Self-Organizing Maps on the Cell Broadband Engine Architecture

To cite this article: Sabine M McConnell 2010 J. Phys.: Conf. Ser. 256 012013

View the article online for updates and enhancements.

Related content
- Scalability of Self-organizing Maps on a GPU cluster using OpenCL and CUDA
  Sabine McConnell, Robert Sturgeon, Gregory Henry et al.
- Accelerated Synchrotron X-ray Diffraction Data Analysis on a Heterogeneous High Performance Computing System
  J Qin and M A Bauer
- CellPilot: Seamless communication within Cell BE and heterogeneous clusters
  N Girard, J Carter, W B Gardner et al.
Self-Organizing Maps on the Cell Broadband Engine Architecture

Sabine M. McConnell
Department of Computing and Information Systems, Trent University, 1600 West Bank Drive, Peterborough, ON, Canada, K9J 7B8
E-mail: sabinemccconnell@trentu.ca

Abstract. We present and evaluate novel parallel implementations of Self-Organizing Maps for the Cell Broadband Engine Architecture. Motivated by the interactive nature of the data-mining process, we evaluate the scalability of the implementations on two clusters using different network characteristics and incarnations (PS3™ console and PowerXCell 8i) of the architecture. Our implementations use varying combinations of the Power Processing Elements (PPEs) and Synergistic Processing Elements (SPEs) found in the Cell architecture. For a single processor, our implementation scaled well with the number of SPEs regardless of the incarnation. When combining multiple PS3™ consoles, the synchronization over the slower network resulted in poor speedups and demonstrated that the use of such a low-cost cluster may be severely restricted, even without the use of SPEs. When using multiple SPEs for the PowerXCell 8i cluster, the speedup grew linearly with increasing number of SPEs for a given number of processors, and linear up to a maximum with the number of processors for a given number of SPEs. Our implementation achieved a worst-case efficiency of 67% for the maximum number of processing elements involved in the computation, but consistently higher values for smaller numbers of processing elements with speedups of up to 70.

1. Introduction and Motivation
Jointly developed by Sony, Toshiba, and IBM, the CBEA has recently developed a foothold in Scientific and High Performance computing using both of its incarnations: the PowerXCell 8i and the Cell processor used in the PS3™ gaming console. The latter in particular has provided a low-cost environment capable of 150 GFlops (single-precision) at a cost of just under $300, which has led to the construction of clusters containing this architecture. Perhaps most notably, the US army has recently purchased 2 200 additional PS3™ units to extend a smaller existing cluster. On the other hand, the PowerXCell is used in IBM’s RoadRunner, which was the first general-purpose computer to break the Petaflop barrier. Example applications using the Cell processor for scientific computing include matrix optimization by Kurzak et al. [14], biomolecular simulation by Fabritiis [5], Fast Fourier Transform by Chow et al. [4], and Ray Tracing by Benthin et al. [2], with a major focus on the use of single units.

Despite the powerful potential of the CBEA that was demonstrated by a flurry of activities over the past five years, programming for this architecture has proven difficult because of its hybrid nature and the requirement for explicit memory management. One application area that sparked a substantial amount of interest in the Cell Broadband Engine is data mining, which is defined as the extraction of novel and useful information from large databases. One
particular learning technique that is frequently used for data mining is the concept of a Self-
Organizing Map, originally developed by Kohonen [13]. Recent applications of SOMs include the
construction of Parton Distribution Functions parameterizations [10], an analysis of spectra [3],
clustering of spatial-temporal precipitation data [11], and cluster analysis of spectrum sources
[17].

With the rapid growth of the data sets in recent years, the iterative nature of data mining
(combined with a multitude of techniques) has introduced a challenge that is unmatched by
typical batch-processing environments; there is a current need for platforms that allow interactive
data mining on the order of minutes or less, ideally at a low cost. The goal of this paper is to
provide and evaluate a set of implementations of one particular learning algorithm for such an
interactive and, for one of the incarnations of the architecture, low-cost computing environment.
In particular, we are interested in the scalability and processor utilization of our approaches for
two different clusters, using both incarnations of the CBEA in combination with two different
networks.

The remainder of this paper is organized as follows. In Section 2 we give an overview of
the relevant background material, including the architecture, the particular learning algorithm
whose implementations we discuss here, parallel versions of said algorithm, and previous work
connecting data-mining techniques to this particular architecture. In Section 3 we introduce the
parallel implementations of the Self-Organizing Maps. We present the results and discussions
in Section 4, followed by conclusions and future work in Section 5.

2. Background

2.1. Data Mining on the CBEA Architecture

The Cell Broadband Engine Architecture connects a 3.2 GHz hyper-threaded simplified PowerPC
(Power Processing Element, or PPE) with eight Synergistic Processing Elements (SPEs) via
a ring-like Element Interconnect Bus with a maximum theoretical bandwidth of 25.2 GB/s.
Able to achieve 25.6 GFlops (single precision), each of the single-threaded SPEs requires only
4 Watt, which initially made the Cell architecture an attractive alternative to contemporary
architectures. Each SPE contains a vector processor, a 256KB private memory area called the
local store, 128 128-bit registers, and a Memory Flow Controller. The architectural design focuses
on performance and bandwidth rather than latency and ease of programming (severely restricting
the portability of any implementation using the SPEs), and memory has to be managed explicitly
by the developer.

A number of data-mining techniques have been parallelized to date on a single Cell processor,
using up to six of the SPEs that are available to the application developer via a Software
Development Kit. For example k-means clustering and the k Nearest Neighbors algorithm are
implemented by Buehrer et al. [1] and k-means clustering, APriori for Association Mining, and a
Radial Basis Function Network by Duan and Strey [7]. Implementation of SOMs for a single Cell
processor are limited to a student project available for download at the IBM developerWorks
web site. However, this project did not implement double buffering, and was restricted to a
single SPE only. Extensions to clusters for an Artificial Neural Network and Support Vector
Machine are presented by Wyganowski [25] and McConnell et al. [18] for Decision Trees and
Artificial Neural Networks.

2.2. Self-Organizing Maps

The SOM, originally proposed by Kohonen [13], is a variant of an Artificial Neural Network which
shares characteristics with Vector Quantization algorithms. The map consists of a set of neurons
represented by weights and arranged in a typically 2-d structure. The neurons are repeatedly
presented with randomly drawn input samples, for which the closest neuron is determined.
The closest neuron, and its neighborhood as determined by a neighborhood function, are then
adjusted to move closer to the input sample, resulting in a non-linear mapping. Kohonen originally proposed two versions of the algorithm: sequential training and batch training. Both typically use the Euclidean distance to determine the closest neuron for a given input sample, and are trained over multiple iterations through the input data.

In the sequential training approach, the neurons in the map are updated after each input sample. After determination of the winning node, the updates are calculated based on the difference between the values of the winning node and that of the input sample according to the rule $m_i(t+1) = m_i(t) + \alpha h_{ci}(t)[x(t) - m_i(t)]$, where $m_i(t)$ denotes the $m$th neuron at iteration $t$, $\alpha$ the learning rate, $h_{ci}$ the neighborhood function, and $x$ the input vector. Both the learning rate and the neighborhood function, which is typically a Gaussian function, decrease over time.

The runtime of the SOM depends on the sizes of the map and the data set, but also on the characteristics of the data. Often, convergence of the algorithm occurs over a large number of iterations through the input data.

The batch training algorithm implemented here omits the learning rate and updates the neurons only once for each iteration, after all input data is presented. The new values of the neurons are calculated as weighted averages of the data assigned to the neurons. Furthermore, the neighborhood function is often reduced to $h_{ic} = \delta(i,c)$, so that only the winning neuron itself is updated.

### 2.3. Parallel implementations of SOMs

Several variants of SOMs have been parallelized, an introduction to and evaluation of several parallelization techniques is, for example, given by Kwiatowski et al. [15]. Parallelizations of the two original versions of the algorithm can be grouped by the level of parallelism they exploit. Network-partitioning techniques distribute parts of the feature map but require copies of the input data at each of the processing elements. Such partitioning results in limited scalability, not only because the number of nodes in a given map is typically small compared to the size of the data itself, but also because of the increased memory requirements. Furthermore, if the online algorithm is used, synchronization between elements is required after presentation of each input element in order to select the best matching unit.

In contrast, data-partitioning approaches distribute the input data but require copies of the weight vectors at each processing element. Strupl and Neruda [21] show the performance advantages of partitioning the input data. If the batch algorithm is used, then global synchronization is necessary only after each input sample is presented, but still required at each iteration.

Using Kohonen’s SOM as a starting point for four new algorithms, Huntsberger and Ajjimarangsee [12] present implementations on a NCUBE/10 hypercube, while Filippi and Lawson discuss the implementation of the SOM on the SMART neurocomputer, which was built as a distributed memory systems using a linear processor array [8]. Filippi and Lawson investigated both a round-robin assignment of the input data and map, and network partitioning across the elements in the processor array, for 10 000 iterations and maps containing 100 and 900 nodes. Openshaw and Turton demonstrate superlinear speedup on a Cray T3D [19] for an 8x8 map and a data set containing 145 791 samples. Finally, Schikuta and Weidman achieve near-linear speedup for four different implementations (distributed or replicated weights and data) on a 16-node Intel iPSC/860 hypercube [20]. Motivated in part by the advantages of larger map sizes, Silva and Marques [22] implement a combination of both partitioning approaches for a 3-cluster machine built on Fast Ethernet, achieving a speedup of 1.27 for 30 162 records with 15 attributes. Yang and Ahuja [23] report a speedup of 3.14 for a cluster of four processors. Lawrence et al. [16] evaluate the performance of a parallel batch-partitioning SOM (using different implementations of network and data partitioning) on a 16-node SP2, for map sizes of up to 64 nodes and three data sets. Finally, in the context of preserving data privacy, Gorgonio
and Costa[9] implement map partitions used with local data, combined with fusion of the partial results at a central site. To our best knowledge, no implementations exist for Self-Organizing Maps on a CBEA cluster.

3. Algorithms and Experimental Setup
In order to evaluate the opportunities and challenges for the performance of the Self-Organizing Maps on the CBEA, we implemented the following SOM variants. All versions use a data-partitioning approach which requires copies of the weights to be distributed to each of the processing elements, rather than copies of the data which would limit scalability severely for larger data sets, particularly in view of the small local store for the SPEs. In all algorithms, updates to the weights are computed and stored for the winning node for each input datum, and used to update the map once per iteration after complete passes through the input data.

The algorithms below differ with respect to whether single or multiple Power Processing Elements and Synergistic Processing Elements are used. Beginning with a sequential version for comparison, we then included the Synergistic Processing Elements of a single Cell processor, followed by a parallel version using only the PPEs. Finally, we combined multiple PPEs and SPEs, which required two levels of synchronization. Where appropriate, we used double buffering to overlap computation and communication.

3.1. Algorithm 1: Sequential version using the PPE only
This version is implemented as baseline for comparison. Algorithm 1 shows a high-level view of this algorithm, which is identical to Kohonen’s batch learning approach.

Algorithm 1 Batch-Learning Self-Organizing Map
1: Initialize the map
2: while Termination criteria not satisfied do
3:   for all Input data do
4:     Determine the winning node for the input datum
5:     Calculate and store update to weights representing winning node
6:   end for
7: end while

3.2. Algorithm 2: Data parallel version using multiple PPEs
The data set is scattered over the different PPEs which communicate with each other via MPI. Initially, the root scatters the input data to the other processor and broadcasts the initial weights. At each iteration, processors calculate updates which are then reduced at the root. The root then updates and broadcasts the weight matrix at each iteration. Disregarding the initial scatter operation that only occurs once and therefore is amortized over an increasing number of iterations, the need to compute the final updates at each iteration through the root represents the main serial component in this implementation. Again, this component will increase with the number of processing elements and the size of the weight map, but not with the size of the input data, and synchronization occurs via reductions using MPI.

3.3. Algorithm 3: Data-parallel version using multiple SPEs of a single Cell processor
We distribute the distance calculation to the SPEs, while the PPE is responsible for calculation of the overall weight updates at each iteration. Each SPE receives a copy of the weights, but is responsible for calculating winners and updates for parts of the input data. At each iteration
Algorithm 2 Distributing the distance calculation over multiple PPEs

1: Initialize the map at the root
2: Broadcast map to the PPEs
3: Scatter data to PPEs
4: while Termination criteria not satisfied do
5:   for all PPEs do
6:     for all Assigned data do
7:       Determine the winning node for the input datum
8:       Calculate and store updates to weights representing winning node
9:     end for
10:   end for
11: Reduce map updates at PPE
12: Broadcast updated map to each PPE
13: end while

through the input data, the PPE has to wait for all SPEs to complete the pass through the
assigned data, followed by an idle period for the SPEs while they wait for the PPE to compute
the overall updates for the weights. The need to compute the final updates at each iteration
through the PPE constitutes the main serial component in this implementation, which will
increase with the number of SPEs and the size of the weight map, but not with the size of
the input data. In this case, synchronization between PPE and SPEs is implemented using
mailboxes.

Algorithm 3 Distributing the distance calculation over multiple SPEs

1: Initialize the map at the PPE
2: Send copies of the map to each SPE
3: Send starting address of assigned data to each SPEs
4: while Termination criteria not satisfied do
5:   for all SPEs do
6:     for all Assigned data do
7:       Determine the winning node for the input datum
8:       Calculate and store updates to weights representing winning node
9:     end for
10:   end for
11: Send updates for map to PPE
12: end for
13: Calculate map updates at PPE
14: Send copies of the map to each SPE
15: end while

3.4. Algorithm 4: Data parallel version using multiple PPEs and SPEs
With the goal of exploring the limitations to using the full computational power of the
Synergistic Processing Elements in each of the processors, we combine the previous two
variants. Synchronization now requires two levels: communication between SPEs and PPE to
communicate partial updates calculated at the SPEs to the PPE which will then calculate final
updates for the data allocated to a single CBEA. This is then followed by global synchronization
using MPI in order to communicate the updates to the root, at which the updates to the map
will be reduced. After the updates are applied to the map by the root, the new weights are
broadcast to each processor. This process is repeated for each iteration through the input data.

**Algorithm 4** Distributing the distance calculation over multiple PPEs and SPEs

1: Initialize the map at the root
2: Broadcast map to the PPEs
3: Send copies of the map to each SPE
4: Scatter data to PPEs
5: Send starting address of assigned data to each SPEs
6: while Termination criteria not satisfied do
7:   for all SPEs do
8:     for all Assigned data do
9:       Determine the winning node for the input datum
10:      Calculate and store updates to weights representing winning node
11:     end for
12:   end for
13:   Send updates for map to PPE
14: end while
15: Calculate updates at the PPEs
16: Reduce map updates at root and calculate new weights
17: Broadcast updated map to each PPE
18: Send copies of the map to each SPE

3.5. Experimental Setup

We investigated the scalability of our approaches on two fundamentally different CBEA clusters: a small local cluster, and a 44-node SHARCNET system. The local cluster consists of five PS3\textsuperscript{TM} consoles, one of which is used as a login node only. The consoles are connected via a Fast Ethernet Switch (EE2400-SS), resulting in a dedicated 100Mb/s Ethernet. The PS3\textsuperscript{TM} cluster uses OpenMPI as the message-passing interface, combined with Torque as the resource manager, and Fedora 9. The SHARCNET cluster prickly consists of a set of varying nodes, which includes 8 nodes of 2 PowerXCell 8i processors each, in addition to one dual-socket dual-core Xeon processor, and three dual-socket quad-core Xeon processors. Running RedHat Enterprise Linux 5.2, these nodes are connected via Gigabit Ethernet.

We developed the applications using the IBM CELL Broadband Engine Software Development Kit (SDK) for Multicore Acceleration, Version 3.0. For both clusters, we evaluate our implementations with respect to the size of both the (numerical-valued) data set and the map, the number of iterations (which is generally on the order of a few hundred for real datasets) through the data, the number of Cell processors, and the number of Synergistic Processing Elements. Typically, the data used for data-mining is a by-product of some other collection process, and therefore often quite noisy and biased. For most data-mining algorithms, single-precision computation, as implemented here, is therefore sufficient. All results shown below are averaged over five runs.

4. Results and Discussion

We evaluated each of the algorithms introduced above on both clusters. The results are organized as follows. First, we present the evaluation for the use of the PPEs only in Section 4.1, while disregarding the computational power of the SPEs in order to determine the effects of the network for both clusters. Section 4.2 then introduces the evaluation of the implementation
Figure 1. Run times (in seconds) for the parallel algorithm using varying numbers of PPEs, no SPEs. Shown are the results for the PS3™ cluster for 1 to 4 processors (+), and 1-16 PPEs on the PowerXCell 8i cluster (*), for 500 iterations, 163,840 input values, 4 attributes, and a map of size 16x16.

for a single Cell processor but multiple SPEs. Finally, we discuss the results achieved when combining multiple Cell processors with multiple SPEs in Section 4.3.

4.1. Scalability without Synergistic Processing Elements

In order to evaluate the effects of the networks, we implemented a version of the SOM that carried out all computation on the PPEs themselves, without involvement of the SPEs (Algorithms 1 and 2). Figure 1 shows the run times for various numbers of processors on both clusters. From the data shown in the figure we see that the speedup (sequential execution divided by the parallel execution time) scaled linearly with the number of processors used for the SHARCNET cluster. For this cluster, the resulting efficiency (speedup divided by the number of processors used) was close to 1 throughout, for example, 0.99 for 16 processors. In contrast, for the smaller cluster the speedup achieved for 2 processors was close to 1.5, but then dropped when the number of processors was increased. In this case, the run time increased and efficiency dropped by half when going from 2 to 4 processors, which clearly shows the restrictions associated with our low-cost cluster.

Reducing the number of iterations or doubling the sample size while holding the map size constant resulted in the same trend for the speedup and efficiency, for both clusters, with almost identical values. For example, our implementations achieved an efficiency of 0.98 for the SHARCNET cluster for 100 iterations, which indicates that the startup cost is amortized early. With respect to a reduction in map size, the efficiency of the implementation on the SHARCNET cluster dropped slightly to .94 for 16 processors. Overall, the implementation scaled well with increasing map sizes and iterations for the SHARCNET cluster, but not for the console cluster. The poor scalability of the implementations on the local cluster was not only rooted in the slower network, but also in the substantial additional overhead to access the network as implemented in the consoles themselves, and shows the limitations of low-cost clusters for applications requiring a fair amount of synchronization.

4.2. Scalability with Synergistic Processing Elements for Single Cell Processor

The performance of both incarnations of the Cell processor was then evaluated using only one PPE but multiple SPEs (Algorithm 3), with the goal of determining whether any performance
gains are available from the PowerXCell 8i processor (double-precision performance set aside, which is not an issue for this particular data-mining technique). Figure 2 shows the results for increasing number of SPEs for four different combinations of map sizes and sample sizes, for both incarnations of the Cell processor.

![Graphs showing runtime for different SPE combinations](image)

**Figure 2.** Run times (in seconds) for both incarnations of a single Cell Processor, using multiple SPEs, for 500 iterations and 4 attributes. Shown are the time for the PS3\textsuperscript{TM}(+), and the PowerXCell 8i (□).

The results for both incarnations are almost identical, indicating that the subtle differences in architecture for the two different incarnations do not have an effect for our implementations when adding the SPEs. Overall, the experimentally determined serial fraction is small, and the efficiency is close to 1 with a minimum of 91%. A comparison of the values for single processors depicted in Figures 1 and 2d) shows that the speedup achieved for the use of a single SPE (1911s for console) as opposed to the PPE only (7516s for console) is roughly 4 for both incarnations (sample size 163,840 samples, 4 attributes, 16x16 map, 500 iterations).

Table 1 shows the speedup for the console, for varying map sizes and numbers of SPEs, when increasing the number of SPEs and relative to the use of a single SPE for the Euclidean distance calculation. In general, the achieved speedup tends to be slightly larger for larger map sizes and data sets, for which the initial overhead of mailbox communication with the SPEs (starting addresses of data and weights for DMA transfers, plus parameter settings) is better amortized. Overall, the speedup scales linear with the number of SPEs when using a single Cell processor.

4.3. Scalability with Synergistic Processing Elements for Multiple Cell Processors
Following the implementations utilizing either multiple PPEs without SPEs, or multiple SPEs of a single Cell processor, we then combined both approaches. Our evaluation included the scalability for an increasing number of processors for a given number of SPEs, and the scalability
Table 1. Speedup for Increasing Map Size and Numbers of SPEs (4 attributes, 500 iterations, 163 840 Samples)

<table>
<thead>
<tr>
<th>Map</th>
<th>2 SPEs</th>
<th>3 SPEs</th>
<th>4 SPEs</th>
<th>5 SPEs</th>
<th>6 SPEs</th>
</tr>
</thead>
<tbody>
<tr>
<td>4x4</td>
<td>1.96</td>
<td>2.95</td>
<td>3.75</td>
<td>4.56</td>
<td>5.52</td>
</tr>
<tr>
<td>16x16</td>
<td>2.00</td>
<td>3.07</td>
<td>3.98</td>
<td>4.97</td>
<td>6.11</td>
</tr>
<tr>
<td>32x32</td>
<td>2.00</td>
<td>3.07</td>
<td>4.00</td>
<td>4.99</td>
<td>6.14</td>
</tr>
</tbody>
</table>

for an increasing number of SPEs for a given number of processors. Figure 3 shows the run times for both clusters as the numbers of SPEs and processors are increased, initially for up to four processors only for comparison purposes (run times and speedup for larger numbers of processors for the SHARCNET cluster will be discussed below). As can be seen from the data shown in the

![Figure 3](image_url)

Figure 3. Run times (seconds) for multiple PPEs and SPEs on the local cluster (left) and for up to 4 processors on the SHARCNET cluster (right). Shown are the results for one (+), two (*), three (x), and four processors (□), for 163 840 samples, 16x16 map, and 500 iterations.

figure, the run times for the local cluster only level off for more than two processors, regardless of how many SPEs are used, which is caused by the slow network access and slower network. This effect is also illustrated in Figure 4, which shows the speedups for increasing number of processors and various numbers of SPE on the local cluster only.

The speedup for 1 processor scales linearly with the number of SPEs. For two processors, the speedup drops when adding the fourth PPE, but scales linear up to this point. Performance for three and four processors is poor, with no improvement to the run times for 4 processors when adding SPEs.

Figure 5 shows the run times for increasing number of samples on the local cluster, and illustrates the reason for the poor performance of the local cluster. The SPEs tend to spend a large fraction of the code in synchronization or waiting for data: 59%, 55%, and 54% are spent in the vectorized portion of the code, for 163 840, 327 680, and 491 520 samples, respectively.

In conclusion, the local cluster shows severely limited scalability for more than 2 SPEs and processors because of the poor utilization of the SPEs.

In contrast to the effect observed on the local cluster, the fraction of vectorized time compared
Figure 4. Speedups for the PS3\textsuperscript{TM} cluster, using multiple PPEs and SPEs (163,840 samples, 16x16 map, 500 iterations). Shown are the results for one (+), two (∗), four (x), and six (□) SPEs.

Figure 5. Run times on the PS3\textsuperscript{TM} cluster (four PPEs, six SPEs, 163,840 to 491,520 samples, four attributes, 16x16 map, and 500 iterations). Shown are the total time for the PPE (+), the total time for the SPE (∗), and the time for the vectorized portion of the code for the SPE (x).

to overall runtime does not drop as dramatically for the SHARCNET cluster: 98% for one processor, which steadily decreases to 82% for 16 processor. Figure 6 and Table 2 present these results from slightly different perspectives.

Figure 6 shows the speedups for increasing number of processors and various numbers of SPEs on the SHARCNET cluster. In this case, the speedup refers to the decrease in runtime as the number of processors is increased and the number of SPEs is held constant. The data shown in the figure indicates that for a for smaller number of processors, the speedup is roughly linear with the number of SPEs. For any given number of utilized SPEs, the speedup levels off for some number of processors. The higher the number of SPEs, the earlier the peak speedup is reached (for example, the maximum speedup occurs for 7 processors for 6 SPEs, but for 13 processors for one SPE). The maximum speedup of 18 occurred for two SPEs and 14 processors.

Table 2 shows the speedups achieved on the SHARCNET cluster, for varying numbers of SPEs and Cell processors, but now with reference to the total number of processing elements. Starting with a single PPE and SPE, linear speedup would result in a value of 6 in the table when
Figure 6. Speedup for implementation using multiple PPEs and SPEs on the PowerXCell 8i cluster (163,840 samples, four attributes, 16x16 map, 500 iterations). Shown are the results for one (+), two (*), four (x), and six (□) SPEs.

fixing the number of PPE (across a row), a value of 16 when fixing the number of SPEs (down a column), and a value of 96 for the bottom right entry using 6 SPEs in each of the 16 PPEs. The data shown in the table indicates that, for a given number of processors, the speedup grows linear with the number of SPEs. The maximum speedup is reached earlier for larger numbers of SPEs, and roughly equal numbers of Cell processors and SPEs achieve the highest efficiencies close to 1. The maximum speedup is 71%, in the case of 7 processors and 6 SPEs, the minimum efficiency of 67% is reached for the maximum number of processing elements (96 SPEs in total). While the slow network characteristics restricted the scalability of our implementation on the PS3™ cluster, we were able to use up to 60 processing elements with an efficiency close to 1 on the SHARCNET cluster.

5. Conclusions and Future Work
We evaluated the scalability of various SOM implementations on two clusters employing different networks and incarnations of the Cell processor. The performance of the implementation for a single Cell processor using multiple SPEs was comparable for both incarnations, and scaled well with respect to the size of the data set, the number of iterations and the size of the map. The slower network and access times for the PS3™ cluster resulted in a lower fraction of time spent in vectorized code for the SPEs when multiple Cell processors were used, even without the use of the SPEs. This demonstrates that the use of such a low-cost cluster may be severely restricted. When using multiple Cell processors in combination with multiple SPEs for the PowerXCell 8i cluster, the speedup grows linearly with increasing number of SPEs for a given number of processors. For a given number of SPEs, the speedup initially grows linear with the number of...
Table 2. Speedup for Increasing Numbers of Processors and SPEs (16x16 map, 4 attributes, 500 iterations, 163,840 Samples)

<table>
<thead>
<tr>
<th>Processors</th>
<th>1 SPE</th>
<th>2 SPEs</th>
<th>4 SPEs</th>
<th>6 SPEs</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1.99</td>
<td>3.99</td>
<td>7.96</td>
<td>12.88</td>
</tr>
<tr>
<td>3</td>
<td>3.07</td>
<td>6.11</td>
<td>13.15</td>
<td>19.18</td>
</tr>
<tr>
<td>4</td>
<td>3.99</td>
<td>7.92</td>
<td>15.76</td>
<td>25.37</td>
</tr>
<tr>
<td>5</td>
<td>4.98</td>
<td>9.88</td>
<td>19.59</td>
<td>37.37</td>
</tr>
<tr>
<td>6</td>
<td>6.12</td>
<td>13.11</td>
<td>25.89</td>
<td>37.46</td>
</tr>
<tr>
<td>7</td>
<td>7.22</td>
<td>15.64</td>
<td>38.10</td>
<td>71.14</td>
</tr>
<tr>
<td>8</td>
<td>7.82</td>
<td>15.41</td>
<td>36.38</td>
<td>65.12</td>
</tr>
<tr>
<td>9</td>
<td>9.74</td>
<td>19.04</td>
<td>36.38</td>
<td>65.25</td>
</tr>
<tr>
<td>10</td>
<td>9.75</td>
<td>19.08</td>
<td>36.26</td>
<td>64.82</td>
</tr>
<tr>
<td>11</td>
<td>11.10</td>
<td>24.88</td>
<td>66.60</td>
<td>64.92</td>
</tr>
<tr>
<td>12</td>
<td>12.87</td>
<td>25.02</td>
<td>66.34</td>
<td>65.20</td>
</tr>
<tr>
<td>13</td>
<td>12.82</td>
<td>24.98</td>
<td>66.48</td>
<td>65.12</td>
</tr>
<tr>
<td>14</td>
<td>15.36</td>
<td>36.33</td>
<td>66.59</td>
<td>65.15</td>
</tr>
<tr>
<td>15</td>
<td>15.30</td>
<td>36.32</td>
<td>66.04</td>
<td>65.10</td>
</tr>
<tr>
<td>16</td>
<td>15.36</td>
<td>36.35</td>
<td>66.24</td>
<td>65.20</td>
</tr>
</tbody>
</table>

processors, but then levels off.

Our implementation achieves an efficiency of 67% for the maximum number of processing elements involved in the computation (16 processors, with 6 SPEs), but consistently close to an efficiency of 1 for smaller numbers of PPEs and roughly equal numbers of PPEs and SPEs. Despite the current uncertainty of the future of the Cell architecture, it is likely to survive in some scaled-down version. Therefore, it might be worthwhile to consider the implementation of additional techniques as long as the clusters are available, given our successful implementation on a single Cell processor, and the scalability of the approach on the SHARCNET cluster.

Acknowledgments
This work was made possible by a Fellowship Award from the Shared Hierarchical Academic Research Computing Network (SHARCNET:www.sharcnet.ca).


