Nagy Z, Kék L, Kincses Z, Kiss A, Szolgay P Toward exploitation of cell multi-processor array in time-consuming applications by using CNN model INTERNATIONAL JOURNAL OF CIRCUIT THEORY AND APPLICATIONS 36:(5-6) pp. 605-622. (2008)

DOI: 10.1002/cta.508

Toward Exploitation of Cell Multi-processor Array in Time-Consuming Applications by Using CNN Model

Zoltán Nagy, László Kék[+], Zoltán Kincses [++], András Kiss[+], Péter Szolgay[+]

Computer and Automation Research Institute, Hungarian Academy of Sciences, Budapest, Hungary

+Also with Dept. Information Technology, Pázmány Péter Catholic University, Budapest, Hungary

++ Dept. Image Processing and Neurocomputing, University of Pannonia, Veszprém, Hungary

***Abstract*—Array computers can be useful in the solution of numerical spatiotemporal problems such as the state equation of the CNN or partial differential equations (PDE). IBM has recently introduced the Cell Broadband Engine (Cell BE) Architecture, which contains 8 identical vector processors in an array structure. In this paper the implementation of CNN simulation kernel on the Cell BE is described. The simulation kernel is optimized according to the special requirements of the Cell BE and can implement linear as well as nonlinear (piecewise linear) templates. In addition, the 2-D Princeton Ocean Model is implemented on the Cell BE. The area/speed/power tradeoffs of our solution and different hardware implementations are also compared.**

I. Introduction

Performance of the general purpose computing systems is usually improved by increasing the clock frequency and adding more processor cores. However, to achieve very high operating frequency very deep pipeline is required, which cannot be utilized in every clock cycle due to data and control dependencies. If an array of processor cores is used, the memory system should handle several concurrent memory accesses, which requires large cache memory and complex control logic. In addition, applications rarely occupy all of the available integer and floating point execution units fully.

Array processing to increase the computing power by using parallel computation can be a good candidate to solve architectural problems (distribution of control signals on a chip). Huge computing power is a requirement if we want to solve complex tasks and optimize to dissipated power and area at the same time. There are a number of different implementations of array processors commercially available. The CSX600 accelerator chip from Clearspeed Inc. [1] contains two main processor elements, the Mono and the Poly execution units. The Mono execution unit is a conventional RISC processor responsible for program flow control and thread switching. The Poly execution unit is a 1-D array of 96 execution units, which work on a SIMD fashion. Each execution unit contains a 64bit floating point unit, integer ALU, 16bit MAC (Multiply Accumulate) unit, an I/O unit, a small register file and local SRAM memory. Although the architecture runs only on 250MHz clock frequency the computing performance of the array may reach 25GFlops. The Mathstar FPOA (Field Programmable Object Array) architecture [2] contains different types of 16bit execution units, called

This work was partially supported by the IBM Hungary under grant number 68/K/2007.

Silicon Objects, which are arranged on a 2-D grid. The connection between the Silicon Objects is provided by a programmable routing architecture. The three main object types are the 16bit integer ALU, 16bit MAC and 64 word register file. Additionally, the architecture contains 19Kb on-chip SRAM memories. The Silicon objects work independently on a MIMD (Multiple Instruction Multiple Data) fashion. FPOA designs are created in a graphical design environment or by using MathStar's Silicon Object Assembly Language. The Tilera Tile64 architecture [3] is a regular array of general purpose processors, called Tile Processors, arranged on an 8x8 grid. Each Tile Processor is 3-way VLIW (Very Long Instruction Word) architecture and has a local L1, L2 cache and a switch for the on-chip network. The L2 cache is visible for all processors forming a large coherent shared L3 cache. The clock frequency of the architecture is in the 600-900MHz range providing 192GOps peak computing power. The processors work with 32bit data words but floating point support is not described in the datasheets.

In this work we have concentrated on topographic IBM Cell heterogeneous array processor architecture mainly because its development system is open source, and we intended to compare the results to our previous FPGA based implementations. It is exploited here in solving complex, time consuming problems.

Considering the structure of the paper, in Section II, the Cell processor architecture is introduced. Section III describes the basic CNN model implemented on Cell BE. The details of implementation of the CNN model on Cell BE are shown in Section IV. Then, we investigate the 2-D Princeton Ocean Model in Section V. Finally, conclusions are drawn in Section VI.

II. Cell Processor Architecture

A. *Cell Processor Chip*

The Cell Broadband Engine Architecture (CBEA) [6] is designed to achieve high computing performance with better area/performance and power/performance ratios than the conventional multi-core architectures. The CBEA defines a heterogeneous multi-processor architecture where general purpose processors called Power Processor Elements (PPE) and SIMD[1] processors called Synergistic Processor Elements (SPE) are connected via a high speed on-chip coherent bus called Element Interconnect Bus (EIB). The CBEA architecture is flexible and the ratio of the different elements can be defined according to the requirements of the different applications. The first implementation of the CBEA is the Cell Broadband Engine (Cell BE or informally Cell) designed for the Sony Playstation 3 game console, and it contains 1 PPE and 8 SPEs. The block diagram of the Cell is shown in Figure 1.

The PPE is a conventional dual-threaded 64bit PowerPC processor which can run existing operating systems without modification and can control the operation of the SPEs. To simplify processor design and achieve higher clock speed instruction reordering is not supported by the PPE. The EIB is not a bus as suggested by its name but a ring network which contains 4 unidirectional rings where two rings run counter to the direction of the other two. The dual-channel Rambus XDR memory interface provides very high 25.6GB/s memory bandwidth. I/O devices can be accessed via two Rambus FlexIO interfaces where one of them (the Broadband Interface (BIF)) is coherent and makes it possible to connect two Cell processors directly.

[1] SIMD – Single Instruction Multiple Data

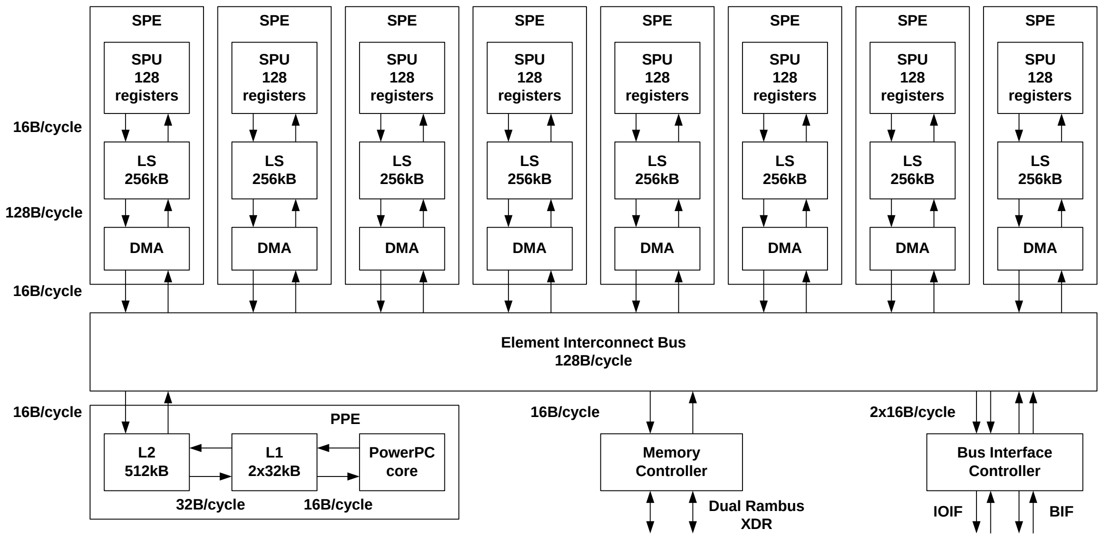

Figure 1. Block diagram of the Cell processor

The SPEs are SIMD only processors which are designed to handle streaming data. Therefore they do not perform well in general purpose applications and cannot run operating systems. Block diagram of the SPE is shown in Figure 2.

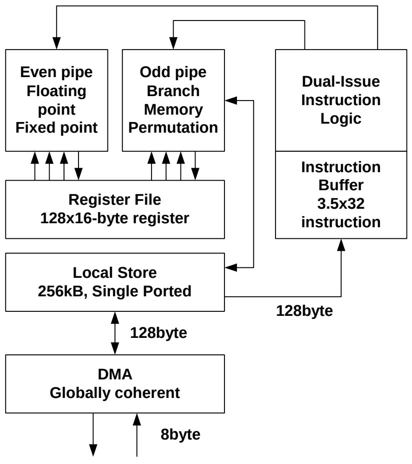
Even pipe Floating point Fixed point
Odd pipe Branch Memory Permutation
Dual-Issue Instruction Logic
Instruction Buffer 3.5x32 instruction
Register File 128x16-byte register
Local Store 256kB, Single Ported
128byte
128byte
DMA Globally coherent
8byte

Figure 2. Block diagram of the Synergistic Processor Element

The SPE has two execution pipelines: the even pipeline is used to execute floating point and integer instructions while the odd pipeline is responsible for the execution of branch, memory and permute instructions. Instructions for the even and odd pipeline can be issued in parallel. Similarly to the PPE the SPEs are also in-order processors. Data for the instructions are provided by the very large 128 element register file where each register is 16byte wide. Therefore SIMD instructions of the SPE work on 16byte-wide vectors, for example four single precision floating point numbers or eight 16bit integers. The register file has 6 read and 2 write ports to provide data for the two pipelines. The SPEs can only address their local 256kB SRAM memory but they can access the main memory of the system by DMA instructions. The Local Store is 128byte wide for the DMA and instruction fetch unit, while the Memory unit can address data on 16byte boundaries by using a buffer register. 16byte data words arriving from the EIB are collected by the DMA engine and written to the memory in one cycle. The DMA engines can handle up to 16 concurrent DMA operations where the size of each DMA operation can be 16kB. The DMA engine is part of the globally coherent memory address space but we must note that the local store of the SPE is not coherent.

B. *Cell Blade Systems*

Cell blade systems are built up from two Cell processor chips interconnected with a broadband interface. They offer extreme performance to accelerate compute-intensive tasks. The IBM Blade Center QS20 (see Figure 3.) is equipped with two Cell processor chips, Gigabit Ethernet, and 4x InfiniBand I/O capability. Its computing power is ~400GFLOPS peak. Further technical details are as follows:

- Dual 3.2GHz Cell BE Processor Configuration
- 1GB XDRAM (512MB per processor)
- Blade-mounted 40GB IDE HDD
- Dual Gigabit Ethernet controllers
- Double-wide blade (uses 2 BladeCenter slots)

Several QS20 may be interconnected in a Blade Center house with max. ~2.8TFLOPS peak computing power. It can be reached by utilizing max. 7 Blades per chassis.

The second generation blade system is the IBM Blade Center QS21 providing extraordinary computing density – up to 6.4 TFLOPS in a single Blade Center house.

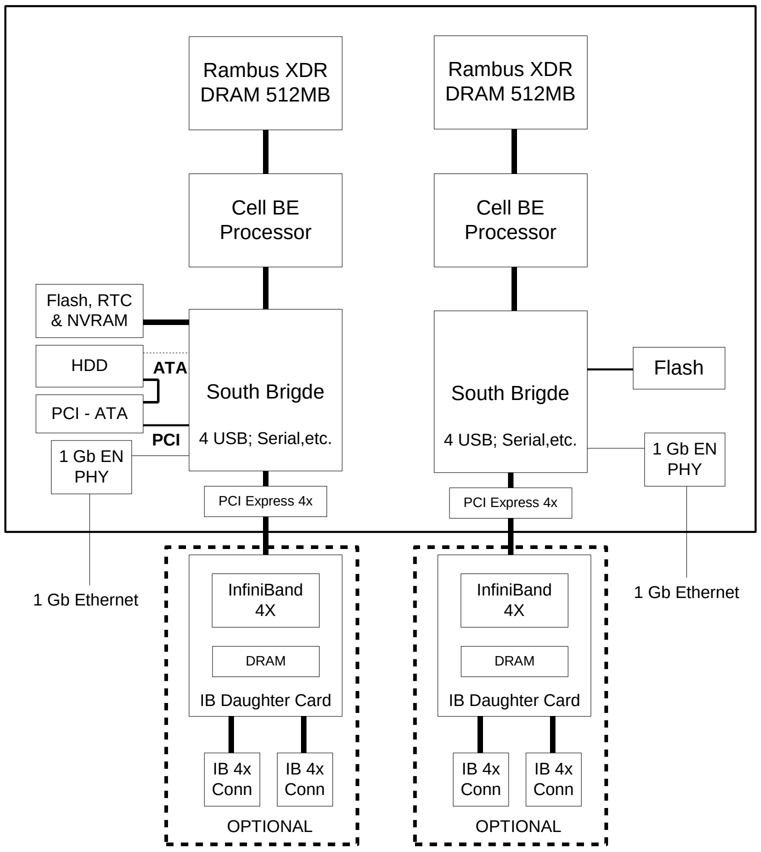

Figure 3. IBM Blade Center QS20 architecture

III. CNN MODEL

Our primary goal is to get an efficient CNN [4] implementation on the Cell architecture. Consider the CNN model and its hardware effective discretization in time.

A. *Linear templates*

The state equation of the original Chua-Yang model [4] is as follows:

$$\dot{x}_{ij}(t) = -x_{ij} + \sum_{C(kl)\in N_r(i,j)} \mathbf{A}_{ij,kl} y_{kl}(t) + \sum_{C(kl)\in N_r(i,j)} \mathbf{B}_{ij,kl} u_{kl} + z_{ij} \quad (1)$$
,

where u_{kl}, x_{ij}, and y_{kl} are the input, the state, and the output variables. **A** and **B** are the feedback and feed-forward templates, and z_{ij} is the bias term. $N_r(i,j)$ is the set of neighboring cells of the (i,j)th cell. The discretized form of the original state equation (1) is derived by using the forward Euler form. It is as follows:

$$x_{ij}(n+1) = (1-h)x_{ij}(n) + \\ + h\left(\sum_{C(kl)\in N_r(i,j)} \mathbf{A}_{ij,kl} y_{kl}(n) + \sum_{C(kl)\in N_r(i,j)} \mathbf{B}_{ij,kl} u_{kl} + z_{ij} \right) \quad (2)$$

In order to simplify computation variables are eliminated as far as possible. First of all, the Chua-Yang model is changed to the Full Signal Range (FSR) [7] model. Here the state and the output of the CNN are equal. In cases when the state is about to go to saturation, the state variable is simply truncated. In this way the absolute value of the state variable cannot exceed +1. The discretized version of the CNN state equation with FSR model is as follows:

$$x_{ij}(n+1) = \begin{cases} 1 & if \;\; v_{ij}(n) > 1 \\ v_{ij}(k) & if \; \left| v_{ij}(n) \right| \le 1 \\ -1 & if \; v_{ij}(n) < -1 \end{cases}$$

$$v_{ij}(n) = (1-h)x_{ij}(n) + \\ + h\left(\sum_{C(kl)\in N_r(i,j)} \mathbf{A}_{ij,kl} x_{kl}(n) + \sum_{C(kl)\in N_r(i,j)} \mathbf{B}_{ij,kl} u_{kl} + z_{ij} \right) \quad (3)$$

Now the x and y variables are combined by introducing a truncation, which is simple in the digital world from computational aspect. In addition, the h and (1-h) terms are included into the **A** and **B** template matrices resulting templates $\hat{A}, \hat{B}$.

By using these modified template matrices, the iteration scheme is simplified to a 3x3 convolution plus an extra addition:

$$v_{ij}(n+1) = \sum_{C(kl)\in N_r(i,j)} \hat{\mathbf{A}}_{ij,kl} x_{kl}(n) + g_{ij} \quad (4.1)$$

$$g_{ij} = \sum_{C(kl)\in N_r(i,j)} \hat{\mathbf{B}}_{ij,kl} u_{kl} + hz_{ij} \quad (4.2)$$

If the input is constant or changing slowly, g_{ij} can be treated as a constant and should be computed only once at the beginning of the computation.

B. *Nonlinear templates*

Nonlinear CNN theory was invented by Roska and Chua [5]. In some interesting spatio-temporal problems (Navier-Stokes equations) the nonlinear templates (nonlinear interactions) play key role. In general the nonlinear CNN template values are defined by an arbitrary nonlinear function of input variables (nonlinear **B** template), output variables (nonlinear **A** template) or state variables and may involve some time delays. The survey of the nonlinear templates shows that in many cases the nonlinear template values depend on the difference of the value of the currently processed cell (C_{ij}) and the value of the neighboring cell (C_{kl}). The Cellular Wave Computing Library [14] contains zero- and first-order nonlinear templates. Therefor we focused our efforts to implement this two types of templates, our solution can be extended to handle any type of nonlinearity.

In case of the zero-order nonlinear templates, the nonlinear functions of the template contain horizontal segments only as shown in Figure 4(a). This kind of nonlinearity can be used, e.g., for grayscale contour detection [14].

In case of the first-order nonlinear templates, the nonlinearity of the template contains straight line segments as shown in Figure 4(b). This type of nonlinearity is used, e.g., in the global maximum finder template [14]. Naturally, some nonlinear templates exist in which the template elements are defined by two or more nonlinearities, e.g., the grayscale diagonal line detector [14].

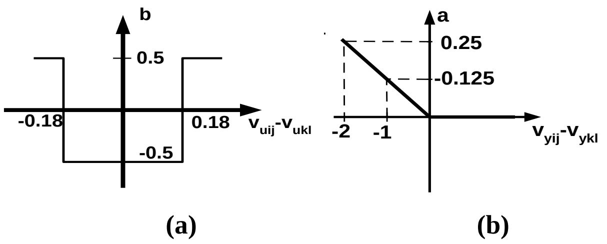

Figure 4. Zero- (a) and first-order (b) nonlinearity

IV. How to Map CNN Array to Cell Processor Array?

A. *Linear Dynamics*

The computation of (4.1) and (4.2) on conventional CISC processors is rather simple. The appropriate elements of the state window and the template are multiplied and the results are summed. Due to the small number of registers on these architectures, 18 Load instructions are required, which slow down the computation. Most of the CISC architectures provide SIMD extensions to speed computation up, but the usefulness of these optimizations is also limited by the small amount of registers.

The large (128-entry) register file of the SPE makes it possible to store the neighborhood of the currently processed cell and the template elements. The number of load instructions can be decreased significantly.

Since the SPEs cannot address the global memory directly, the user’s application running on the SPE is responsible to carry out data transfer between the local memory of the SPE and the global memory via DMA transactions. Depending on the size of the user’s code the 256kbyte local memory of the SPE can approximately store data for a 128x128 sized cell array. To handle larger array only the lines that contain the neighborhood of the currently processed line and required for the next iteration should be stored in the local memory, but it requires continuous synchronized data transfer between the global memory and the local memory of the SPE.

The SPEs in the Cell architecture are SIMD-only units hence the state values of the cells should be grouped into vectors. The size of the registers is 128bit and 32bit floating point numbers are used during the computation. Accordingly, our vectors contain 4 elements. Let's denote the state value of the i^{th} cell by Si.

It seems obvious to pack 4 neighboring cells into one vector. However, constructing the vector which contains the left and right neighbors of the cells is somewhat complicated because 2 “rotate” and 1 “select” instructions are needed to generate the required vector (see Figure 5.). This limits the utilization of the floating-point pipeline because 3 integer instructions (rotate and select) must be carried out before issuing a floating-point multiply-and-accumulate (MAC) instruction.

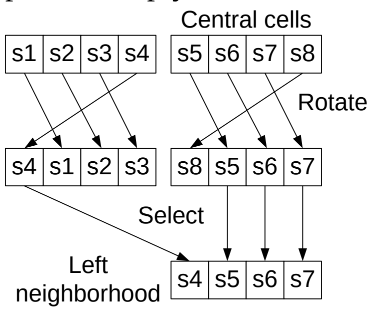

Figure 5. Generation of the left neighborhood

This limitation can be removed by slicing the CNN cell array into 4 vertical stripes and rearranging the cell values. In the above case, the 4-element vector contains data from the 4 different slices as shown in Figure 6. This makes it possible to eliminate the shift and shuffle operations to create the neighborhood of the cells in the vector. The rearrangement should be carried out only once, at the beginning of the computation and can be carried out by the PPE. Though, this solution improves the performance of the simulation data, dependency between the successive MACs still cause floating-point pipeline stalls. In order to eliminate this dependency the inner loop of the computation must be rolled out. Instead of waiting for a result of the first MAC, the computation of the next group of cells is started. The level of unrolling is limited by the size of the register file.

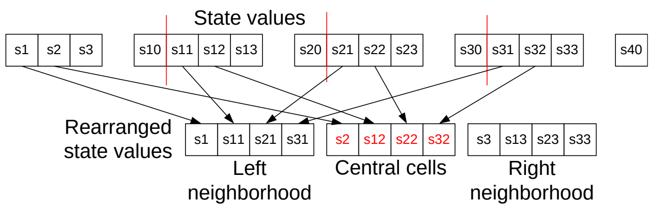

Figure 6. Rearrangement of the state values

To measure the performance of the simulation a 256x256 sized cell array was used and 10 forward Euler iterations were computed, using a diffusion template. By using the IBM Systemsim simulator detailed statistics can be obtained about the operation of the SPEs while executing a program. Additionally, a static timing of the program can be created where the pipeline stalls can be identified. The instruction histogram is shown in Figure 7. Without unrolling, more than 13 million clock cycles are required to complete the computation and the utilization of the SPE is only 35%. Most of the time, the SPE is stalled, due to data dependency between the successive MAC operations. By unrolling the inner loop of the computation and computing 2, 4 or 8 sets of cells, most of the pipeline stall cycles can be eliminated and the required clock cycles can be reduced to 3.5 million. When the inner loop of the computation is unrolled 8 times the efficiency of the SPE can be increased to 90%. Performance of one SPE in the computation of the CNN dynamics is 624 million cell iteration/s. As shown in Figure 15. the SPE is nearly one order faster than a high performance desktop microprocessor.

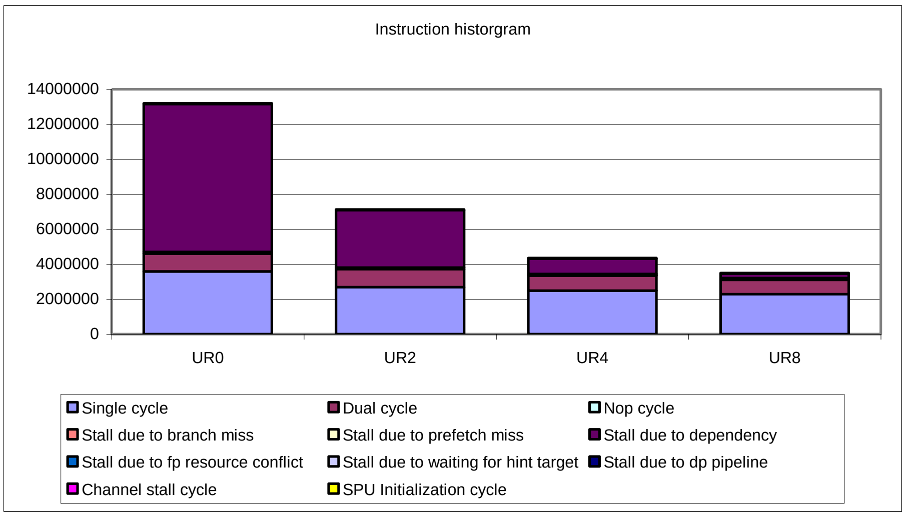

Figure 7. Results of the loop unrolling

To achieve even faster computation multiple SPEs can be used. The data can be partitioned between the SPEs by horizontally striping the CNN cell array. The communication of the state values is required between the adjacent SPEs when the first or last line of the stripe is computed. Due to the row-wise arrangement of the state values, this communication between the adjacent SPEs can be carried out by a single DMA operation. Additionally, the ring structure of the EIB is well suited for the communication between neighboring SPEs.

To measure the performance of the optimized program 16 iterations were computed on a 256x256 sized cell array. The number of required clock cycles is summarized in Figure 8. By using only one SPE, the computation is carried out in 3.3 million clock cycles or 1.04ms, assuming 3.2GHz clock frequency. If 2 SPEs are used to perform the computation, the cycle count is reduced about by half, and nearly linear speedup can be achieved. However, in case of 4 or 8 SPEs the performance cannot be improved. When 4 SPEs are used, SPE number 2 requires more than 5 million clock cycles to compute its stripe. This is larger than the cycle count in case of a single SPE and the performance is degraded.

The examination of the utilization of the SPEs shows that SPE 1 and SPE 2 are stalled, most of the time, and wait for the completion of the memory operations (channel stall cycle). The utilization of these SPEs is less than 15%, while the other SPEs are almost as efficient as a single SPE**.** Investigating the required memory bandwidth shows that one SPE should load 2 32bit floating point values and the result should be stored in the main memory. If 600 million cells are updated every second each SPE requires 7.2Gb/s memory I/O bandwidth and the available 25.6Gb/s bandwidth is not enough to support all the 8 SPEs.

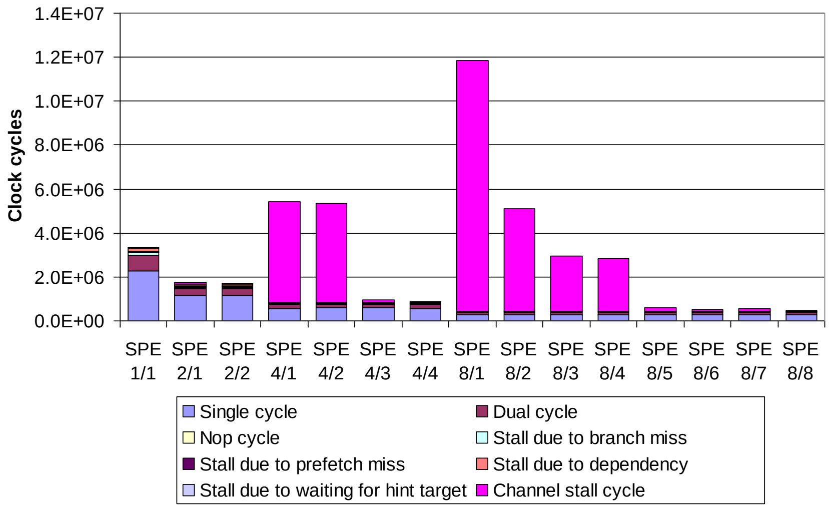

Figure 8. Intruction histogram in case of one and multiple SPEs

To reduce this high bandwidth requirement pipelining technique can be used as shown in Figure 9. In this case the SPEs are chained one after the other, and each SPE computes a different iteration step, using the results of the previous SPE and the Cell processor is working similarly to a systolic array. Only the first and last SPE in the pipeline should access main memory, the other SPEs are loading the results of the previous iteration directly from the local memory of the neighboring SPE. Due to the ring structure of the Element Interconnect Bus (EIB), communication between the neighboring SPEs is very efficient. The instruction histogram of the optimized CNN simulator kernel is summarized in Figure 10.

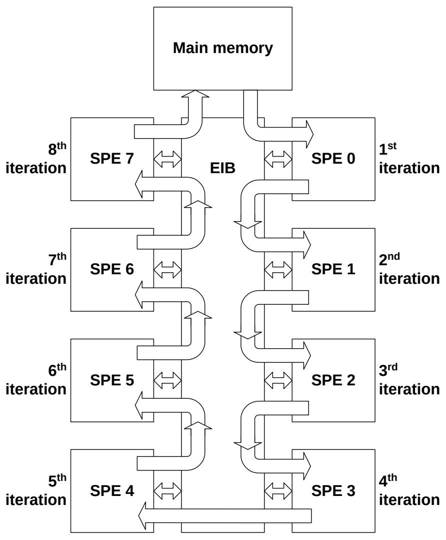

Figure 9. Data-flow of the pipelined multi-SPE CNN simulator

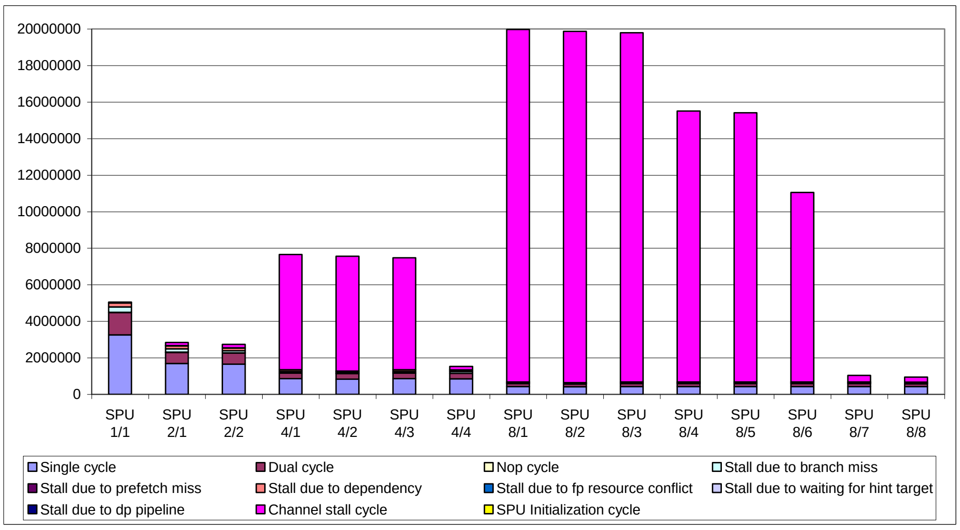

Figure 10. Instruction histogram in case of SPE pipeline

Though the memory bandwidth bottleneck is eliminated by using the SPEs in a pipeline, overall performance of the simulation kernel can not be improved. The instruction histogram in Figure 10. still show huge amount of channel stall cycles in the 4 and 8 SPE case. To find the source of these stall cycles detailed profiling of the simulation kernel is required. First, profiling instructions are inserted into the source code to determine the length of the computation part of the program and the overhead. The results are summarized on Figure 11.

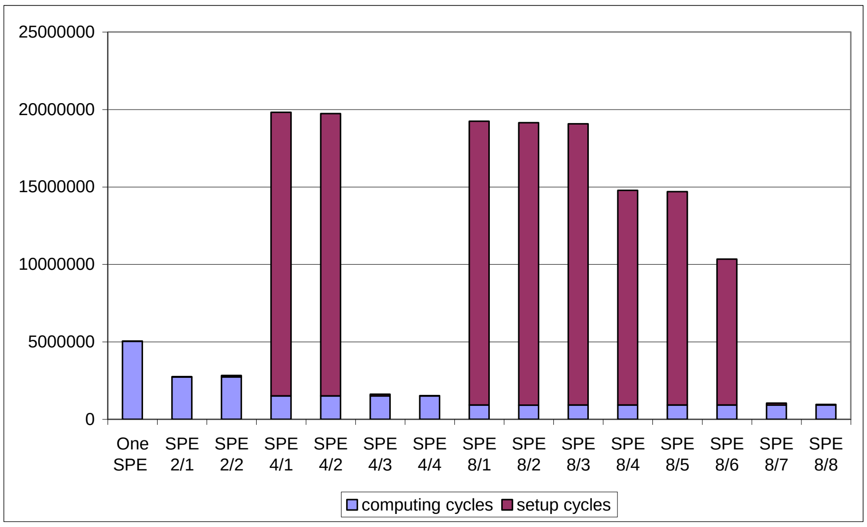

Figure 11. Startup overhead in case of SPE pipeline

Profiling results show that in the 4 and 8 SPE cases some SPEs spend huge amount of cycles in the setup portion of the program. This behavior is caused by the SPE thread creation and initialization process. First, the required number of SPE threads is created, then the addresses required to set up communication between the SPEs and form the pipeline are determined. Meanwhile the started threads are blocked. In the 4 SPE case threads on SPE 0 and SPE 1 are created first, hence these threads should wait not only for the address determination phase but also until the other two threads are created. Unfortunately this thread creation overhead cannot be eliminated. For a long analogic algorithm which uses several templates, the SPEs and the PPE should be used in a client/server model. In this case the SPE threads are not terminated after each template operation but blocked until the PPU determines the parameters for the next operation. To continue the computation, the SPE threads can be simply unblocked and the thread creation and initialization overhead occurs only once in the beginning of the program. Speedup of the implemented CNN simulator kernel running on multiple SPEs is summarized in Figure 12. In the 2 SPE case nearly 2 times faster computation can be achieved while using more SPEs requires more synchronization between the SPEs. Therefore 3.3 and 5.4 times speedup can be achieved in the 4 and 8 SPE cases, respectively.

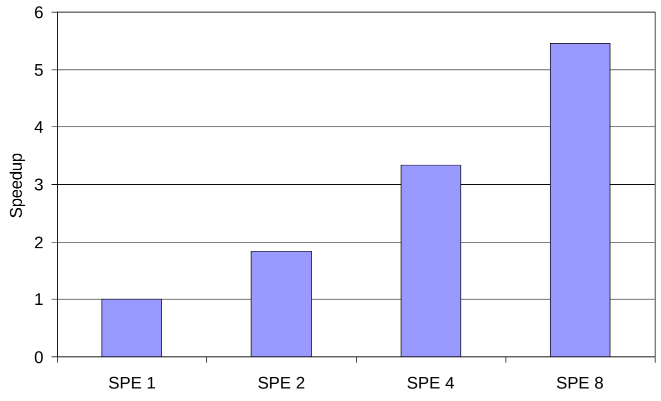

Figure 12. Speedup of the multi-SPE CNN simulation kernel

B. *Nonlinear Dynamics*

To make using zero- and first-order nonlinear templates possible on a conventional scalar processor or on the Cell processor, the nonlinear functions belonging to the templates should be stored in Look Up Tables (LUTs).

In case of conventional scalar processors, each kind of nonlinearity should be partitioned into segments, according to the number of intervals it contains. The parameters of the nonlinear function and the boundary points should be stored in LUTs for each nonlinear template element. In case of the zero-order nonlinear templates, only one parameter should be stored in the LUT, while in case of the first-order nonlinearity, the gradient value and the constant shift of the current section should be stored. By using this arrangement, for zero-order nonlinear templates, the difference of the value of the currently processed cell and the value of the neighboring cell should be compared to the boundary points. The result of this comparison is used to acquire the adequate nonlinear value. In case of the first-order nonlinear template, additional computation is required. After identifying the proper interval of nonlinearity, the difference should be multiplied by the gradient value and added to the constant.

Since the SPEs on the Cell processor are vector processors, the values of the nonlinear function and the boundary points are also stored as a 4-element vector. In each step four differences are computed in parallel and all boundary points must be examined to determine the four nonlinear template elements. To get an efficient implementation, double buffering, vectorization, pipelining, and loop unrolling techniques can be used similarly to the linear template implementation. To investigate the performance of loop unrolling in a nonlinear case, the same 256x256 sized cell array and 16 forward Euler iterations were used as in linear case. The global maximum finder template [14] was used where the template values were defined by a first order nonlinearity which can be partitioned into 2 or 4 segments. Here the number of required instructions was calculated in the case when the inner loop of the computation was not rolled out as well as unrolled 2, 4 and 8 times. In addition, the nonlinearity was segmented into two (A) and four (B) segments. The result of the investigation is shown in Figure 13. We can see that most of the time was spent by an SPE with the single and dual cycle instructions and stall due to dependency, which can be reduced by the unrolling technique. Without unrolling the SPE is stalled due to dependency more than 8 million clock cycles in case the nonlinearity is partitioned into two parts (A). However, by using the unrolling technique it is reduced to nearly 259 000 clock cycles, so almost 50% less clock cycles are required for the computation in the 8 times unrolled case.

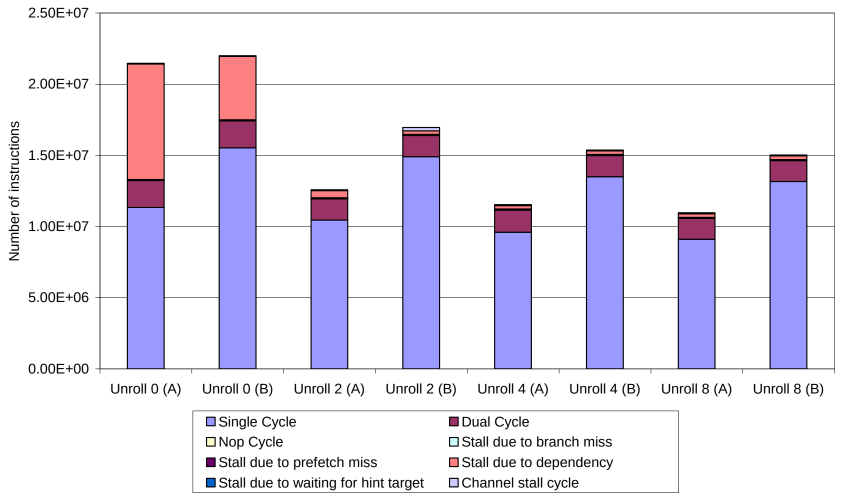

Figure 13. Comparison of the instruction number in case of different unrolling

In addition, the effect of increasing the number of segments of the nonlinearity was investigated. Our experiments show that the number of required instructions is more than 25% higher in case of using four nonlinearity segments (B) than if the nonlinearity is partitioned only into two parts (A). The comparison of the instruction distribution of the upper cases shows that the main difference between them is the number of required single cycles due to usage of more conditional structures during the implementation as shown in Figure 13. In the four segment case (B) the number of stall cycles is smaller than in the two segment case (B) without loop unrolling. By using loop unrolling 30% performance increase can be achieved.

In nonlinear case the performance can also be improved if multiple SPEs are used. Finally, we studied the effect of multiple SPEs. In this test case the inner loop of the computation was rolled out at 8 times, and ran on 1, 2, 4 and 8 SPEs in parallel with the nonlinearity partitioned into two (A) and four segments (B). The comparison of the performance of these cases is shown in Figure 14.

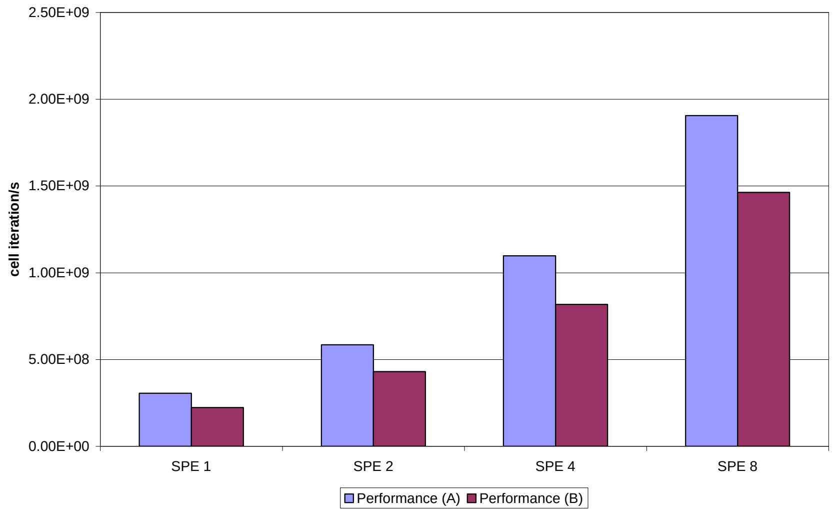

Figure 14. Performance comparison of one and multiple SPEs

If the nonlinearity is partitioned into two segments (A) the maximum performance of the simulation on 1 SPE is about 223 million cell iterations/s, which is almost 24% more than if the nonlinearity is partitioned into four parts (B) as seen in Figure 14. The performance depends not only on the number of segments, but also on the number of SPEs as in the linear case. So if the number of SPEs increases, the performance of the simulations increases in the same way.

C. *Performance comparisons*

The performance of the implementation on the Cell architecture was tested by running the global maximum finder template on a 256x256 image for 16 iterations. The achievable performance of the Cell using different number of SPEs is compared to the performance of the Intel Core 2 Duo T7200 2GHz scalar processor and the linear and nonlinear Falcon Emulated Digital CNN-UM architecture. The results are shown in Figure 15. Comparison of the performance of the single SPE solution to a high performance microprocessor in the linear case showed that about 6 times speedup can be achieved. By using all the 8 SPEs about 35 times speedup can be achieved. Compared to emulated digital architectures one SPE can outperform a single Falcon Emulated Digital CNN-UM core. When using nonlinear templates the performance advantage of the Cell architecture is much higher. In a single SPE configuration 64 times speedup can be achieved while using 8 SPEs the performance is 429 times higher. Typical computing time of one template operation along with area and power requirements of the different architectures are summarized on Table I.

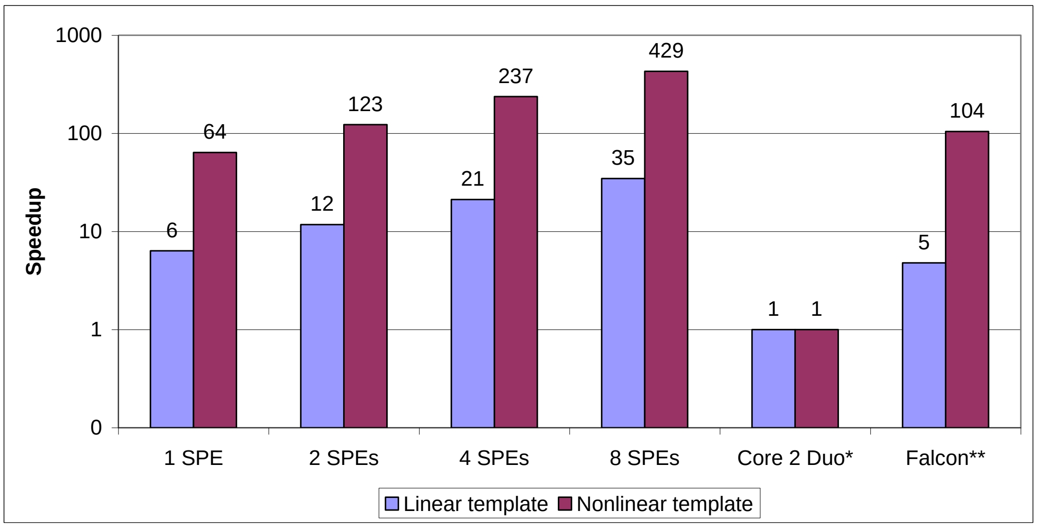

Figure 15. Performance of the implemented CNN simlator on the Cell architecture compared to other architectures (CNN cell array size: 256x256, 16 forward Euler iterations, *Core 2 Duo T7200 @2GHz, **Falcon Emulated Digital CNN-UM implemented on Xilinx Virtex-5 FPGA (XC5VSX95T) @550MHz 1 PE (max. 71 PE)

TABLE I. COMPARISON OF DIFFERENT CNN IMPLEMANTATIONS: 2GHZ CORE 2 DUO PROCESSOR, EMULATED DIGITAL CNN RUNNING ON CELL PROCESSORS AND ON VIRTEX FPGAS, AND Q-EYE ANALOG VLSI CHIP

Parameters	**CNN Implementations**			
	Core 2 Duo	***Q-Eye***	***FPGA***	***CELL (8 SPEs)***
Speed (linear template, µs)	4092.6	250	737.2	111.8
Speed (nonlinear template, µs)	84691.4	-	737.2	197.33
Power (W)	65	0.1	20	85
Area (mm^2)	143	-	~389	253

(CNN cell array size: 176x144, 16 forward Euler iterations)

Let us mention that solution of a given computational problem can be much faster implemented in a Cell architecture then on FPGA.

V. OCEAN MODEL AND ITS IMPLEMENTATION

Several studies proved the effectiveness of the CNN-UM solution of different PDEs [8] [9]. But the results cannot be used in real life implementations because of the limitations of the analog CNN-UM chips such as low precision, temperature sensitivity or the application of non-linear templates. Some previous results show that emulated digital architectures can be very efficiently used in the computation of the CNN dynamics [10] [15] and in the solution of PDEs [11] [12] [13]. Using the CNN simulation kernel described in the previous sections helped to solve Navier-Stokes PDE on the Cell architecture. The details will be presented here.

Simulation of compressible and incompressible fluids is one of the most exciting areas of the solution of PDEs because these equations appear in many important applications in aerodynamics, meteorology, and oceanography. Modeling ocean currents plays a very important role both in medium-term weather forecasting and global climate simulations. In general, ocean models describe the response of the variable density ocean to atmospheric momentum and heat forcing. In the simplest barotropic ocean model a region of the ocean's water column is vertically integrated to obtain one value for the vertically different horizontal currents. The more accurate models use several horizontal layers to describe the motion in the deeper regions of the ocean. Such a model is the Princeton Ocean Model (POM) [16] being a sigma coordinate model in which the vertical coordinate is scaled on the water column depth. Though the model is three-dimensional, it includes a 2-D sub-model (external mode portion of the 3-D model). Investigation of it is not worthless because it is relatively simple, easy to implement, and it provides a good basis for implementation of the 3-D model.

The governing equations of the 2-D model can be expressed from the equations of the 3-D model by making some simplifications. Using the sigma coordinates these equations have the following form:

$$\frac{\partial \eta}{\partial t}+\frac{\partial \overline{U}D}{\partial x}+\frac{\partial \overline{V}D}{\partial y}=0 \tag{5.1}$$

$$\frac{\partial \overline{U}D}{\partial t}+\frac{\partial \overline{U}^2 D}{\partial x}+\frac{\partial \overline{U}\,\overline{V}D}{\partial y}-f\overline{V}D+gD\frac{\partial \eta}{\partial x}=-<wu(0)>+<wu(-1)> -\frac{gD}{\rho_0}\int_{-1}^{0}\int_{\sigma}^{0}\left[D\frac{\partial \rho'}{\partial x}-\frac{\partial D}{\partial x}\sigma'\frac{\partial \rho'}{\partial \sigma}\right]d\sigma' d\sigma \tag{5.2}$$

$$\frac{\partial \overline{V}D}{\partial t}+\frac{\partial \overline{U}\,\overline{V}D}{\partial x}+\frac{\partial \overline{V}^2 D}{\partial y}+f\overline{U}D+gD\frac{\partial \eta}{\partial y}=-<wv(0)>+<wv(-1)> -\frac{gD}{\rho_0}\int_{-1}^{0}\int_{\sigma}^{0}\left[D\frac{\partial \rho'}{\partial y}-\frac{\partial D}{\partial y}\sigma'\frac{\partial \rho'}{\partial \sigma}\right]d\sigma' d\sigma \tag{5.3}$$

where *x,y* are the conventional 2-D Cartesian coordinates; $\sigma=\frac{z-\eta}{H+\eta}, D\equiv H+\eta$, where *H(x,y)* is the bottom topography and η(*x,y,t*) is the surface elevation. The overbars denote vertically integrated velocities such as $\overline{U}\equiv\int_{-1}^{0}Ud\sigma$.

The wind stress components are -<*wu*(0)> and -<*wv*(0)>, and the bottom stress components are -<*wu*(-1)> and -<*wv*(-1)>. *U*, *V* are the horizontal velocities, *f* is the Coriolis parameter, *g* is gravitational acceleration, ρ_0 and ρ' are the reference and in situ density, respectively.

The solution of equations (5.1)-(5.3) is based on the freely available Fortran source code of the POM [16]. The discretization in space is done according to the Arakawa-C differencing scheme where the variables are located on a staggered mesh. The mass transports *U* and *V* are located at the center of the box boundaries facing the x and y directions, respectively. All other parameters are located at the center of mesh boxes. The horizontal grid uses curvilinear orthogonal coordinates.

By using the original Fortran source code a new C based solution is developed which is optimized for the SPEs of the Cell architecture. Since the relatively small local memory of the SPEs does not allow to store all the required data, an efficient buffering method is required. In our solution a belt of 5 rows is stored in the local memory from the array: 3 rows are required to form the local neighborhood of the currently processed row, one line is required for data synchronization, and one line is required to allow overlap of the computation and communication. The maximum width of the row is about 300 elements.

During implementation the environment of the CNN simulation kernel was used. Memory bandwidth requirements are reduced by using the buffering technique described above and forming a pipeline using the SPEs to compute several iterations in parallel. Data dependency between the instructions is eliminated by using loop unrolling.

For testing and performance evaluation purposes a simple initial setup was used which is included in the Fortran source code of the POM. This solves the problem of the flow through a channel which includes an island or a seamount at the center of the domain. The size of the modeled ocean is 1024km, the north and south boundaries are closed, the east and west boundaries are open, the grid size is 128×128 and the grid resolution is 8km. The simulation timestep is 6s and 360 iterations are computed. Experimental results of the average iteration time are summarized on Table II.

TABLE II. COMPARISON OF DIFFERENT CNN OCEAN MODEL IMPLEMANTATIONS: 2GHZ CORE 2 DUO PROCESSOR, EMULATED DIGITAL CNN RUNNING ON CELL PROCESSORS

Parameters	**CNN Implementations**	
	Core 2 Duo	***CELL (8 SPEs)***
Iteration time (ms)	8.2	1.11
Power (W)	65	85
Area (mm^2)	143	253

(CNN cell array size: 128x128, 1 iteration)

VI. CONCLUSION

Complex spatio-temproral dynamical problems are analyzed by a topographic array processor. The Cellular Nonlinear Circuits were successfully used to solve different PDE solutions. Basic CNN simulation kernel was successfully implemented on the Cell architecture. By using this kernel both linear and nonlinear CNN arrays can be simulated. The kernel was optimized according to the special requirements of the Cell architecture. Performance comparison showed that about 6 times speedup can be achieved with respect to a high performance microprocessor in the single SPE solution, while the speedup is 35 times higher when all the 8 SPEs are utilized. When using nonlinear templates the performance advantage of the Cell architecture is much higher. In a single SPE configuration 64 times speedup can be achieved while using 8 SPEs the performance is 429 times higher.

The 2-D part of the POM was successfully implemented on the Cell architecture and significant performance improvement was achieved. In the future we are planning to implement the 3-D part of the POM.

ACKNOWLEDGMENT

The authors would like to thank Professor Tamás Roska for many helpful discussions and suggestions.

REFERENCES

[1] ClearSpeed Inc. homepage: http://www.clearspeed.com/

[2] MathStar Inc. homepage: http://www.mathstar.com/

[3] Tilera Inc. homepage: http://www.tilera.com/

[4] T. Roska and L. O. Chua, "The CNN Universal Machine: an Analogic Array Computer", *IEEE Transaction on Circuits and Systems-II*, vol. 40, pp. 163-173, 1993.

[5] T. Roska and L. O. Chua, "Cellular Neural networks with nonlinear and delay-type template elements and non-uniform girds," Int. J. Circuit Theory and Applications, vol. 20, pp. 469-481, 1992

[6] J. A. Kahle, M. N. Day, H. P. Hofstee, C. R. Johns, T. R. Maeurer, and D. Shippy. Introduction to the cell multiprocessor. IBM Journal of Research and Development [Online] http://www.research.ibm.com/journal/rd/494/kahle.html, 2005.

[7] S. Espejo et.al. "A VLSI-Oriented Continuous-Time CNN Model", Int. Journal of Circuit Theory and Applications, vol. 24, 341-356, 1996.

[8] P. Szolgay, G. Vörös, and Gy. Erőss "On the Applications of the Cellular Neural Network Paradigm in Mechanical Vibrating System", *IEEE. Trans. Circuits and Systems-I, Fundamental Theory and Applications*, vol. 40, no. 3, pp. 222-227, 1993.

[9] Z. Nagy and P. Szolgay, "Numerical solution of a class of PDEs by using emulated digital CNN-UM on FPGAs", *Proc. Of 16th European Conf. On Circuits Theory and Design*, Cracow, vol. II, pp. 181-184, September 1-4, 2003.

[10] Z. Nagy and P. Szolgay, "Configurable Multi-layer CNN-UM Emulator on FPGA", *IEEE Transaction on Circuit and Systems I: Fundamental Theory and Applications*, vol. 50, pp. 774-778, 2003.

[11] Z. Nagy and P. Szolgay, "Solving Partial Differential Equations on Emulated Digital CNN-UM Architectures", *Functional Differential Equations,* vol. 13, No. 1, pp. 61-87 (2006)

[12] P. Kozma, P. Sonkoly, and P. Szolgay, "Seismic Wave Modeling on CNN-UM Architecture", *Functional Differential Equations,* vol. 13, No. 1, pp. 43-60 (2006)

[13] Z. Nagy, Zs. Vörösházi, and P. Szolgay, "Emulated Digital CNN-UM Solution of Partial Differential Equations", *Int. J. CTA*, vol. 34, No. 4, pp. 445-470 (2006)

[14] Cellular Wave Computing Library [Online] Available: http://cnn-technology.itk.ppke.hu/

[15] Z. Nagy, Z. Kincses, L. Kék, P. Szolgay: CNN Model on Cell Multiprocessor Array, Proceedings of the European Conference on Circuit Theory and Design (ECCTD'2007), pp. 276-279, Sevilla, 2007

[16] The Princeton Ocean Model (POM) Homepage [Online] Available: http://www.aos.princeton.edu/WWWPUBLIC/htdocs.pom/index.html