Pangborn, A., von Laszewski, G., Roberts, D., Cavenaugh, J.S.: 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. pp. 226–233. IEEE (2009)
4. Frey, P.W., Slate, D.J.: Letter recognition using holland-style adaptive classifiers. Machine Learning 6, 161–182 (1991)
5. Huang, Z.: Extensions to the k-means algorithm for clustering large data sets with categorical values. Data Min. Knowl. Discov. 2(3), 283–304 (1998)
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, 135926:1–135926:15 (2012)
7. Ivanova, E., Sokolinsky, L.: Decomposition of natural join based on domain-interval fragmented column indices. In: Information and Communication Technology, Electronics and Microelectronics (MIPRO), 2015 38th International Convention on. pp. 210–213 (May 2015)

8. Jeffers, J., Reinders, J.: Intel Xeon Phi Coprocessor High-Performance Programming. Morgan Kaufmann, 1 edn. (3 2013)
9. Kaufman, L., Rousseeuw, P.J.: Finding Groups in Data: An Introduction to Cluster Analysis. John Wiley (1990)
10. Kohlhoff, K.J., Sosnick, M.H., Hsu, W.T., Pande, V.S., Altman, R.B.: CAMPAIGN: an open-source library of gpu-accelerated data clustering algorithms. Bioinformatics 27(16), 2321–2322 (2011)
11. Lloyd, S.P.: Least squares quantization in PCM. IEEE Transactions on Information Theory 28(2), 129–136 (1982)
12. Melnykov, V., Chen, W.C., Maitra, R.: Mixsim: An r package for simulating data to study performance of clustering algorithms. Journal of Statistical Software 51(12) (Jan 2012)
13. Movchan, A., Zymbler, M.L.: Time series subsequence similarity search under dynamic time warping distance on the intel many-core accelerators. In: Amato, G., Connor, R.C.H., Falchi, F., Gennaro, C. (eds.) Similarity Search and Applications - 8th International Conference, SISAP 2015, Glasgow, UK, October 12-14, 2015, Proceedings. LNCS, vol. 9371, pp. 295–306. Springer (2015)
14. 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. 10(6), 905–925 (1998)
15. Pan, C., Zymbler, M.: A parallel algorithm for market basket analysis on the cell processor. Bulletin of the South Ural State University. Series: Mathematical Modelling, Programming and Computer Software. 5, 48–57 (2010)
16. Patwary, M.M.A., Satish, N., Sundaram, N., Manne, F., Habib, S., Dubey, P.: Pardicle: Parallel approximate density-based clustering. In: Damkroger, T., Dongarra, J. (eds.) International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2014, New Orleans, LA, USA, November 16-21, 2014. pp. 560–571. IEEE (2014)
17. Rechkalov, T., Zymbler, M.: Accelerating medoids-based clustering with the intel many integrated core architecture. In: Application of Information and Communication Technologies - 9th International Conference, AICT 2015, Rostov-on-Don, Russia, October 14-16, 2015. Proceedings. pp. 413–417. IEEE (2015)
18. Reynolds, A.P., Richards, G., de la Iglesia, B., Rayward-Smith, V.J.: Clustering rules: A comparison of partitioning and hierarchical clustering algorithms. J. Math. Model. Algorithms 5(4), 475–504 (2006)
19. Wu, F., Wu, Q., Tan, Y., Wei, L., Shao, L., Gao, L.: A vectorized k-means algorithm for intel many integrated core architecture. In: Wu, C., Cohen, A. (eds.) Advanced Parallel Processing Technologies - 10th International Symposium, APPT 2013, Stockholm, Sweden, August 27-28, 2013, Revised Selected Papers. LNCS, vol. 8299, pp. 277–294. Springer (2013)
20. Xiao, Y., Feng, R., Leung, C., Sum, P.: GPU accelerated spherical k-means training. In: Lee, M., Hirose, A., Hou, Z., Kil, R.M. (eds.) Neural Information Processing - 20th International Conference, ICONIP 2013, Daegu, Korea, November 3-7, 2013. Proceedings, Part II. LNCS, vol. 8227, pp. 392–399. Springer (2013)
21. Yan, B., Zhang, Y., Yang, Z., Su, H., Zheng, H.: DVT-PKM: an improved GPU based parallel k-means algorithm. In: Huang, D., Jo, K., Wang, L. (eds.) Intelligent Computing Methodologies - 10th International Conference, ICIC 2014, Taiyuan, China, August 3-6, 2014. Proceedings. LNCS, vol. 8589, pp. 591–601. Springer (2014)

22. Zheng, H., Wu, J.: Accelerate k-means algorithm by using GPU in the hadoop framework. In: Chen, Y., Balke, W., Xu, J., Xu, W., Jin, P., Lin, X., Tang, T.Y., Hwang, E. (eds.) Web-Age Information Management - WAIM 2014 International Workshops: BigEM, HardBD, DaNoS, HRSUNE, BIDASYS, Macau, China, June 16-18, 2014 Revised Selected Papers. LNCS, vol. 8597, pp. 177–186. Springer (2014)
23. Zymbler, M.L.: Best-match time series subsequence search on the intel many integrated core architecture. In: Morzy, T., Valduriez, P., Bellatreche, L. (eds.) Advances in Databases and Information Systems - 19th East European Conference, ADBIS 2015, Poitiers, France, September 8-11, 2015, Proceedings. LNCS, vol. 9282, pp. 275–286. Springer (2015)