DOI: 10.1109/CNNA.2008.4588670

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
nagyz@sztaki.hu, kek@sztaki.hu, szolgay@sztaki.hu
‡Dept. Image Processing and Neurocomputing, University of Pannonia, Veszprém, Hungary
kincsesz@vision.vein.hu
†Dept. Information Technology, Pázmány Péter Catholic University, Budapest, Hungary
kissa@digitus.itk.ppke.hu

***Abstract*—Array computers can be useful in the solution of numerical spatiotemporal problems such as 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 the paper the implementation of the 3-D Princeton Ocean Model on the Cell BE is discussed. 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. [1]

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 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 MathStars Silicon Object Assembly Language. The Tilera Tile64 architecture [3] is a regular array of general purpose processors, called Tile Processors, arranged on an 8×8 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. It is exploited here in solving complex, time consuming problems.

II. Cell Processor Architecture

A. Cell Processor Chip

The Cell Broadband Engine Architecture (CBEA) [4] 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 Single Instruction Multiple Data 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

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

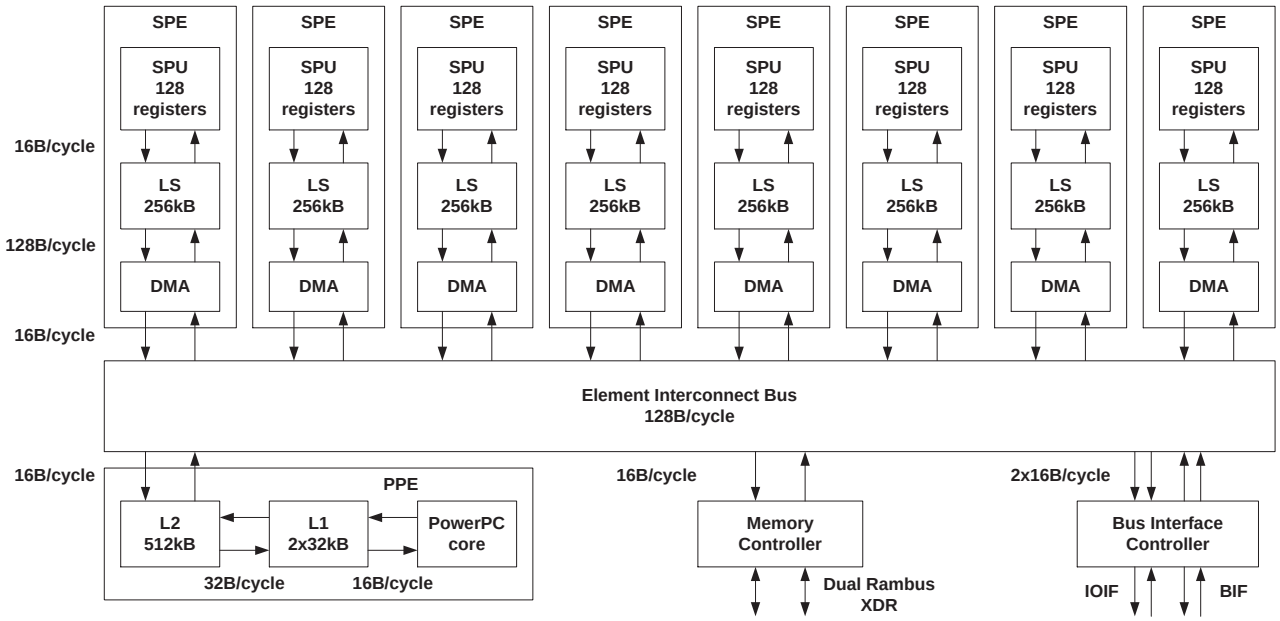

Fig. 1. Block diagram of the Cell processor

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.

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.

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

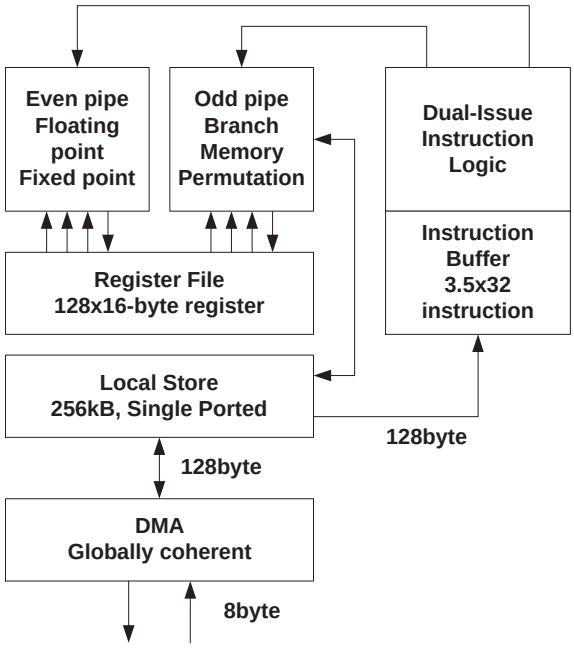

Fig. 2. Block diagram of the Synergistic Processor Element

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.

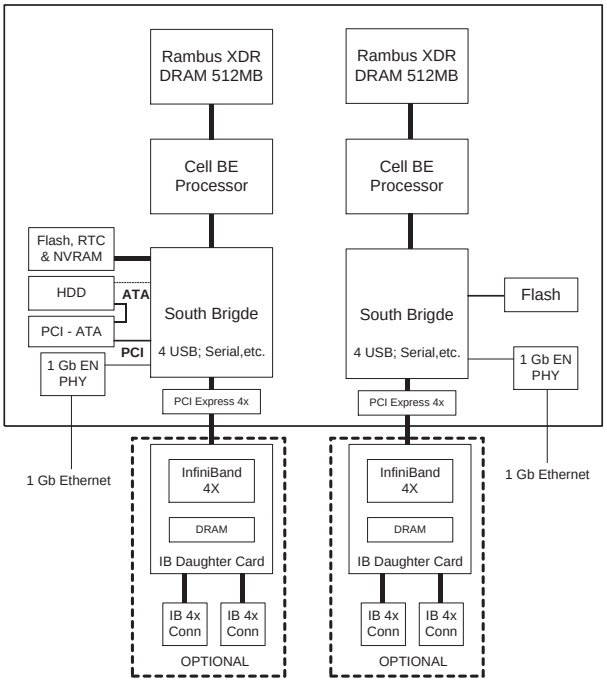

Fig. 3. IBM Blade Center QS20 architecture

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.

III. Ocean model and its implementation

Several studies proved the effectiveness of the CNN-UM solution of different PDEs [5] [6]. 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 [7] [8] and in the solution of PDEs [9] [10] [11]. Using the CNN simulation kernel described in [8] 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 oceans 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) [12] being a sigma coordinate model in which the vertical coordinate is scaled on the water column depth.

$$\frac{\partial DU}{\partial x}+\frac{\partial DV}{\partial y}+\frac{\partial \omega}{\partial \sigma}+\frac{\partial \eta}{\partial t}=0 \tag{1}$$

$$\begin{aligned}&\frac{\partial UD}{\partial t}+\frac{\partial U^2D}{\partial x}+\frac{\partial UVD}{\partial y}+\frac{\partial U\omega}{\partial \sigma}-fVD+\\&+gD\frac{\partial \eta}{\partial x}+\frac{gD^2}{\rho_0}\int_\sigma^0\left[\frac{\partial \rho'}{\partial x}-\frac{\sigma'}{D}\frac{\partial D}{\partial x}\frac{\partial \rho'}{\partial \sigma'}\right]d\sigma'=\\&=\frac{\partial}{\partial \sigma}\left[\frac{K_M}{D}\frac{\partial U}{\partial \sigma}\right]+F_x\end{aligned} \tag{2}$$

$$\begin{aligned}&\frac{\partial VD}{\partial t}+\frac{\partial UVD}{\partial x}+\frac{\partial V^2D}{\partial y}+\frac{\partial V\omega}{\partial \sigma}-fUD+\\&+gD\frac{\partial \eta}{\partial y}+\frac{gD^2}{\rho_0}\int_\sigma^0\left[\frac{\partial \rho'}{\partial y}-\frac{\sigma'}{D}\frac{\partial D}{\partial y}\frac{\partial \rho'}{\partial \sigma'}\right]d\sigma'=\\&=\frac{\partial}{\partial \sigma}\left[\frac{K_M}{D}\frac{\partial V}{\partial \sigma}\right]+F_y\end{aligned} \tag{3}$$

$$\begin{aligned}&\frac{\partial TD}{\partial t}+\frac{\partial TUD}{\partial x}+\frac{\partial TVD}{\partial y}+\frac{\partial T\omega}{\partial \sigma}=\\&=\frac{\partial}{\partial \sigma}\left[\frac{K_H}{D}\frac{\partial T}{\partial \sigma}\right]+F_T-\frac{\partial R}{\partial z}\end{aligned} \tag{4}$$

$$\begin{aligned}&\frac{\partial SD}{\partial t}+\frac{\partial SUD}{\partial x}+\frac{\partial SVD}{\partial y}+\frac{\partial S\omega}{\partial \sigma}=\\&=\frac{\partial}{\partial \sigma}\left[\frac{\partial K_H}{\partial D}\frac{\partial S}{\partial \sigma}\right]+F_S\end{aligned} \tag{5}$$

$$\frac{\partial q^2 D}{\partial t} + \frac{\partial U q^2 D}{\partial x} + \frac{\partial V q^2 D}{\partial y} + \frac{\partial \omega q^2}{\partial \sigma} = \\ = \frac{\partial}{\partial \sigma}\left[\frac{\partial K_q}{\partial D}\frac{\partial q^2}{\partial \sigma}\right] + \frac{2K_M}{D}\left[\left(\frac{\partial U}{\partial \sigma}\right)^2 + \left(\frac{\partial V}{\partial \sigma}\right)^2\right] + \\ + \frac{2g}{\rho}K_H\frac{\partial \rho}{\partial \sigma} - \frac{2Dq^3}{B_1 \ell} + F_q \quad (6)$$

$$\frac{\partial q^2 l D}{\partial t} + \frac{\partial U q^2 l D}{\partial x} + \frac{\partial V q^2 l D}{\partial y} + \frac{\partial \omega q^2 l}{\partial \sigma} = \\ = \frac{\partial}{\partial \sigma}\left[\frac{\partial K_q}{\partial D}\frac{\partial q^2 l}{\partial \sigma}\right] + \\ + E_1 l\left(\frac{K_M}{D}\left[\left(\frac{\partial U}{\partial \sigma}\right)^2 + \left(\frac{\partial V}{\partial \sigma}\right)^2\right] + E_3\frac{g}{\rho_0}K_H\frac{\partial \rho}{\partial \sigma}\right) \\ - \frac{Dq^3}{B_1}W + F_l \quad (7)$$

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 $\eta(x, y, t)$ is the surface elevation. The U and V terms are the horizontal and vertical velocities, T is the temperature and S is the salinity. The ω denotes the transformed vertical velocity according to the σ coordinates. The F_x and the F_y values are the horizontal viscosity and diffusion terms. The solution of equations (1)-(7) is based on the freely available Fortran source code of the POM [12]. 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.

The equations, governing the dynamics of coastal circulation, contain fast moving external gravity waves and slow moving internal gravity waves. It is desirable in terms of computer economy to separate the vertically integrated equations (external mode) from the vertical structure equations (internal mode). This technique, known as mode splitting permits the calculation of the free surface elevation with little sacrifice in computational time by solving the velocity transport separately from the three-dimensional calculation of the velocity and the thermodynamic properties.

The external mode calculation is responsible for computing surface elevation and the vertically averaged velocities. The internal mode computes horizontal and vertical velocities (U, V), temperature (T) and salinity (S). During the calculation the former uses short time step, whereas the latter uses longer time step. Using this method many external steps are evaluated for every long internal time step. These results are used for the internal mode computation.

The velocity external mode equations are obtained by integrating the internal mode equations over the depth, thereby eliminating all vertical structure. Thus, by integrating Equations (1)-(3) from $\sigma = -1$ to $\sigma = 0$ the following equations can be obtained:

$$\frac{\partial \eta}{\partial t} + \frac{\partial \overline{U} D}{\partial x} + \frac{\partial \overline{V} D}{\partial y} = 0 \quad (8a)$$

$$\frac{\partial \overline{U} D}{\partial t} + \frac{\partial \overline{U}^2 D}{\partial x} + \frac{\partial \overline{UV} D}{\partial y} - f\overline{V}D + gD\frac{\partial \eta}{\partial x} = \\ = - < wu(0) > + < wu(-1) > - \\ - \frac{gD}{\rho_0}\int_{-1}^{0}\int_{\sigma}^{0}\left[D\frac{\partial \rho'}{\partial x} - \frac{\partial D}{\partial x}\sigma'\frac{\partial \rho'}{\partial \sigma}\right]d\sigma' d\sigma \quad (8b)$$

$$\frac{\partial \overline{V} D}{\partial t} + \frac{\partial \overline{UV} D}{\partial x} + \frac{\partial \overline{V}^2 D}{\partial y} + f\overline{U}D + gD\frac{\partial \eta}{\partial y} = \\ = - < wv(0) > + < wv(-1) > - \\ - \frac{gD}{\rho_0}\int_{-1}^{0}\int_{\sigma}^{0}\left[D\frac{\partial \rho'}{\partial y} - \frac{\partial D}{\partial y}\sigma'\frac{\partial \rho'}{\partial \sigma}\right]d\sigma' d\sigma \quad (8c)$$

The overbars denote vertically integrated velocities such as $\overline{U} \equiv \int_{-1}^{0} U d\sigma$. 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.

IV. Implementation on Cell processor array

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

Since the SPEs cannot address the global memory directly, the users 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. 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. Depending on the size of the users code the 256Kbyte local memory of the SPE can approximately store data for a 128 cell wide array.

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 ith cell by si.

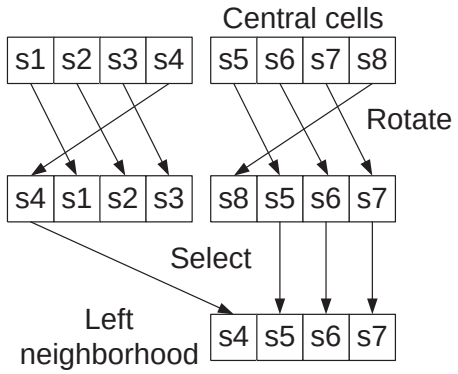

Fig. 4. Generation of the left neighborhood

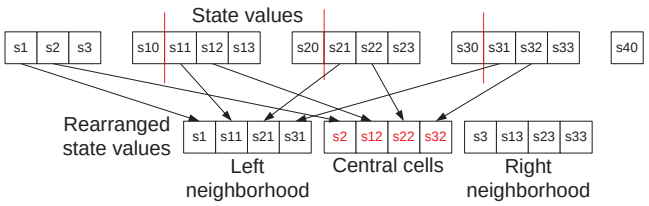

Fig. 5. Rearrangement of the state values

It seems obvious to pack 4 neighboring cells into one vector $\{s5, s6, s7, s8\}$. However, constructing the vector which contains the left $\{s4, s5, s6, s7\}$ and right $\{s6, s7, s8, s9\}$ neighbors of the cells is somewhat complicated because 2 "rotate" and 1 "select" instructions are needed to generate the required vector (see Figure 4). This limits the utilization of the floating-point pipeline because 3 integer instructions (rotate and select) must be carried out to generate the left and right neighborhood of the cell, before a floating point instruction can be issued.

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 5. 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, data dependency between the floating-point instructions may still cause pipeline stalls. In order to eliminate this dependency the inner loop of the computation must be rolled out. Instead of waiting for the result of the first floating-point instruction, the computation of the next group of cells is started. The level of unrolling is limited by the size of the register file.

To achieve even faster computation multiple SPEs can be used. The data can be partitioned between the SPEs by horizontally striping the 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.

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

Parameters	CNN Implementation	
	Core2Duo	CELL (6 SPEs)
Iteration time in 2D (ms)	8.2	0.103
Iteration time in 3D (ms)	1117	12.98
Computation time of a 72 hour simulation (s)	1962.7	23.16
Power (W)	65	85
Area (mm2)	143	253

(CNN cell array size: 128×128×32)

V. PERFORMANCE COMPARISONS

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 128×128×32 and the horizontal grid resolution is 8km. The simulation was ran using 6s internal timestep and 180s external timestep, the simulated time interval was 72 hours. Experimental results of the average iteration time are summarized in Table I.

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. Comparison of the required computation time of one iteration in external (2D) and internal (3D) mode show that the external mode computations can be carried out 126 times faster. The result is significant saving on computation time. Performance of the six-SPE solution is compared to the performance of a high performance microprocessor. The external mode calculations on the Cell processor are 79-time faster than on the Core 2 Duo microprocessor, while in the internal mode 86-time speedup can be achived. During a 72 hours simulation using both internal and external mode calculations 85-time speedup was measured.

VI. CONCLUSION

Complex spatio-temporal dynamical problems are analyzed by a topographic array processor. The Cellular Nonlinear Circuits were successfully used to solve the 3-D Princeton Ocean Model and significant performance improvement was achieved. Our solution was optimized according to the special requirements of the Cell architecture. Performance comparison showed that about 17-time speedup can be achieved with respect to a high performance microprocessor in the single SPE solution, while the speedup is 85-time higher when all the 6 SPEs are utilized.

ACKNOWLEDGMENT

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

REFERENCES

[1] "ClearSpeed Inc." [Online] http://www.clearspeed.com/.
[2] "MathStar Inc." [Online] http://www.mathstar.com/.
[3] "Tilera Inc." [Online] http://www.tilera.com/.
[4] 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*, 2005.
[5] P. Szolgay, G. Vörös, and G. 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.
[6] 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*, vol. II, pp. 181–184, September 2003.
[7] ——, "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.
[8] Z. Nagy, Z. Kincses, L. Kék, and P. Szolgay, "CNN Model on Cell Multiprocessor Array," *Proceedings of the European Conference on Circuit Theory and Design (ECCTD'2007)*, pp. 276–279, 2007.
[9] Z. Nagy and P. Szolgay, "Solving Partial Differential Equations on Emulated Digital CNN-UM Architectures," *Functional Differential Equations*, vol. 13, pp. 61–87, 2006.
[10] 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.
[11] Z. Nagy, Z. Vörösházi, and P. Szolgay, "Emulated Digital CNN-UM Solution of Partial Differential Equations," *Int. J. CTA*, vol. 34, no. 4, pp. 445–470, 2006.
[12] "The Princeton Ocean Model (POM)," [Online] http://www.aos.princeton.edu/aos.