ISSN No. 0976-5697

Volume 4, No. 9, July-August 2013

International Journal of Advanced Research in Computer Science

RESEARCH PAPER

Available Online at www.ijarcs.info

# Clustering algorithm optimized for Cell Broadband Engine Architecture

Ioan Ungurean
Department of Computers, Electronics and Automation
Stefan cel Mare University of Suceava Suceava, Romania
ioanu@eed.usv.ro

***Abstract:*** In this paper, we want to evaluate the performance of Cell B.E. processor which is based on the Cell Broadband Engine Architecture (CBEA). For this purpose, we chose a clustering algorithm that we have optimized for this architecture by efficiently harnessing the facilities provided. Performance of the Cell B.E. processor was evaluated by executing the algorithm using computation in single and double precision. In both cases, performance was evaluated with and without SIMDization. For single precision, we obtained a maximum speedup of 29.07 by activating 6 SPE processors without SIMDization and a speedup of 10.9 for 6 SPE processors with SIMDization. For double precision, we obtained a maximum speedup of 14.51 by activating 6 SPE processors without SIMDization and a speedup of 8.34 for 6 SPE processors with SIMDization.

***Keywords:*** Cell Broadband Engine Architecture, DMA, SIMD, speed-up, parallel computing

## I. INTRODUCTION

In order to meet the computing requirements which, in the last decade, are increasingly higher, there are developed computer systems that in addition to the central processor have acceleration units. An example of this is the Cell BE processor [1][2][3] from the PlayStation3 game consoles that has a core processor based on PowerPC architecture and eight specialized cores for intensive calculations. In order to use at maximum the facilities provided by these computer systems, the applications should be developed and optimized for these architectures.

An application developed for the execution on a normal processor can be compiled and executed on these systems but will not use all available computing facilities. For this reason, applications should be developed specifically for these architectures.

In this paper, we want to evaluate performance that can be achieved by execution on a clustering algorithm on Cell B.E. processor compared to the sequential version executed on normal processors. The remainder of this paper is organized as follows: Section 2 presents an overview of the Cell Broadband Engine architecture, Sections 3 present the algorithm that is optimized for Cell Broadband Engine, Sections 4 present the optimization and parallelization strategies used, Section 5 evaluates the performance achieved by execution of the algorithm on PlayStation 3, and the final conclusions are drawn in Section 6.

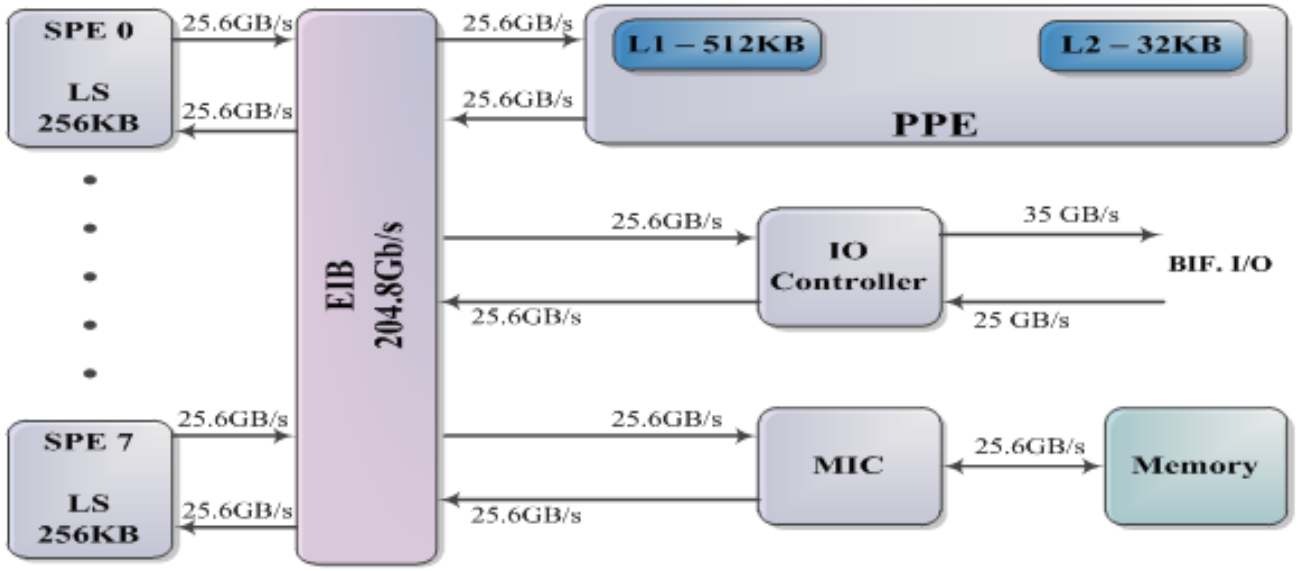


Figure 1. Cell Broadband Engine Architecture[3]

## II. CELL BROADBAND ENGINE

For PlayStation3 (PS3) game console, STI consortium consisting of Sony, Toshiba, and IBM developed the Cell Broadband Engine architecture (CBEA) [3] .

Cell BE processor was developed based on CBEA architecture, which is behind the PS3 game console launched in November 2006. CBEA architecture is based on 64bits PowerPC architecture to which eight cores for intensive calculations were added. Practically, it contains a PowerPC processor called PPE (Power PC Element), and 8-acceleration cores called SPE (Synergistic Processing Elements) [3].

All these processors are interconnected through a high speed bus called EIB (Element Interconnect Bus), as can be seen in Figure 1. SPE processors have not cache memory; they have just 256KB of local memory for code and data area. Application that is executed on the SPE cores must fit into this dimension in terms of requirements for code and data. In order to access the main memory, the SPE cores can use DMA transfers. We can say that the SPE cores are intermediate option between the conventional processors with cache memory and GPU processors that are used increasingly in HPC applications. SPE processors have RISC architecture with a SIMD extension with 128-bit registers.

PS3 game consoles allowed the installation of Linux operating systems and allowed programmers to develop applications optimized for CBEA architecture. In order to come in the aid of application's developers, IBM provides an SDK for CBEA architecture. This SDK contains libraries for efficient use of DMA transfers between main memory and SPE cores and the SIMD extension.

SPE cores of the Cell BE processor are not optimized for computations in double precision; reason for that IBM developed the PowerXCell 8i processor in which the SPE cores are optimized for computations in double precision. Based on these processors, IBM developed the QS20 and QS22 blade servers for HPC systems development. Processors with CBEA architecture entered powerful in the HPC field in 2009 when Roadrunner supercomputer was the first supercomputer that has exceeded 1PFlops for computing power [4].

This processor allows achieving a higher performance because it allows the SPE core to initiate a DMA transfer between the local memory and main memory, and during DMA transfer, it can perform other computing tasks. The most effective way to exploit this facility is to us double buffered for DMA transfers. In this method computing operations on data from local memory are performed in parallel with DMA transfers of data that will occur in calculations in the next step. It should be noted that the efficiency of this method depends very much on the algorithm that is optimized for CBEA architecture.

According to the programming guide of this processor, if the facilities provided by these processors are effective used, we can achieve a speedup of almost 100 related to the sequence variants. From the programmer's point of view, three important aspects should be followed to achieve maximum performance, namely: SPE cores must perform operations in parallel; each SPE must use SIMD in order to maximize the operations that they carry out, and double buffed method must be used in order to perform DMA transfer operations in parallel with computing operations.

## III. OVERVIEW OF THE ALGORITHM

From the pattern recognition field [5], we chose the k-medoids algorithm described in [6]. This algorithm requires knowledge of the number M of clusters in which the set of patterns will be classified. Initially, the medoids of the M clusters are represented by M random patterns from the dataset.

The remaining patterns are included in the appropriate clusters with the nearest medoids. Once all patterns are included in one of the clusters, for each class, the centroids are calculated (each cluster taken into account all patterns that have been assigned). The new medoids of the clusters are the patterns that are closest to the centroid. The assignment procedure of the patterns is resumed until the centroids determined during two consecutive iterations coincide [5].

The algorithm has as input the set of patterns, which will be classified (N patterns, each pattern having p features) and the number of clusters in which will be classified the input dataset. In order to avoid the computation, at each iteration, the distance between the medoids (which is patterns from input dataset) and the patterns a matrix of distances is computed for the set of input forms. Algorithm 1 shows the sequential variant of this method.

Algorithm 1. Sequential algorithm.

Initialize the dynamic medoids attached to those M clusters with M random patterns from the input data set.

Compute the matrix of distances between the N patterns from the dataset.

***Repeat***

**for** each pattern from the input dataset **do**

*) includes the pattern in the cluster for which the distance between pattern and medoid is minimal (determines the minimum from the matrix of distances)

■

**for** each cluster from the M clusters **do**

Calculate the centroid of the cluster.

Find the pattern that is the closest to the centroid.

Initialize the medoid with the new pattern found.

■

**Until** new medoids coincide with those from the previous step.

## IV. OVERVIEW OF THE ALGORITHM

The first step in the implementation of the algorithm for the CBEA processor is the distribution of computational tasks to the SPE cores. In order to split the calculations to SPE cores, the matrix of distance is divided equally by the number of the SPE processors on which the algorithm is executed. The follow up, we provide the method of dividing the matrix of distance to the noSPE processors. This problem is reduced to equally split the area of a triangle (above the main diagonal). For each processor SPE, we must determine the number of rows in the matrix associated and the offset from where these rows start.

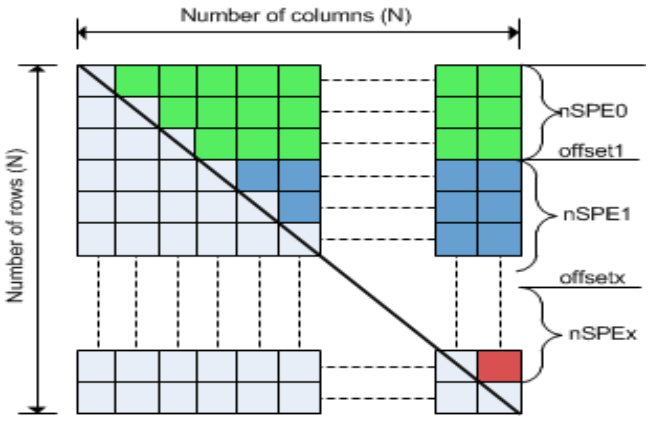


Figure 2. Splitting the matrix of distances to the SPE processors

It is considered that the matrix has N rows and N columns, so the area of the triangle above the main diagonal is (N*N)/2 (see Figure 2). Therefore the area associated to each SPE processor is (n*n)/(2*noSPE).

To find the number of rows associated to the first SPE processor (nSPE0) a quadratic equation must be solved:

$$\frac{(n-offset0+n-offset0-nSPE0)*nSPE0}{2}=\frac{n*n}{2*noSPE} \quad (1)$$

$$nSPE0^2-2*(n-offset0)*nSPE0+\frac{n^2}{noSPE}=0 \quad (2)$$

In the same way, the number of processors assigned to each PPE processor is determined by replacing the offset value offset0 with the value associated to each processor.

As redundant data, each SPE processor will know the dynamic medoids associated to the M clusters. The algorithm executed on the PPE processor is:

Algorithm 2. The algorithm executed by the PPE processor

Divide equally input dataset to SPE processors who are activated.

Initialize the medoids attached to the M clusters with M random patterns from the input dataset.

Divide the distance matrix to the SPE processors.

Send the command to the SPE cores in order to calculate the matrix of distances.

Wait for the SPE processor to complete the assigned jobs.

***Repeat:***

Send the command to each SPE processor in order to **d**etermine for each associated pattern the cluster to which belongs and to determine the local centroid for each cluster (the patterns are equally divided to the SPE processors).

Determine global centroids with local centroids provided by each SPE processor.

Send the command to the SPE processors in order to determine the each local pattern that is closest to each centroid associated to the M clusters (the patterns are equally divided to the SPE processors).

Wait for the SPE processor to complete the assigned jobs.

Determine each pattern that are the closer to the each centroid using the patterns received from the SPE processors. These patterns will be the new medoids.

**Until** new medoids coincide with those from the previous step.

The algorithm executed on each SPE processor is:

Algorithm 3. The algorithm executed by the SPE processors

***Repeat:***

Wait commands from the PPE processor.

**If** is the command for the computing of the matrix of distances.

Take the number of rows associated and offset for the matrix and distances.

Compute the associated part of the matrix (it uses DMA transfer in order to access the patterns and the matrix of distances from the main memory).

Signals the PPE processor that the associated operations were performed.

**else if** is the command to determine the cluster for each form and to compute the centroid for each cluster.

Take the number associated patterns and the offset. Determine for each associated pattern the cluster to which belongs.

Compute the centroid for the clusters using local patterns. Signals the processor PPE that were made related operations and send local centers of gravity calculated.

**else if** is the command for determination of patterns closer to the centroid.

Take the number associated patterns and the offset. Determine the local patterns which are closer to each centroid.

Signals the processor PPE that were made related operations and send the patterns which are closer to the centroids.

■

***Until** the command to end the application is received.*

## V. EXPERIMENTAL RESULTS

The proposed algorithm was executed on a PlayStation 3 game console that has a Cell BE processor with 6 SPE cores active. For a detailed analysis of the performance of Cell BE processor, the algorithm was executed using calculations in single and double precision with and without utilization of the SIMD library. For comparison, we performed a serial implementation of the algorithm which was executed on the PPE core of the Cell BE. Input data were generated randomly in order to execute different versions of the algorithm in terms of the number of patterns and the number of characteristics of the patterns. We generated 1000000 test forms with 1024 and 256 characteristics (p = 256 and p = 1024).

Figure 3 presents the results obtained for 1000000 patterns where we use single precision and M = 100. Figure 3 presents the speed-up obtained by activating 1, 2, 3, 4, 5, and 6 SPE processors. From this figure, we can see a significant difference between the algorithm implementation with utilization of the SIMD libraries and without utilization this library. In this case, we have achieved a maximum speed-up of 29.07 with SIMD library and six SPE processors and 10.9 with six SPE processors without SIMD library.

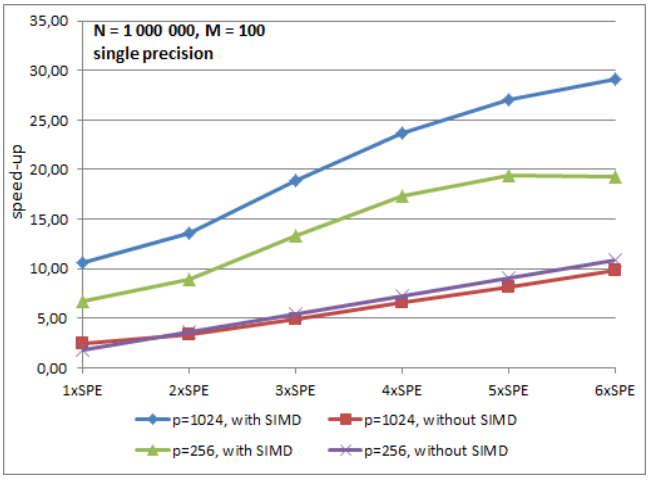


Figure 3. Speed-ups achieved for single precision computing

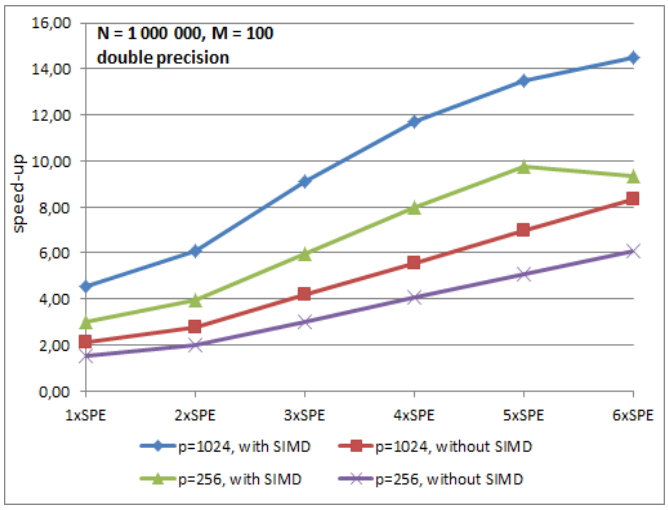


Figure 4. Speed-ups achieved for double precision computing

Figure 4 presents the results for 1000000 patterns where we use double precision and M = 100. In Figure 4 presents the speed-up obtained by activating 1, 2, 3, 4, 5, and 6 SPE processors. In this case, we have achieved a maximum speed-up of 14.51 with six SPE processors and SIMD library and 8.34 with six SPE processors and without SIMD library. It can be seen that in this case, speedup are lower than single precision because the volume of transferred data and computational effort is much higher than in the first case.

## VI. CONCLUSIONS

From the obtained results, we can see that the speed-up reaches high values in relation to the number of cores used. Furthermore, to achieve these results, the algorithm has been specifically optimized for the CBEA architecture. It is observed that the best results are obtained if we use SIMD library. This is occurred because the algorithm performs calculations using large vectors. If we chose an algorithm that does not make calculations on vectors the utilization of the SIMD library will not bring in a performance benefit.

## VII. REFERENCES

[1] Buttari, Alfredo, et al. "A rough guide to scientific computing on the PlayStation 3." ICL, University of Tennessee Knoxville, Tech. Rep. UT-CS-07-595 (2007).

[2] Ungurean, Ioan, and Nicoleta-Cristina Gaitan. "Speech analysis for medical predictions based on Cell Broadband Engine." Signal Processing Conference (EUSIPCO), 2012 Proceedings of the 20th European. IEEE, 2012.

[3] ****, An enhanced Cell Broadband Engine processor with improved double-precision floating-point performance: PowerXCell 8i Processor, IBM DeveloperWorks, May 2008.

[4] Barker, Kevin J., et al. "Entering the petaflop era: the architecture and performance of Roadrunner." Proceedings of the 2008 ACM/IEEE conference on Supercomputing. IEEE Press, 2008.

[5] Kumar, A., and N. Kannathasan. "A Survey on Data Mining and Pattern Recognition Techniques for Soil Data Mining." IJCSI International Journal of Computer Science Issues 8.3 (2011).

[6] Sergios Theodoridis and Konstantinos Koutroumbas. 2006. Pattern Recognition, Third Edition. Academic Press, Inc., Orlando, FL, USA.