DOI:10.15774/PPKE.ITK.2011.003

An Optimal Mapping of Numerical Simulations of Partial Differential Equations to Emulated Digital CNN-UM Architectures

András Kiss

A thesis submitted for the degree of
Doctor of Philosophy

Scientific adviser:
Zoltán Nagy

Supervisor:
Péter Szolgay

Faculty of Information Technology
Péter Pázmány Catholic University

Budapest, 2011

DOI:10.15774/PPKE.ITK.2011.003

DOI:10.15774/PPKE.ITK.2011.003

I would like to dedicate this thesis to my beloved grandfathers.

DOI:10.15774/PPKE.ITK.2011.003

DOI:10.15774/PPKE.ITK.2011.003

Acknowledgements

It is not so hard to get a Doctoral Degree if you are surrounded with talented, motivated, optimistic, wise people who are not hesitating to give guidance if you get stuck and knowledge to pass through difficulties. There are two men who motivated me to continue my study after the university, and pushed me forward continuously to reach my humble goals. They know my path, because they already walked on it. This work could not have come into existence without the aid of my supervisor and mentor Professor Peter Szolgay and my adviser and friend Dr. Zoltán Nagy.

I am also grateful to my closest collaborators for helping me out in tough situations, to Dr. Zsolt Vörösházi, Sándor Kocsárdi, Zoltán Kincses, Péter Sonkoly, László Füredi and Csaba Nemes.

I would further like to say thanks to my talented colleagues who continuously suffer from my crazy ideas, and who not chases me away with a torch, especially to Éva Bankó, Petra Hermann, Gergely Soós, Barna Hegyi, Béla Weiss, Dániel Szolgay, Norbert Bérci, Csaba Benedek, Róbert Tibold, Tamás Pilissy, Gergely Treplán, Ádám Fekete, József Veres, Ákos Tar, Dávid Tisza, György Cserey, András Oláh, Gergely Feldhoffer, Giovanni Pazienza, Endre Kósa, Ádám Balogh, Zoltán Kárász, Andrea Kovács, László Kozák, Vilmos Szabó, Balázs Varga, Tamás Fülöp, Gábor Tornai, Tamás Zsedrovits, András Horváth, Miklós Koller, Domonkos Gergelyi, Dániel Kovács, László Laki, Mihály Radványi, Ádám Rák, Attila Stubendek.

I am grateful to the Hungarian Academy of Sciences (MTA-SZTAKI) and Péter Pázmány Catholic University, where I spent my Ph.D. years.

DOI:10.15774/PPKE.ITK.2011.003

I am indebted to Katalin Keserű from MTA-SZTAKI, and various offices at Péter Pázmány Catholic University for their practical and official aid.

I am very grateful to my mother and father and to my whole family who always tolerated the rare meeting with me and supported me in all possible ways.

DOI:10.15774/PPKE.ITK.2011.003

Contents

DOI:10.15774/PPKE.ITK.2011.003

DOI:10.15774/PPKE.ITK.2011.003

DOI:10.15774/PPKE.ITK.2011.003

DOI:10.15774/PPKE.ITK.2011.003

List of Figures

DOI:10.15774/PPKE.ITK.2011.003

DOI:10.15774/PPKE.ITK.2011.003

DOI:10.15774/PPKE.ITK.2011.003

DOI:10.15774/PPKE.ITK.2011.003

List of Tables

DOI:10.15774/PPKE.ITK.2011.003

DOI:10.15774/PPKE.ITK.2011.003

List of Abbreviations

ALU Arithmetic Logic Unit

ASIC Application-specific Integrated Circuit

ASMBL Advanced Silicon Modular Block

BRAM Block RAM

CAD Computer Aided Design

CBEA Cell Broadband Engine Architecture

CFD Computational Fluid Dynamics

CFL Courant-Friedrichs-Lewy

CISC Complex Instruction Set Computer

CLB Configurable Logic Blocks

CNN Cellular Neural/Nonlinear Network

CNN-UM Cellular Neural/Nonlinear Network - Universal Machine

DMA Direct Memory Acces

DSP Digital Signal Processing

EIB Element Interconnect Bus

FIFO Firs In First Out

FIR Finite-impulse Response

DOI:10.15774/PPKE.ITK.2011.003

FPE Falcon Processing Element

FPGA Field Programmable Gate Array

FPOA Field Programmable Object Array

FSR Full Signal Range

GAPU Global Analogic Programming Unit

IPIF Intellectual Property Interface

ITRS International Technology Roadmap for Semiconductors

LUT Look-Up Table

MAC Multiply - accumulate Operation

MIMD Multiple Instruction Multiple Data

MPI Message Passing Interface

OPB On-Chip Peripheral Bus

PDE Partial Differential Equation

PE Processing Element

PPC PowerPC

PPE Power Processor Element

RISC Reduced Instruction Set Computer

SDK Software Development Kit

SIMD Single Instruction Multiple Data

SLR Super Logic Region

SPE Synergistic Processor Element

SSI Stacked Silicon Interconnect

DOI:10.15774/PPKE.ITK.2011.003

TFLOPS Tera Floating Point Operations Per Second

VHDL VHSIC hardware description language

VLIW Very Long Instruction Word

VLSI Very-large-scale integration

VPE Vector Processing Element

DOI:10.15774/PPKE.ITK.2011.003

DOI:10.15774/PPKE.ITK.2011.003

Chapter 1

Introduction

Due to the rapid evolution of computer technology the problems on many processing elements, which are arranged in regular grid structures (array processors), become important. With the large number of the processor cores not only the speed of the cores but their topographic structure becomes an important issue. These processors are capable of running multiple tasks in parallel. In order to make an efficiently executed algorithm, the relative distance between two neighboring processing elements should be taken into consideration. In other words it is the precedence of locality phenomenon. This discipline requires the basic operations to be redesigned in order to work on these hardware architectures efficiently.

In the dissertation solutions for solving hard computational problems are searched, where the area and dissipated power is minimal, the number of implemented processor, the speed and the memory access are maximal. A solution is searched within this parameter space for an implementation of a partial differential equation, and the solution is optimized for some variable of this parameter space (e.g.: speed, area, bandwidth). The search space will be always limited by the special properties of the hardware environment.

There are several known problems, which cannot be computed in real time with the former resources, or just very slowly. The aim of the research is the examination of these hard problems. As an example a fluid flow simulation is going to be analyzed, and a hardware implementation for the problems is going to be introduced.

DOI:10.15774/PPKE.ITK.2011.003

The motivation of the dissertation is to develop a methodology for solving partial differential equations, especially for liquid and gas flow simulations, which helps to map these problems optimally into inhomogenous and reconfigurable architectures. To reach this goal two hardware platforms as experimental framework were built up, namely the IBM Cell Broadband Engine Architecture and the Xilinx Field Programmable Gate Array (FPGA) as reconfigurable architecture.

After the creation of the framework, which models fluid flow simulations, it is mapped to these two architecture which follows different approach (see Chapter 2). To fully utilize the capabilities of the two architecture several optimization procedure had to performed. Not only the structure of the architectures are taken into consideration, but the computational precision too (introduced in Chapter 3). It is important to examine the precision of the arithmetic unit on FPGA, because significant speedup or lesser area requirement and power dissipation can be achieved. With the investigation of the computational precision, the decision can be taken, wether the problem fits onto the selected FPGA or not. It relies mainly on the number of operations in the arithmetic unit. To get a complete, standalone machine, the processing element on FPGA should be extended with a control unit (it is going to e introduced in Chapter 4).

The IBM Cell processor represents a bounded architecture, which builds up from heterogeneous processor cores. From the marketing point of view, the Cell processor failed, but its significant innovations (e.g.: heterogeneous processor cores, ring bus structure) can be observed in todays modern processors (e.g.: IBM Power 7 [13], Intel Sandy Bridge [14]). According to the special requirement of the processor, vectorized datas were used which composed of floating point numbers. For the development of the software the freely available IBM software development kit (SDK) with C programming language was used.

Xilinx FPGAs are belonging to the leading reconfigurable computers since a while. Due to the fast Configurable Logic Blocks (CLB) and to the large number of interconnections arbitrary circuits can be implemented on them. In order to accelerate certain operations, dedicated elements (e.g.: digital signal processing (DSP) blocks) are available on the FPGA. The FPGA's CLB and DSP can be treated like different type of processors which can handle different operations efficiently. Due to the configurable parameters of the FPGA the processed data

DOI:10.15774/PPKE.ITK.2011.003

can be represented in arbitrary type and size. During the research fixed point and floating point number arithmetic units with different mantissa width were investigated in order to find the optimal precision for a qualitative good result. During the implementation process I used the Xilinx Foundation ISE softwares with VHSIC hardware description language (VHDL) language. For the software simulation I used the MentorGraphics Modelsim SE software.

The Chapter 1 is built up as follows. First the CNN paradigm is introduced. It is capable to solve complex spatio-temporal problems. The standard CNN cell should be extended with a control unit and with memory units in order to get an universal machine, which is based on the stored programmability (the extension is introduced in Chapter 4). Several studies proved the effectiveness of the CNN-UM solution of different PDEs [15, 16]. After the previous section several implementations of the CNN-UM principle are listed. The emulated digital CNN-UM is a reasonable alternative to solve different PDEs. It has mixed the benefits of the software simulation and the analog solution. Namely the high configurability and the precision of the software solution and the high computing speed of the analog solution. In the later Chapters the CNN simulation kernel is going to be introduced on the IBM Cell processor and on the FPGA. In order to get the reader a clear understanding from the framework a brief introduction of the hardware specifications are presented. The final section is an outlook of the recent trends in many core systems.

DOI:10.15774/PPKE.ITK.2011.003

1.1 Cellular Neural/Nonlinear Network

The basic building blocks of the Cellular Neural Networks, which was published in 1988 by L. O. Chua and L. Yang [17], are the uniform structured analog processing elements, the cells. A standard CNN architecture consists of a rectangular 2D-array of cells as shown in Figure (1.1). With interconnection of many 2D arrays it can be extended to a 3-dimensional, multi-layer CNN structure. As it is in organic structures the simplest way to connect each cell is the connection of the local neighborhood via programmable weights. The weighted connections of a cell to its neighbors are called the cloning template. The CNN cell array is programmable by changing the cloning template. With the local connection of the cells difficult computational problems can be solved, like modeling biological structures [18] or the investigation of the systems which are based on partial differential equations [19].

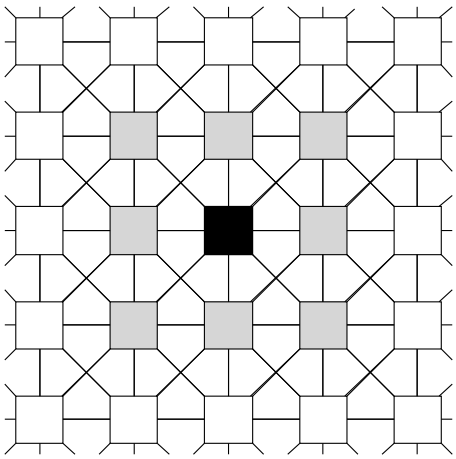


Figure 1.1: Location of the CNN cells on a 2D grid, where the gray cells are the direct neighbors of the black cell

1.1.1 Linear templates

The state equation of the original Chua-Yang model [17] is as follows:

$$\dot{x}_{ij}(t) = -x_{ij} + \sum_{C\,(kl)\in N_r(i,j)} \mathbf{A}_{ij,kl} y_{kl}(t) + \sum_{C\,(kl)\in N_r(i,j)} \mathbf{B}_{ij,kl} u_{kl} + z_{ij} \qquad (1.1)$$

DOI:10.15774/PPKE.ITK.2011.003

where u_{kl}, x_{ij}, and y_{kl} are the input, the state, and the output variables. A and B matrices 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 output y_{ij} equation of the cell is described by the following function (see Figure 1.2):

$$y_{ij} = f(x_{ij}) = \frac{|x_{ij}+1| - |x_{ij}-1|}{2} = \begin{cases} 1 & x_{ij}(t) > 1 \\ x_{ij}(t) & -1 \leq x_{ij}(t) \leq 1 \\ -1 & x_{ij}(t) < -1 \end{cases} \tag{1.2}$$

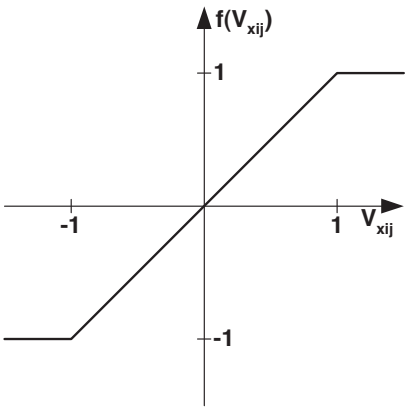


Figure 1.2: The output sigmoid function

The discretized form of the original state equation (1.1) is derived by using the forward Euler form. It is as follows:

$$\begin{aligned} x_{ij}(n+1) &= (1-h)x_{ij}(n) + \\ &+h\left(\sum_{C(kl)\in N_r(i,j)} \mathbf{A}_{ij,kl}y_{kl}(n) + \sum_{C(kl)\in N_r(i,j)} \mathbf{B}_{ij,kl}u_{kl} + z_{ij}\right) \end{aligned} \tag{1.3}$$

In order to simplify computation variables are eliminated as far as possible (e.g.: combining variables by extending the template matrices). First of all, the Chua-Yang model is changed to the Full Signal Range (FSR) [20] 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

DOI:10.15774/PPKE.ITK.2011.003

equation with FSR model is as follows:

$$\begin{gathered} x_{ij}(n+1) = \begin{cases} 1 & if \quad v_{ij}(n) > 1 \\ v_{ij}(k) & if \quad |v_{ij}(n)| \leq 1 \\ -1 & if \quad v_{ij}(n) < -1 \end{cases} \\ v_{ij}(n) = (1-h)x_{ij}(n) + \\ +h\left(\sum_{C(kl)\in N_r(i,j)} A_{ij,kl}x_{kl}(n) + \sum_{C(kl)\in N_r(i,j)} B_{ij,kl}u_{kl}(n) + z_{ij}\right) \end{gathered} \tag{1.4}$$

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 $\hat{\mathrm{A}}$, $\hat{\mathrm{B}}$.

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

$$v_{ij}(n+1) = \sum_{C\,(kl)\in N_r(i,j)} \hat{\mathbf{A}}_{ij,kl}x_{kl}(n) + g_{ij} \tag{1.5a}$$

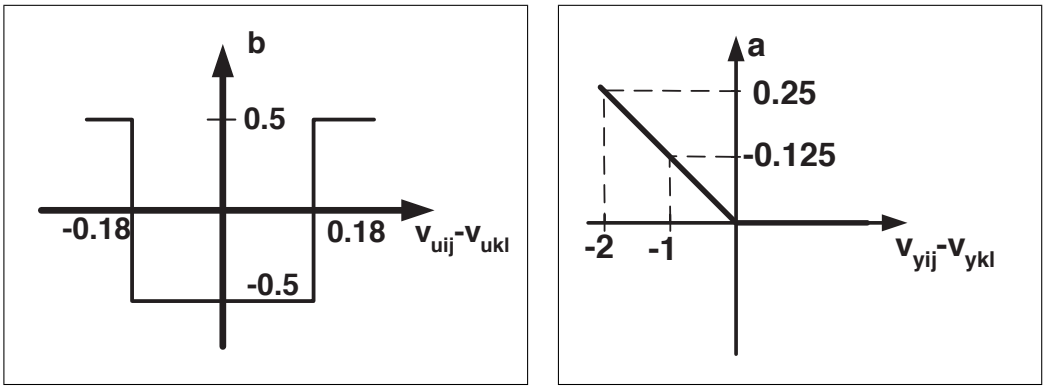
$$g_{ij} = \sum_{C\,(kl)\in N_r(i,j)} \hat{\mathbf{B}}_{ij,kl}u_{kl} + hz_{ij} \tag{1.5b}$$

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.

1.1.2 Nonlinear templates

The implementation of nonlinear templates are very difficult on analog VLSI and quite simple on emulated digital CNN. 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 [21] contains zero- and first-order nonlinear templates.

DOI:10.15774/PPKE.ITK.2011.003



(a) (b)

Figure 1.3: Zero- (a) and first-order (b) nonlinearity

In case of the zero-order nonlinear templates, the nonlinear functions of the template contains horizontal segments only as shown in Figure 1.3(a). This kind of nonlinearity can be used, e.g., for grayscale contour detection [21].

In case of the first-order nonlinear templates, the nonlinearity of the template contains straight line segments as shown in Figure 1.3(b). This type of nonlinearity is used, e.g., in the global maximum finder template [21]. Naturally, some nonlinear templates exist in which the template elements are defined by two or more nonlinearities, e.g., the grayscale diagonal line detector [21].

1.2 Cellular Neural/Nonlinear Network - Universal Machine

If we consider the CNN template as an instruction, we can make different algorithms, functions from these templates. In order to run these algorithms efficiently, the original CNN cell has to be extended (see Figure 1.4) [22]. The extended architecture is the Cellular Neural/Nonlinear Network - Universal Machine (CNN-UM). According to the Turing-Church thesis in case of the algorithms, which are defined on integers or on a finite set of symbols, the Turing Machine, the grammar and the μ - recursive functions are equivalent. The CNN-UM is universal in Turing sense because every μ - recursive function can be computed on this architecture.

DOI:10.15774/PPKE.ITK.2011.003

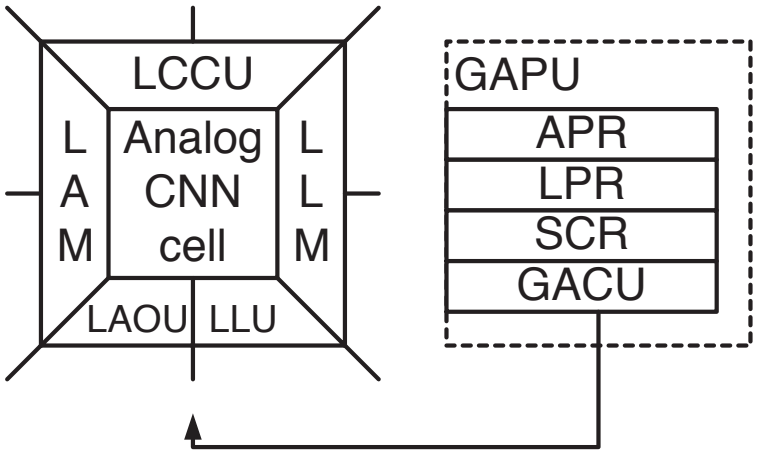


Figure 1.4: The architecture of the CNN Universal Machine, the extended CNN nucleus and the functional blocks of the GAPU

In order to run a sequence of templates, the intermediate results should be stored localy. Local memories connected to the cell store analog (LAM: Local Analog Memory) and logic (LLM: Local Logical Memory) values in each cell. A Local Analog Output Unit (LAOU) and a Local Logic Unit (LLU) perform cell-wise analog and logic operations on the local (stored) values. The LAOU is a multiple-input single output analog device. It combines local analog values into a single output. It is used for analog addition, instead of using the CNN cell for addition. The output is always transferred to one of the local memories. The Local Communication and Control Unit (LCCU) provides for communication between the extended cell and the central programming unit of the machine, across the Global Analogic Control Unit part of the Global Analogic Programming Unit (GAPU).

The GAPU is the "conductor" of the whole analogic CNN universal machine, it directs all the extended standard CNN universal cells. The GAPU stores, in digital form, the sequence of instructions. Before the computations, the LCCU receives the programming instructions, the analog cloning template values A, B, z, the logic function codes for the LLU, and the switch configuration of the cell specifying the signal paths. These instructions are stored in the registers of the GAPU. The Analog Program Register (APR) stores the CNN templates,

DOI:10.15774/PPKE.ITK.2011.003

the Logic Program Register (LPR) stores the LLU functions and the Switch Configuration Register (SCR) contains the setting of switches of an elementary operation of CNN cell.

1.3 CNN-UM Implementations

Since the introduction of the CNN Universal Machine in 1993 [22] several CNN-UM implementations have been developed. These implementations are ranged from the simple software simulators to the analog VLSI solutions.

The software simulator program (for multiple layers) running on a PC calculates the CNN dynamics for a given template by using one of the numerical methods either by a gray-scale code or a black and white image and can simulate the CNN dynamics for a given sequence of templates. The software solutions are flexible, with the configuration of all the parameters (template sizes, accuracy, etc.), but they are insufficient, considering the performance of computations.

The fastest CNN-UM implementations are the analog/mixed-signal VLSI (Very Large Scale Integration) CNN-UM chips [23], [20], [24]. The recent arrays contain 128×128 and 176×144 processing elements [25], [24]. Speed and dissipation advantage are coming from the operation mode. The solution can be derived by running the transient. The drawback of this implementation is the limited accuracy (7-8 bit), noise sensitivity (fluctuation of the temperature and voltage), the moderate flexibility, the number of cells is limited (e.g., 128×128 on ACE16k, or 176×144 on eye-RIS), their cost is high, moreover the development time is long, due to the utilization of full-custom VLSI technology.

It is obvious that for those problems which can be solved by the analog (analogic) VLSI chips, the analog array dynamics of the chip outperform all the software simulators and digital hardware emulators.

The continuous valued analog dynamics when discretized in time and values can be simulated by a single microprocessor, as shown in the case of software simulators. Emulating large CNN arrays needs more computing power. The performance can be improved by using emulated digital CNN-UM architectures where small specialized processor cores are implemented. A special hardware

DOI:10.15774/PPKE.ITK.2011.003

accelerator can be implemented either on multi-processor VLSI ASIC digital emulators (e.g CASTLE) [26], on DSP-, SPE-, and GPU-based hardware accelerator boards (e.g. CNN-HAC [27], Cell Broadband Engine Architecture [28], Nvidia Cuda [29], respectively), on FPGA-based reconfigurable computing architectures (e.g. FALCON [30]), as well. Generally, they speed up the software simulators, to get higher performance, but they are slower than the analog/mixed-signal CNN-UM implementations.

A special Hardware Accelerator Board (HAB) was developed for simulating up to one-million-pixel arrays (with on-board memory) with four DSP (16 bit fixed point) chips. In fact, in a digital HAB, each DSP calculates the dynamics of a partition of the whole CNN array. Since for the calculation of the CNN dynamics a major part of DSP capability is not used, special purpose chips have been developed.

The first emulated-digital, custom ASIC VLSI CNN-UM processor – called CASTLE.v1 – was developed in MTA-SZTAKI in Analogical and Neural Computing Laboratory between 1998 and 2001 for processing binary images [26], [31]. By using full-custom VLSI design methodology, this specialized systolic CNN array architecture greatly reduced the area requirements of the processor and makes it possible to implement multiple processing elements (with distributed ALUs) on the same silicon die. The second version of the CASTLE processor was elaborated with variable computing precision (1-bit 'logical' and 6/12-bit 'bitvector' processing modes), its structure can be expanded into an array of CASTLE processors. Moreover, it is capable of processing 240×320-sized images or videos at 25fps in real-time with low power dissipation (in mW range), as well. Emulated-digital approach can also benefit from scaling-down by using new manufacturing technologies to implement smaller and faster circuits with reduced power dissipation.

Several fundamental attributes of the Falcon architecture [30] are based on CASTLE emulated-digital CNN-UM array processor architecture. However, the most important features which were greatly improved in this FPGA-based implementation are the flexibility of programming, the scalable accuracy of CNN computations, and configurable template size. Therefore, the majority of these

DOI:10.15774/PPKE.ITK.2011.003

modifications increased the performance. In case of CASTLE an important drawback was that its moderate (12-bits) precision is enough for image processing but not enough for some applications require more precise computations. Moreover, the array size is also fixed, which makes difficult to emulate large arrays, especially, when propagating cloning templates are used. To overcome these limitations configurable hardware description languages and reconfigurable devices (e.g. FPGA) were used to implement the FALCON architecture by employing rapid prototyping strategy [32]. This made it possible to increase the flexibility and create application optimized processor configurations. The configurable parameters are the following:

- bit width of the state-, constant-, and template-values,
- size of the cell array,
- number and size of templates,
- number and arrangement of the physical processor cores.

DOI:10.15774/PPKE.ITK.2011.003

1.4 Field Programmable Gate Arrays

Using reconfigurable computing (RC) and programmable logic devices for accelerating execution speed is derived from the late 1980, at the same time with the spread of the Field Programmable Gate Array (FPGA) [33]. The innovative progression of the FPGAs – which can be configured infinitely many times – lead the developments in a new line. With the help of these devices almost as many hardware algorithms can be implemented as software algorithms on conventional microprocessors.

The speed advantage of the hardware execution on FPGAs, which is practically 10-100 times faster compared to the equivalent software algorithms, sparked the interest of developers attention who are working with digital signal processors (DSP) and with other hard computational problems. The RC developers realized the fact, that with FPGAs a significant performance gain can be achieved in certain applications compared to the microprocessors, mainly in those applications which requires individual bit widths and high instruction-level parallelism. But the most important argument with the FPGA is the following: the commercially available devices evolves according to Moore's law, the FPGA, which contains a large number of SRAMs and regularly placed logical blocks, scales with the ITRS (International Technology Roadmap for Semiconductors) memory roadmap [34]. Often they are the frontrunners in the development and in the application of new manufacturing technologies. For that reason, the reconfigurable devices evolves technically faster than the microprocessors.

1.4.1 The general structure of FPGAs

The FPGA, as reconfigurable computing architecture, builds up from regularly arranged logic blocks in two-dimension. Every logic block contains a Look-Up Table (LUT), which is a simple memory and can implement an arbitrary n-input logical function. The logic blocks are communicating with each other via programmable interconnect networks, which can be neighboring, hierarchical and long-line interconnections. The FPGA contains I/O blocks too, which interfaces the internal logic blocks to the outer pads. The FPGAs evolves from the initial homogenous architectures to todays heterogenous architectures: they are

DOI:10.15774/PPKE.ITK.2011.003

containing on-chip memory blocks and DSP blocks (dedicated multipliers, and multiply/accumulator units).

Despite the fact that there are many FPGA families, the reconfigurable computers are exclusively SRAM programmable devices. That means, the configuration of the FPGA, the code as a generated 'bitfile' – which defines the device implemented algorithm – stored in an on-chip SRAM. With the loading of the configurations into the SRAM memory different algorithms can be executed efficiently. The configuration defines the logic function, which is computed by the logic blocks and the interconnection-pattern.

From the mid 1980 the FPGA designers developed a number of programmable logic structures into this architecture. The general structure is shown in Figure 1.5. This basic circuit contains programmable combination logic, synchronous flip-flop (or asynchronous latch) and some fast carry logic, for decreasing the need for area and delay dependency during the implementation. In this case the output can be chosen randomly: it can be a combination logic, or flip-flop. It can be also observed, that certain configuration uses memory for choosing the output for the multiplexer.

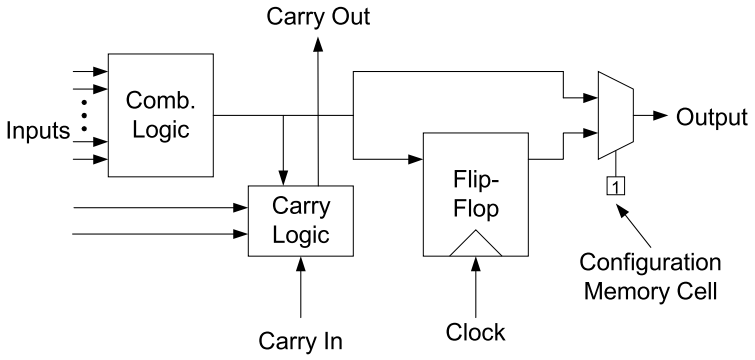


Figure 1.5: General architecture of the programmable logic block

There are a number of design methodology for the implementation of the combinational logic in the configurable logic block. Usually the configurable combinational logic is implemented with memory (Look-Up Table, LUT), but there are several architectures which uses multiplexers and logical gates instead of memories. In order to decrease the trade-offs generated by the programmable

DOI:10.15774/PPKE.ITK.2011.003

interconnections, in case of many reconfigurable FPGA architectures, the logical elements are arranged into clusters with fast and short-length wires. By using fast interconnects of clusters more complex functions with even more input can be implemented. Most LUT based architecture uses this strategy to form clusters with two ore more 4 or 6 input logical elements, which are called configurable logic block (CLB) in case of Xilinx FPGAs.

Basically the I/O architecture is the same in every FPGA family (see Figure 1.6). Mainly a tri-state buffer belongs to the output and an input buffer to the input. One by one, the tri-state enable signal, the output signal and the input signal can be registered and un-registered inside the I/O block, which depends on the method of the configuration. The latest FPGAs are extended with many new possibilities, which greatly increased the complexity of the basic structure. For example the Xilinx Virtex-6 latest I/O properties are the following:

- Supports more than 50 I/O standards with properties like the Digitally controlled impedance (DCI) active termination (for eliminating termination resistance), or the flexible fine-grained I/O banking,
- Integrated interface blocks for PCI Express 2.0 designs,
- Programmable input delays.

1.4.2 Routing Interconnect

Similarly to the structure of the logical units, the FPGA developers designed several solutions for interconnections. Basically the interconnections can be found within the cluster (for generating complex functions) and out of the cluster too. For that reason, the important properties of a connection are the following: low parasitic resistance and capacitance, requires a small chip area, volatility, reprogrammability and process complexity. Modern FPGAs are using two kind of connection architectures, namely the antifuse and the memory-based architecture.

The main property of the antifuse technology is its small area and low parasitic resistance and capacitance. It barriers the two metal layer with a non-conducting amorphous silicon. If we want to make it conductive, we have to

DOI:10.15774/PPKE.ITK.2011.003

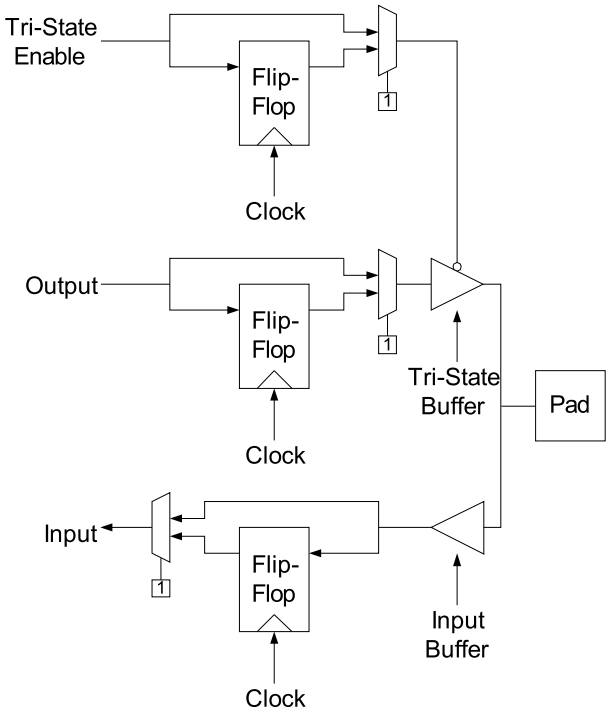


Figure 1.6: Programmable I/O architecture

DOI:10.15774/PPKE.ITK.2011.003

apply an adequate voltage to change the structure of the crystal into a low resistance polycrystalline silicon-metal alloy. The transformation could not turned back and that is why it can be programmed only once. This technology is applied by Actel [35] and QuickLogic [36].

There are several memory based interconnection structure, which are commonly used by the larger FPGA manufacturers. The main advantage of these structures over the antifuse solution is the reprogrammability. This property gives the chance to the architecture designers to make rapid development of the architectures cost efficiently. The SRAM-based technology for FPGA configuration is commonly used by Xilinx [37] and Altera [38]. It contains 6 transistors, which stores the state of the interconnection. It has a great reliability and stores its value until the power is turned off.

Three kind of basic building blocks are used in the structure of the programmable interconnections: multiplexor, pass transistor and tri-state buffer (see Figure 1.7).

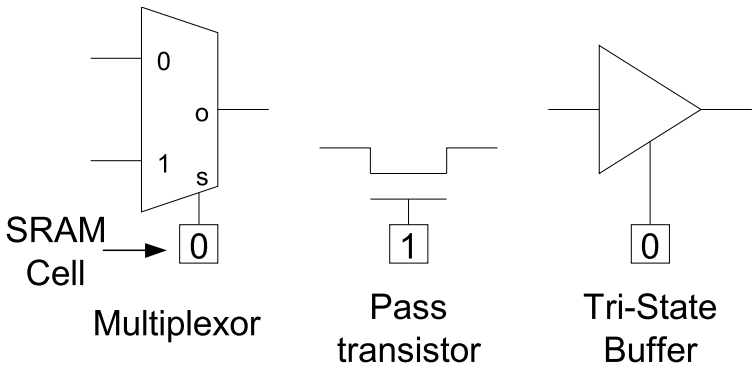


Figure 1.7: Basic programmable switches

Usually multiplexers and pass transistors are used for the interconnections of the internal logical elements and all of the above are used for the external routing. (The use of the two, eight or more input multiplexer – depending on the complexity of the interconnections – is popular among the FPGAs.) The reason of the wiring inside of a logical cluster is follows:

- implementation of low delay interconnections between the elements of the clusters,

DOI:10.15774/PPKE.ITK.2011.003

- to develop a more complex element using the elements of the clusters,
- non-programmable wiring for transferring fast carry bits for avoiding the extra delays when programmable interconnections (routing) are used.

There are three main implementations for the global routing (which are used by the Xilinx by their FPGA family): row-based, symmetric (island) type and cellular architecture. Modern FPGA architectures are using significantly complex routing architectures, but their structures are based on these three.

In case of the symmetric routing architecture the logical clusters are surrounded by segmented horizontal and vertical wiring channels, which can be seen in Figure 1.8.

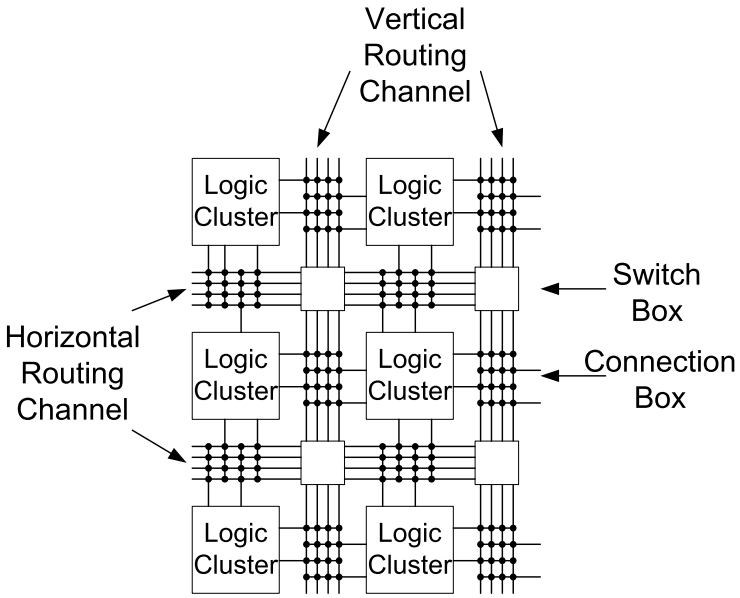


Figure 1.8: Symmetrical wiring

Every cluster connects to the channels via "connection box" and every segment in the routing can be interconnected to each other through a "switch box". The main property of this architecture is that the interconnection is made by segmented wires. This structure can be found on many Xilinx FPGA: furthermore Xilinx provides variable lengths for the segments and it provides the clusters with local connections for improving the efficiency of the architecture.

DOI:10.15774/PPKE.ITK.2011.003

The main difference between the cellular routing architecture and the symmetrical architecture is that the densest interconnections are taking place local between the logical clusters and only a few (if there is any) longer connections exists (e.g.: Xilinx XC6200, see Figure 1.9).

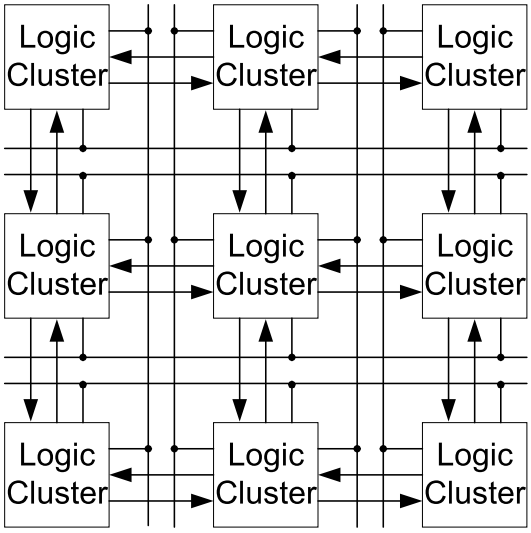


Figure 1.9: Cellular wiring

In most cases this architecture is used in fined grained FPGAs, where the clusters are relatively simple and usually are containing only one logical element. In order to make the routing process more effective, these logical cells are designed in such way, that they may take part of the routing network between other logical elements. The main drawbacks of the cellular routing architecture are the followings:

- The combination pathways, which connects not only the neighborhood, may have a huge delay.
- In case of CAD (Computer Aided Design) tools, there occurs significant problem during the efficient placement of the circuit elements and the wiring of the circuit of the architecture (place & routing).
- The area and delay requirements of the fine grained architecture are significant compared to the number of logical element and routing resource

DOI:10.15774/PPKE.ITK.2011.003

requirements for an algorithm implementation.

The importance of the last factor can be reduced if pipelining technique is used, which provides a continuous operation of the arithmetic unit.

The third type is the row-based routing architecture which can be seen in Figure 1.10.

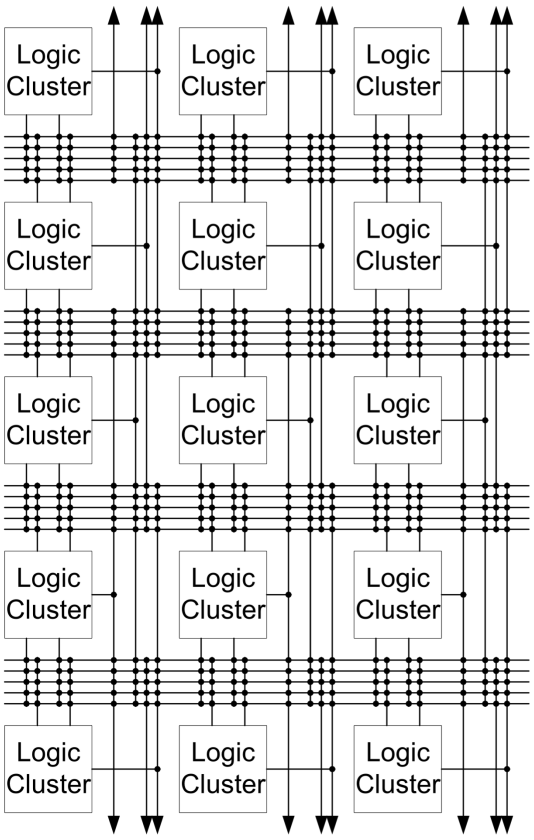


Figure 1.10: Row-based wiring

This type is mainly used in the not reprogrammable FPGA (called "one-time programmable FPGA"), that is why it is used less in todays reconfigurable systems. It uses horizontal interconnections between two logical cluster. As the figure shows there are several vertical interconnections for connecting row-based channels. The row-based architecture uses segmented wires between routing channels for decreasing the delays of the short interconnections.

DOI:10.15774/PPKE.ITK.2011.003

1.4.3 Dedicated elements, heterogenous structure

In advanced FPGAs specialized blocks were also available. These blocks (e.g.: embedded memory, arithmetic unit, or embedded microprocessor) are implemented because they are commonly used elements, therefore the specialized blocks are using less general resources from the FPGA. The result is a highly heterogenous structure.

The memory is a basic building block of the digital system. Flip-flops can be used as a memory, but it will not be efficient in case of storing large amount of data. Firstly in case of Xilinx XC4000 FPGA were the LUTs enough flexible to use it as an asynchronous 16×1 bit RAM. Later it evolved to use as a dual-ported RAM or as a shift register. These clusters can be arranged in a flexible way to implement larger bit-width or deeper memory. E.g.: In case of a 4Kb on-chip RAM on Xilinx Virtex FPGA can be defined in the following hierarchical way: 4096×1, 2048×2, 1024×4, 512×8, 256×16.

Among the adders, which builds up from logical elements, in FPGA we can use embedded multipliers, or Digital Signal Processing (DSP) blocks like separate, dedicated resources. The DSP block can make addition, subtraction, multiplication or multiply-accumulate (MAC) operation. The solving of a MAC operation in one clock cycle can be useful for finite-impulse response (FIR) filtering (which occurs in many DSP applications).

The FPGA manufacturers are integrating complete dedicated microprocessors into their devices in order to implement low bandwidth and/or complex controlling functions. With this capability (nearly) fully embedded systems can be implemented on the FPGA. The embedded processor can be found on Xilinx Virtex-II Pro, on Xilinx Virtex-4 and on Xilinx Virtex-5 FPGA. The Xilinx integrates dedicated hard processor cores (e.g.: IBM PowerPC405) into their devices (see Figure 1.11). These embedded processors are connected with on-chip SRAMs. That means without the configuration of the FPGA it can not make any useful work.

DOI:10.15774/PPKE.ITK.2011.003

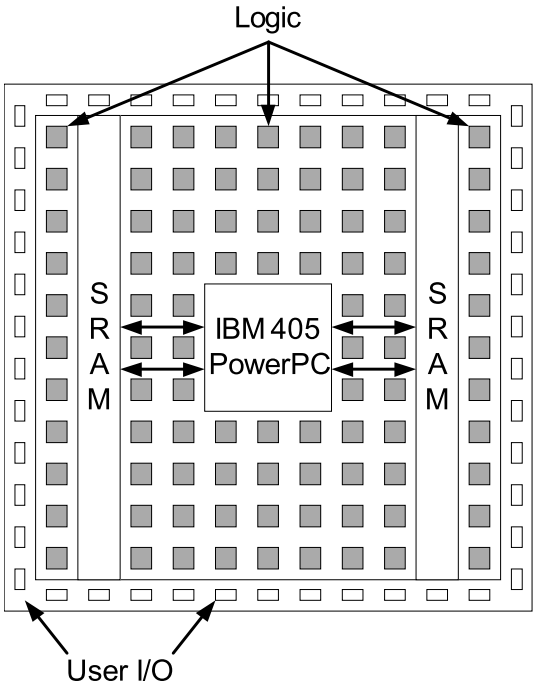


Figure 1.11: IBM PowerPC405 processor integration on Xilinx FPGA

DOI:10.15774/PPKE.ITK.2011.003

1.4.4 Xilinx FPGAs

Xilinx FPGAs are belonging to the leading reconfigurable computers long ago. Due to the fast Configurable Logic Blocks (CLB) and the large number of interconnections arbitrary circuits can be implemented on it. In order to accelerate certain operations dedicated elements (e.g.: digital signal processing (DSP) blocks) are available on the FPGA. Throughout the dissertation all the used FPGA platforms are made by Xilinx. In the next few sections the used FPGAs are going to introduced and a short outlook of the newest and future Xilinx FPGAs are going to be shown.

1.4.4.1 Xilinx Virtex 2 FPGA

The first thing, which was implemented on the XC2V3000 Xilinx FPGA, was the control unit (see in later chapters). The Virtex-II series FPGAs were introduced in 2000. It was manufactured with 0.15 μm 8-layer metal process with 0.12 μm high-speed transistors. Combining a wide variety of flexible features and a large range of component densities up to 10 million system gates, the Virtex-II family enhances programmable logic design capabilities and is a powerful alternative to mask-programmed gates arrays.

There are several improvements compared to the former FPGAs. These improvements include additional I/O capability by supporting more I/O standards, additional memory capacity by using larger 18Kbit embedded block memories, additional routing resources and embedded 18×18 bit signed multiplier blocks.

The XC2V3000 contains 3 million system gates which are organized to 64×56 array forming 14,336 slices. With the 96 18Kb SelectRAM blocks it can provide a maximum of 1,728 Kbit RAM. The block SelectRAM memory resources are dual-port RAM, programmable from 16K x 1 bit to 512 x 36 bits, in various depth and width configurations. Block SelectRAM memory is cascadable to implement large embedded storage blocks. It has also 96 18×18 bit signed multiplier blocks for accelerating multiplications.

The IOB, CLB, block SelectRAM, multiplier, and Digital Clock Management (DCM) elements all use the same interconnect scheme and the same access to the global routing matrix. There are a total of 16 global clock lines, with eight

DOI:10.15774/PPKE.ITK.2011.003

available per quadrant. In addition, 24 vertical and horizontal long lines per row or column as well as massive secondary and local routing resources provide fast interconnect. Virtex-II buffered interconnects are relatively unaffected by net fanout and the interconnect layout is designed to minimize crosstalk. Horizontal and vertical routing resources for each row or column include 24 long lines, 120 hex (which connects every 6^{th} block) lines, 40 double lines (which connects every second block), 16 direct connect lines.

1.4.4.2 Xilinx Virtex 5 FPGAs

The fifth generation of the Xilinx Virtex 5 FPGA is built on a 65-nm copper process technology. The ASMBL (Advanced Silicon Modular Block) architecture is a design methodology that enables Xilinx to rapidly and cost-effectively assemble multiple domain-optimized platforms with an optimal blend of features. This multi-platform approach allows designers to choose an FPGA platform with the right mix of capabilities for their specific design. The Virtex-5 family contains five distinct platforms (sub-families) to address the needs of a wide variety of advanced logic designs:

- The LX Platform FPGAs are optimized for general logic applications and offer the highest logic density and most cost-effective high-performance logic and I/Os.
- The SX Platform FPGAs are optimized for very high-performance signal processing applications such as wireless communication, video, multimedia and advanced audio that may require a higher ratio of DSP slices.
- The FX Platform FPGAs are assembled with capabilities tuned for complex system applications including high-speed serial connectivity and embedded processing, especially in networking, storage, telecommunications and embedded applications.

The above three is available with serial transceiver too.

Virtex-5 FPGAs contain many hard-IP system level blocks, like the 36-Kbit block RAM/FIFOs, 25×18 DSP slices, SelectI/O technology, enhanced clock

DOI:10.15774/PPKE.ITK.2011.003

management, and advanced configuration options. Additional platform dependent features include high-speed serial transceiver blocks for serial connectivity, PCI Express Endpoint blocks, tri-mode Ethernet MACs (Media Access Controllers), and high-performance PowerPC 440 microprocessor embedded blocks.

The XC5VSX95T contains 14,720 slices which builds up from 6-input LUTs instead of 4-input LUTs as in the previous generations. With the 488 18Kb SelectRAM blocks it can provide a maximum of 8,784 Kbit RAM. The block SelectRAM memory resources can be treated as a single or a dual-port RAM, in this case only 244 blocks are available, in various depth and width configurations. Instead of multipliers it uses 640 DSP48E 18×25 bit slices for accelerating multiplications and multiply-accumulate operations.

The XC5VSX240T contains 37,440 Virtex-5 slices. With the 1,032 18Kb single-ported SelectRAM blocks, or 516 36Kb dual-ported SelectRAM blocks it can provide a maximum of 18,576 Kbit RAM in various depth and width configurations. It has also 1056 DSP48E bit slices.

1.4.4.3 The capabilities of the modern Xilinx FPGAs

Built on a 40 nm state-of-the-art copper process technology, Virtex-6 FPGAs are a programmable alternative to custom ASIC technology.

The look-up tables (LUTs) in Virtex-6 FPGAs can be configured as either 6-input LUT (64-bit ROMs) with one output, or as two 5-input LUTs (32-bit ROMs) with separate outputs but common addresses or logic inputs. Each LUT output can optionally be registered in a flip-flop. Four such LUTs and their eight flip-flops as well as multiplexers and arithmetic carry logic form a slice, and two slices form a configurable logic block (CLB).

The advanced DSP48E1 slice contains a 25 x 18 multiplier, an adder, and an accumulator. It can optionally pipelined and a new optional pre-adder can be used to assist filtering applications. It also can cascaded due to the dedicated connections.

It has integrated interface blocks for PCI Express designs compliant to the PCI Express Base Specification 2.0 with x1, x2, x4, or x8 lane support per block.

DOI:10.15774/PPKE.ITK.2011.003

The largest DSP-optimized Virtex-6 FPGA is the XC6VSX475T, which contains 74,400 slices. With the 2,128 18Kb single-ported SelectRAM blocks, or 1,064 36Kb dual-ported SelectRAM blocks it can provide a maximum of 38,304 Kbit RAM in various depth and width configurations. It has also 2,016 DSP48E1 slices.

7th generation Xilinx FPGAs are manufactured with the state-of-the-art, high-performance, low-power (HPL), 28 nm, high-k metal gate (HKMG) process technology. All 7 series devices share a unified fourth-generation Advanced Silicon Modular Block (ASMBL^{TM}) column-based architecture that reduces system development and deployment time with simplified design portability.

The innovative Stacked Silicon Interconnect (SSI) technology enables multiple Super Logic Regions (SLRs) to be combined on a passive interposer layer, to create a single FPGA with more than ten thousand inter- SLR connections, providing ultra-high bandwidth connectivity with low latency and low power consumption. There are two types of SLRs used in Virtex-7 FPGAs: a logic intensive SLR and a DSP/blockRAM/transceiver-rich SLR.

The largest 7th series Xilinx FPGA will contain almost 2 million logic cells forming more than 300,000 slices. It will embed 85Mb blockRAM. The largest DSP optimized Virtex-7 FPGA will contain 5,280 ExtremeDSP48 DSP processors providing 6,737GMACS operations. The total transceiver bandwidth (full duplex) will be 2,784Gb/s. It will also support the latest gen3x8 PCI Express interface and will contain maximum 1,200 I/O pins.

Virtex-7 FPGAs are ideally suited for highest performance wireless, wired, and broadcast infrastructure equipment, aerospace and defense systems, high-performance computing, as well as ASIC prototyping and emulation.

DOI:10.15774/PPKE.ITK.2011.003

1.5 IBM Cell Broadband Engine Architecture

1.5.1 Cell Processor Chip

The Cell Broadband Engine Architecture (CBEA) [39] 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 1.12.

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. IT has a 32kB Level 1 (L1) cache memory, which is a set-associative, parity protected, 128 bit sized cache-line memory, and 512kB Level 2 (L2) unified (data and instruction) cache memory.

The Power Processing Element contains several functional units, which composes the Power Processing Unit shown in Figure 1.13.

The PPU executes the PowerPC Architecture instruction set and the Vector/SIMD Multimedia Extension instructions. The Instruction Unit performs the instruction-fetch, decode, dispatch, issue, branch, and completion portions of execution. It contains the L1 instruction cache. The Load Store Unit performs all data accesses, including execution of load and store instructions. It contains the L1 data cache. The Vector/Scalar Unit includes a Floating-Point Unit (FPU) and a 128-bit Vector/SIMD Multimedia Extension Unit (VXU), which together

[1] SIMD - Single Instruction Multiple Data

DOI:10.15774/PPKE.ITK.2011.003

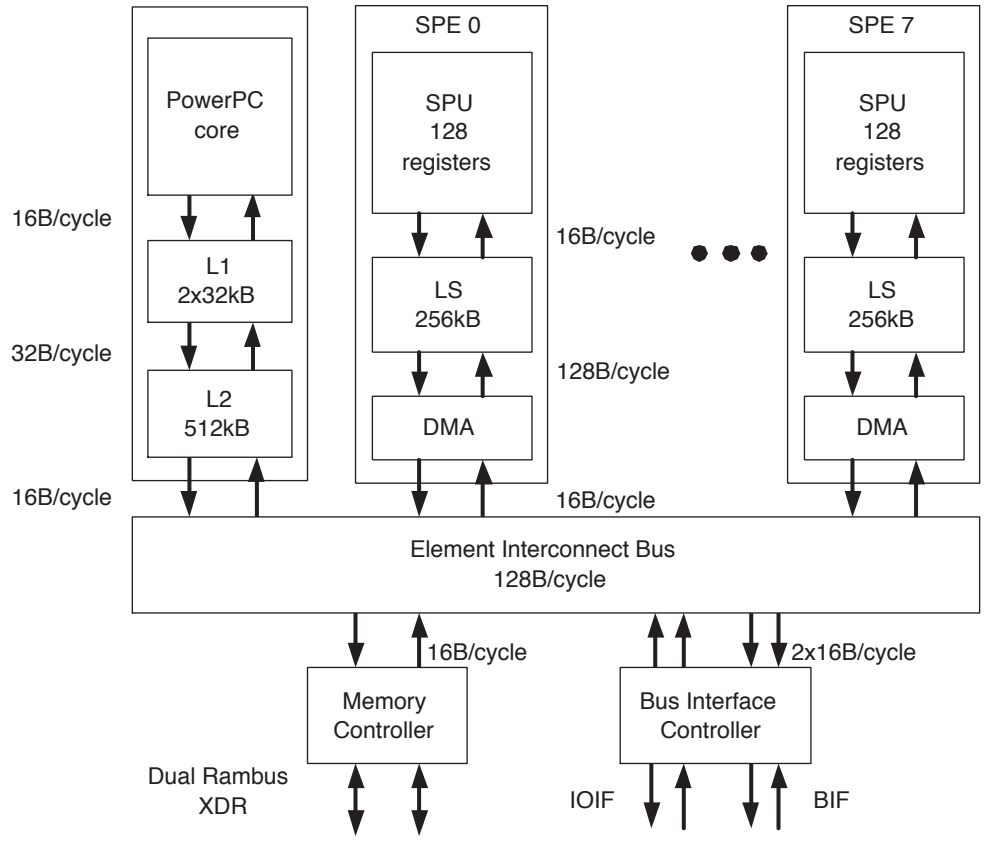


Figure 1.12: Block diagram of the Cell processor

DOI:10.15774/PPKE.ITK.2011.003

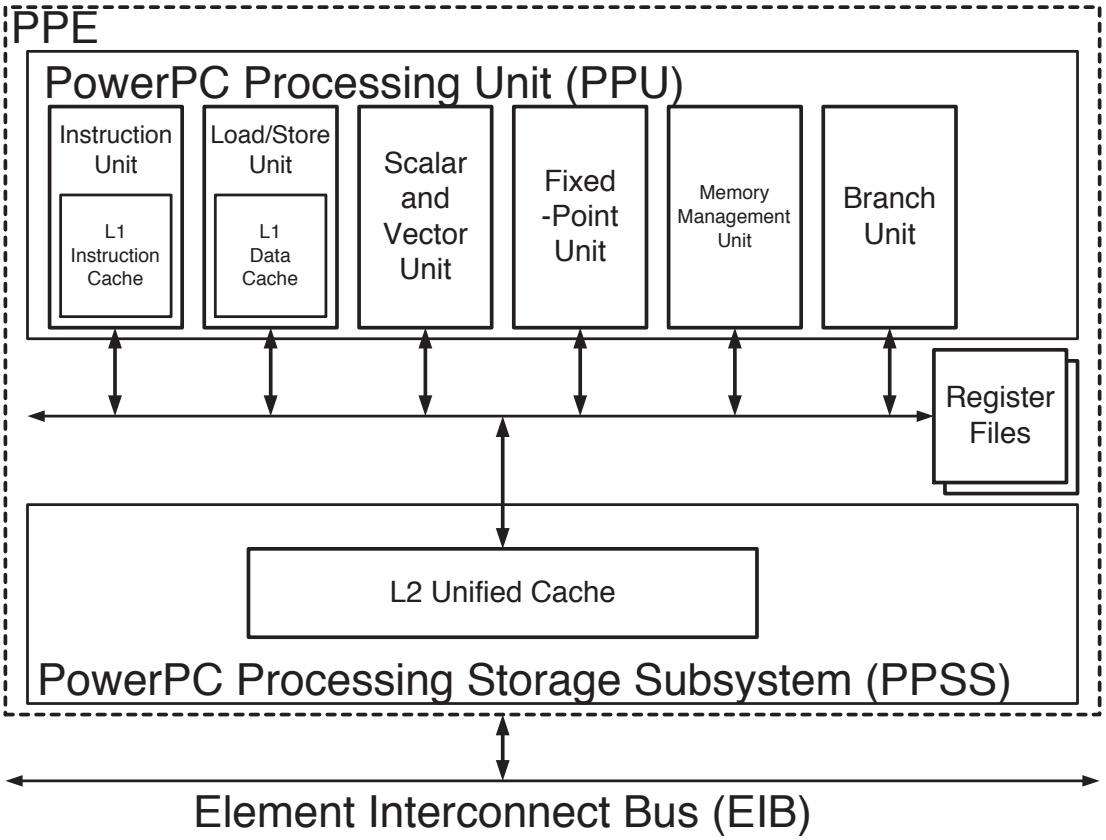


Figure 1.13: Block diagram of the functional units of the PowerPC Processor Element

DOI:10.15774/PPKE.ITK.2011.003

execute floating-point and Vector/SIMD Multimedia Extension instructions. The Fixed-point Unit executes fixed-point operations, including add, multiply, divide, compare, shift, rotate, and logical instructions. The Memory Management Unit manages address translation for all memory accesses.

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 EIB supports full memory-coherent and symmetric multiprocessor (SMP) operations. Thus, a CBE processor is designed to be ganged coherently with other CBE processors to produce a cluster. The EIB consists of four 16-byte-wide data rings. Each ring transfers 128 bytes at a time. Processor elements can drive and receive data simultaneously. The EIBŐs internal maximum bandwidth is 96 bytes per processor-clock cycle. Multiple transfers can be in-process concurrently on each ring, including more than 100 outstanding DMA memory requests between main storage and the SPEs.

The on-chip Memory Interface Controller (MIC) provides the interface between the EIB and physical memory. It supports one or two Rambus Extreme Data Rate (XDR) memory interfaces, which together support between 64 MB and 64 GB of XDR DRAM memory. Memory accesses on each interface are 1 to 8, 16, 32, 64, or 128 bytes, with coherent memory-ordering. Up to 64 reads and 64 writes can be queued. The resource-allocation token manager provides feedback about queue levels.

The dual-channel Rambus XDR memory interface provides very high 25.6GB/s memory bandwidth. The XDR DRAM memory is ECC-protected, with multibit error detection and optional single-bit error correction. 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 1.14.

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

DOI:10.15774/PPKE.ITK.2011.003

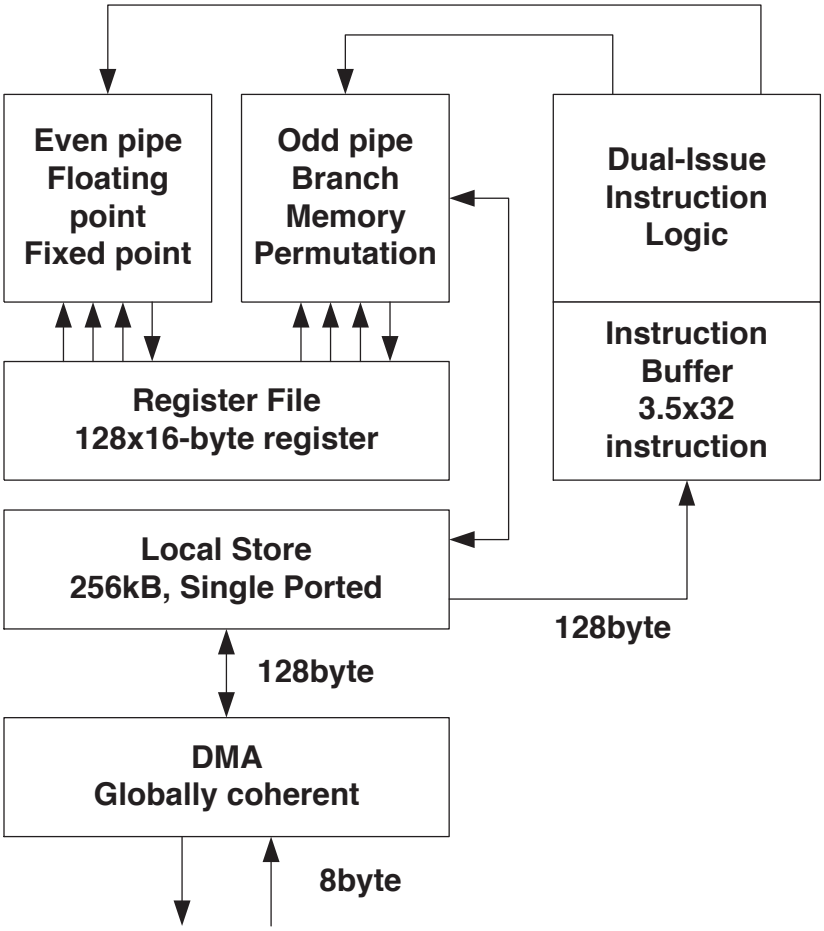


Figure 1.14: Block diagram of the Synergistic Processor Element

DOI:10.15774/PPKE.ITK.2011.003

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.

Consequently, the most significant difference between the SPE and PPE lies in how they access memory. The PPE accesses main storage (the effective-address space) with load and store instructions that move data between main storage and a private register file, the contents of which may be cached. The SPEs, in contrast, access main storage with Direct Memory Access (DMA) commands that move data and instructions between main storage and a private local memory, called a local store or local storage (LS). An SPE's instruction-fetches and load and store instructions access its private LS rather than shared main storage, and the LS has no associated cache. This three-level organization of storage (register file, LS, main storage), with asynchronous DMA transfers between LS and main storage, is a radical break from conventional architecture and programming models, because it explicitly parallelizes computation with the transfers of data and instructions that feed computation and store the results of computation in main storage.

DOI:10.15774/PPKE.ITK.2011.003

1.5.2 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 1.15) 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 maximum 7 Blades per chassis.

The second generation blade system is the IBM Blade Center QS21which is equipped with two Cell processor chips, 1GB XDRAM (512MB per processor) memory, Gigabit Ethernet, and 4x InfiniBand I/O capability. Several QS21 may be interconnected in a Blade Center chassis with max. 6.4TFLOPS peak computing power. The third generation blade system is the IBM Blade Center QS22 equipped with new generation PowerXCell 8i processors manufactured using 65nm technology. Double precision performance of the SPEs are significantly improved providing extraordinary computing density up to 6.4 TFLOPS single precision and up to 3.0 TFLOPS double precision in a single Blade Center house. These blades are the main building blocks of the world's fastest supercomputer (2009) at Los Alamos National Laboratory which first break through the "petaflop barrier" of 1,000 trillion operations per second. The main building blocks of the world's fastest supercomputer, besides the AMD Opteron X64 cores, are the Cell processors. The Cell processors produce 95% computing power of the entire system (or regarding computational task), while the AMD processors

DOI:10.15774/PPKE.ITK.2011.003

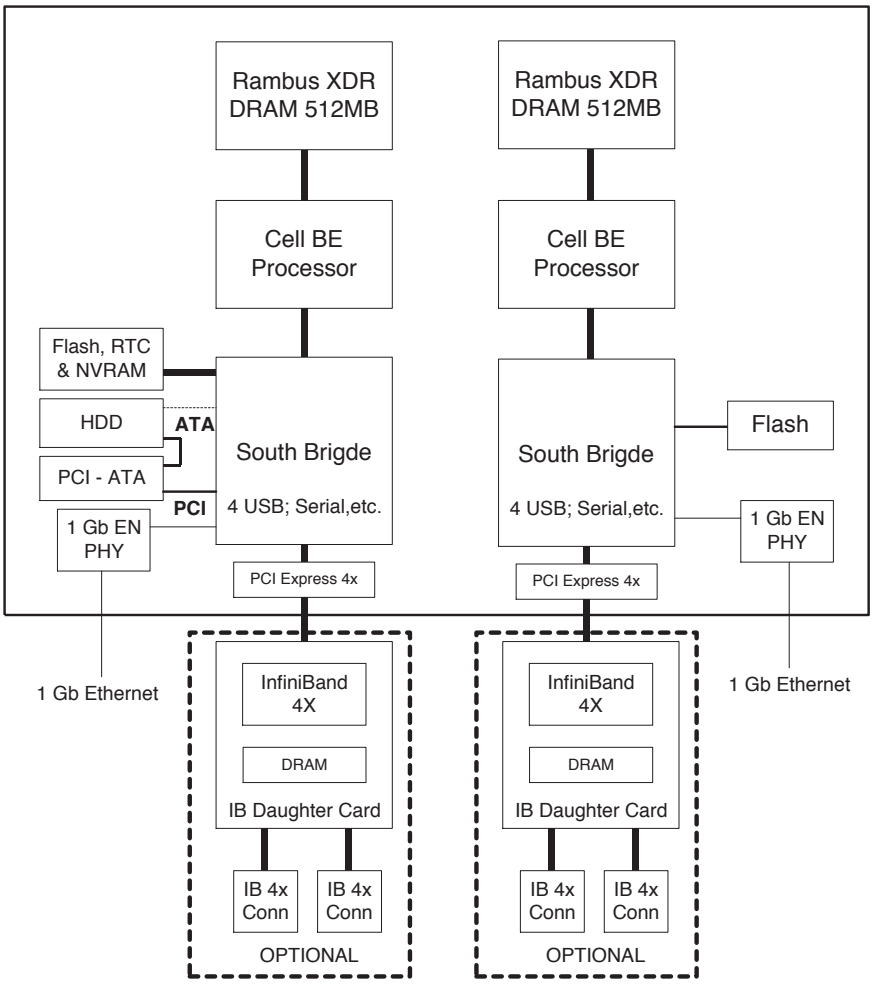


Figure 1.15: IBM Blade Center QS20 architecture

mounted on LS22 board are for supporting internode communication. The peak computing performance is 6.4 TFLOPS single precision and up to 3.0 TFLOPS double precision in a single Blade Center house.

DOI:10.15774/PPKE.ITK.2011.003

1.6 Recent Trends in Many-core Architectures

There are a number of different implementations of array processors commercially available. The CSX600 accelerator chip from Clearspeed Inc. [40] 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 [41] 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 MathStar's Silicon Object Assembly Language.

The Tilera Tile64 architecture [42] 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.

DOI:10.15774/PPKE.ITK.2011.003

Chapter 2

Mapping the Numerical Simulations of Partial Differential Equations

2.1 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 fully all of the available integer and floating point execution units.

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.

In this work the IBM Cell heterogeneous array processor architecture (mainly because its development system is open source), and an FPGA based implementations is investigated. It is exploited here in solving complex, time consuming problems.

The main motivation of the Chapter is to find a method for implementing

DOI:10.15774/PPKE.ITK.2011.003

computationally hard problems on different many-core architectures. Such a hard problem is the implementation of the numerical simulation of Partial Differential Equations, which are used in wide range of engineer applications. One of the toughest PDE is the Navier-Stokes equations, which describes the temporal evolution of fluids.

Emulated-digital CNN are proven to be a good alternative for solving PDEs [43, 44]. In this Chapter a CNN simulation kernel is implemented on a heterogenous Cell architecture and an improved emulated-digital CNN processor is evolved from the Falcon processor on FPGA for solving PDEs. During the implementation of different PDEs on different architecture, the difference between the two platforms are investigated and their performance are compared. Two questions formed during the implementation, namely: *How to map a computationally hard problem on a heterogenous processor architecture and on a reconfigurable architecture (FPGA)? What is the difference in performance between the inhomogenous and the custom architecture?* In the next few section the answer and the method for its investigation are going to be described.

2.1.1 How to map CNN array to Cell processor array?

The primary goal is to get an efficient CNN [17] implementation on the Cell architecture. Because analog CNN architectures are effective solving partial differential equations. The analog CNN chips has limitations (limited precision, only linear templates, sensitive to the environmental noises) and that is why it can not be used in real life applications. A SIMD (Single Instruction Multiple Date) architecture can be implemented efficiently in CNN. Consider the CNN model and its hardware effective discretization in time. With the emulated digital CNN-UM these drawbacks can be neglected.

2.1.1.1 Linear Dynamics

The computation of the discretized version of the original CNN state equations (1.5a) and (1.5b) 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,

DOI:10.15774/PPKE.ITK.2011.003

18 Load instructions are required for loading the templates, 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 128×128 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 are 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.

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 2.1). 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.

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 2.2. 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

DOI:10.15774/PPKE.ITK.2011.003

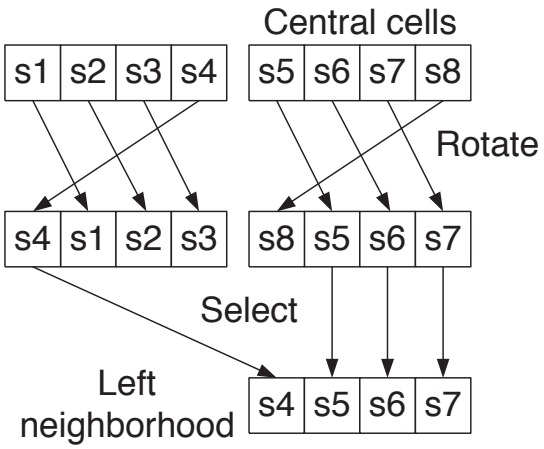


Figure 2.1: Generation of the left neighborhood

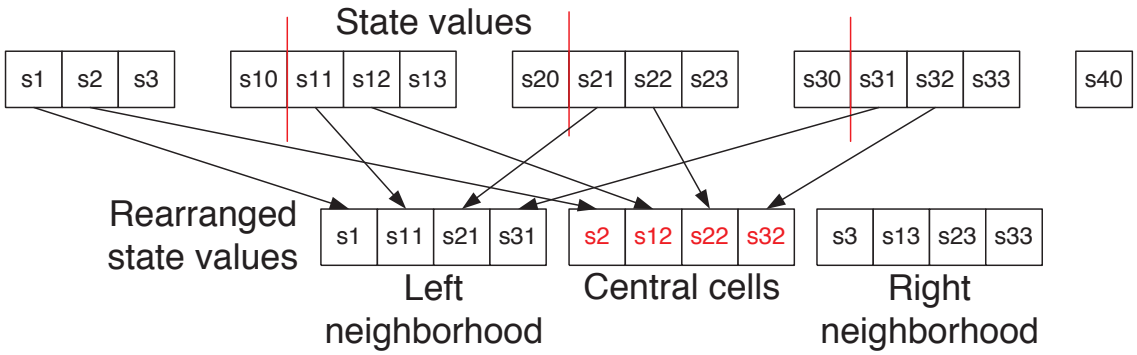


Figure 2.2: Rearrangement of the state values

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.

To measure the performance of the simulation a 256×256 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 tim-

DOI:10.15774/PPKE.ITK.2011.003

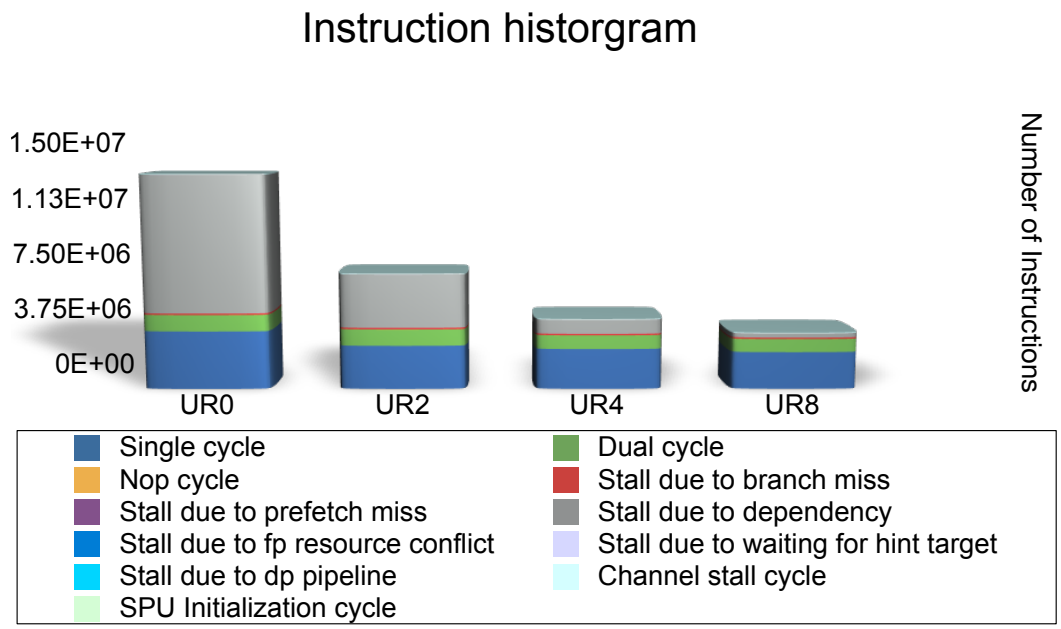


Figure 2.3: Results of the loop unrolling

ing of the program can be created where the pipeline stalls can be identified. The instruction histogram is shown in Figure 2.3. 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, with the cost of larger area, 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 2.4. the SPE is nearly one order faster than a high performance desktop microprocessor. The Falcon processor can outperform the Cell if it is implemented on a large FPGA. On a Xilinx Virtex-5 SX95T more than 70 Falcon PE can be implemented.

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

DOI:10.15774/PPKE.ITK.2011.003

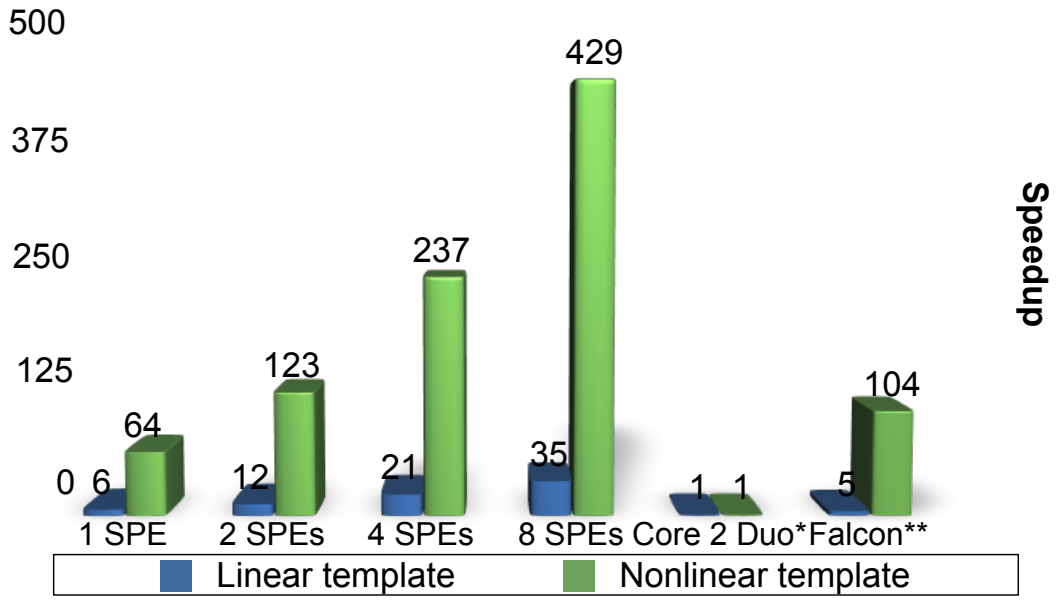


Figure 2.4: Performance of the implemented CNN simulator on the Cell architecture compared to other architectures, considering the speed of the Intel processor as a unit in both linear and nonlinear case (CNN cell array size: 256×256, 16 forward Euler iterations, *Core 2 Duo T7200 @2GHz, **Falcon Emulated Digital CNN-UM implemented on Xilinx Virtex-5 FPGA (XC5VSX95T) @550MHz only one Processing Element (max. 71 Processing Element).

DOI:10.15774/PPKE.ITK.2011.003

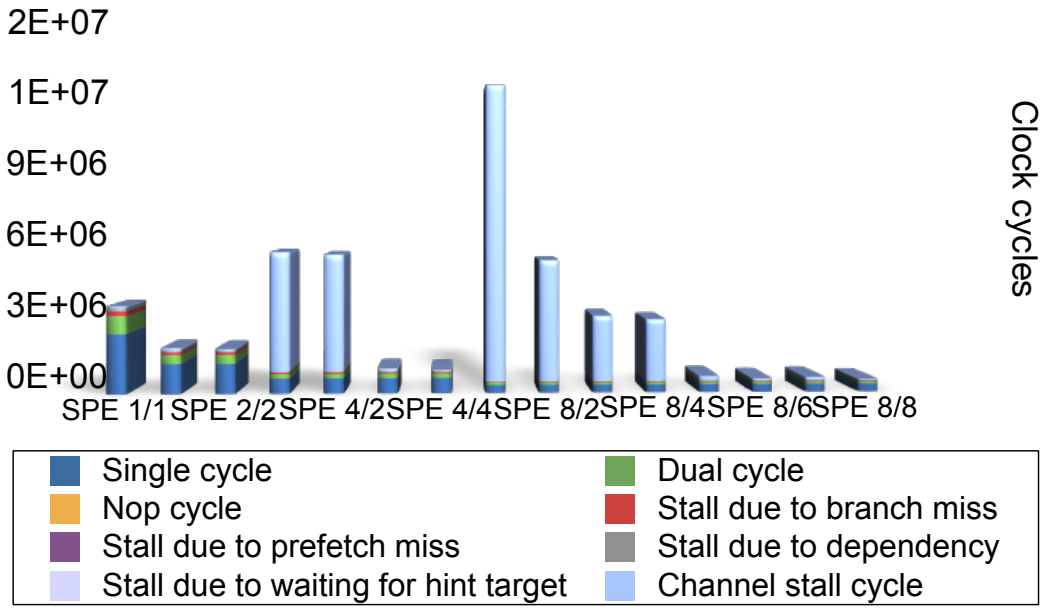


Figure 2.5: Intruction histogram in case of one and multiple SPEs

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 256×256 sized cell array. The number of required clock cycles is summarized in Figure 2.5. 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

DOI:10.15774/PPKE.ITK.2011.003

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.

To reduce this high bandwidth requirement pipelining technique can be used as shown in Figure 2.6. 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 2.7.

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 2.7 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 2.8.

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.

DOI:10.15774/PPKE.ITK.2011.003

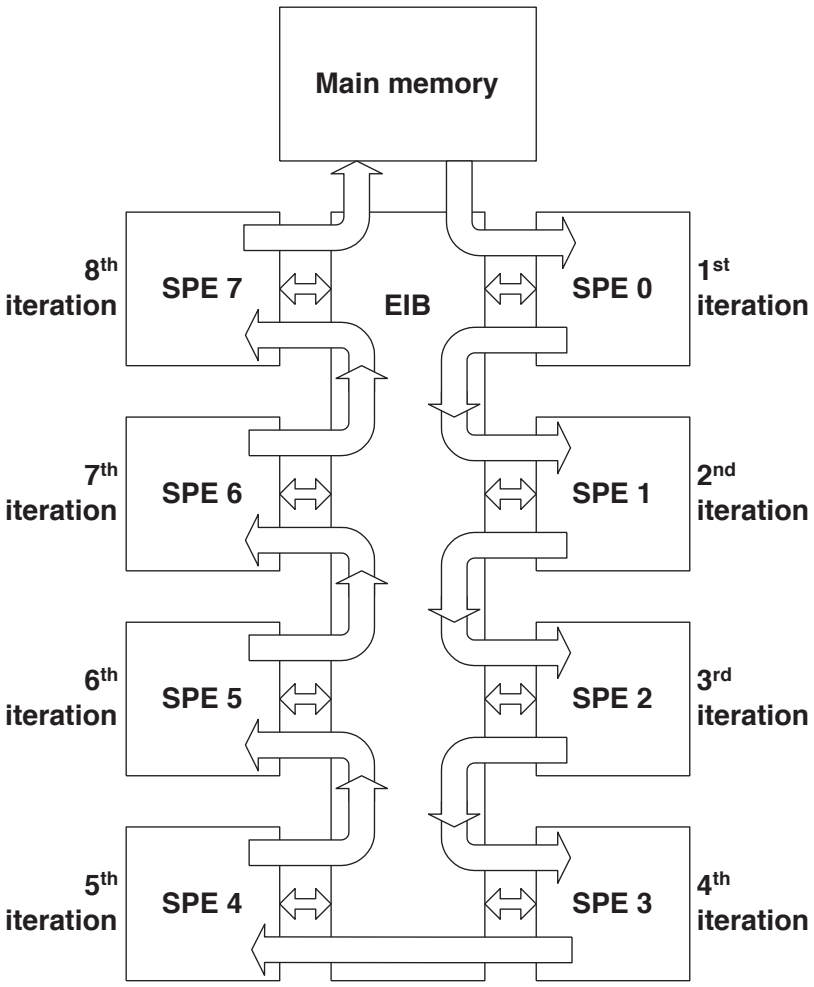


Figure 2.6: Data-flow of the pipelined multi-SPE CNN simulator

DOI:10.15774/PPKE.ITK.2011.003

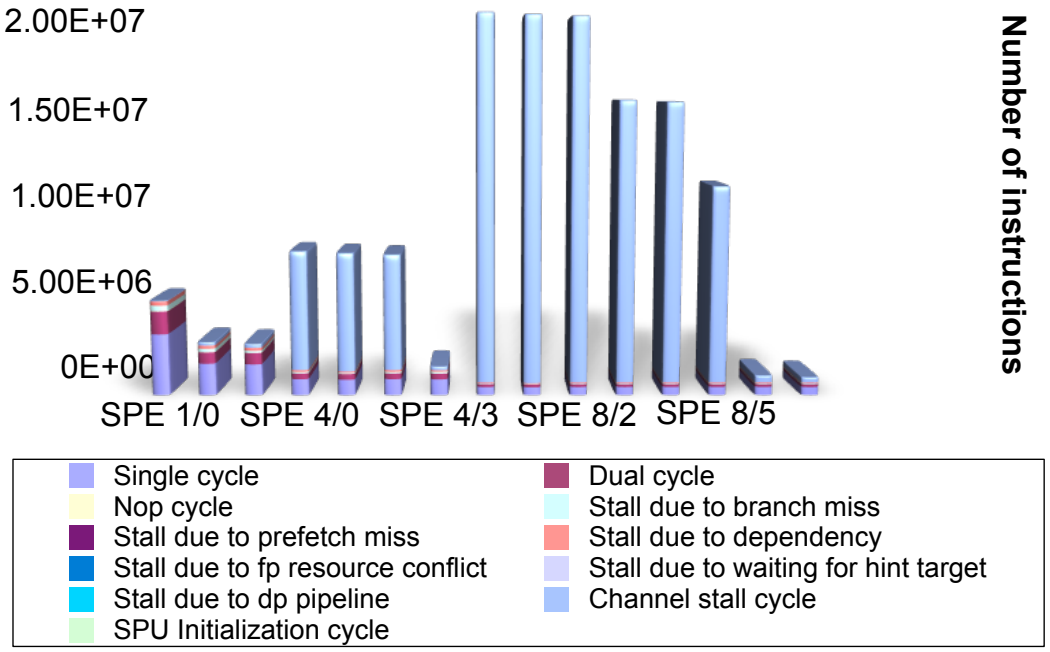


Figure 2.7: Instruction histogram in case of SPE pipeline

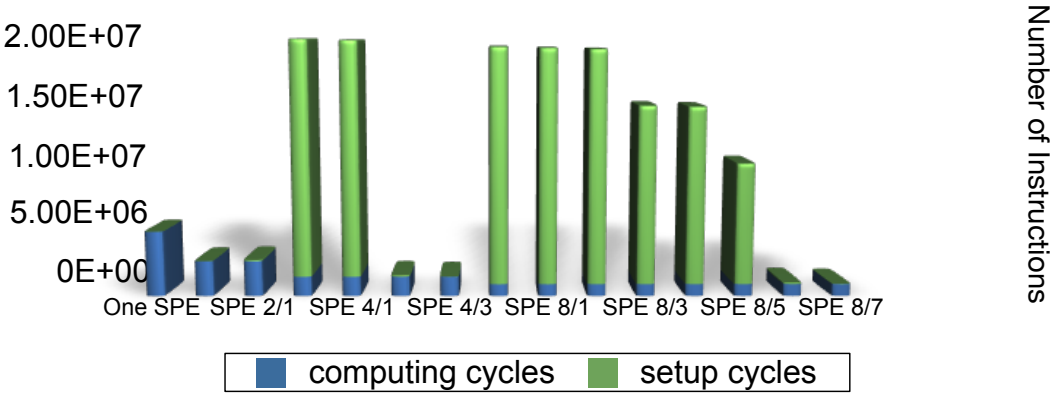


Figure 2.8: Startup overhead in case of SPE pipeline

DOI:10.15774/PPKE.ITK.2011.003

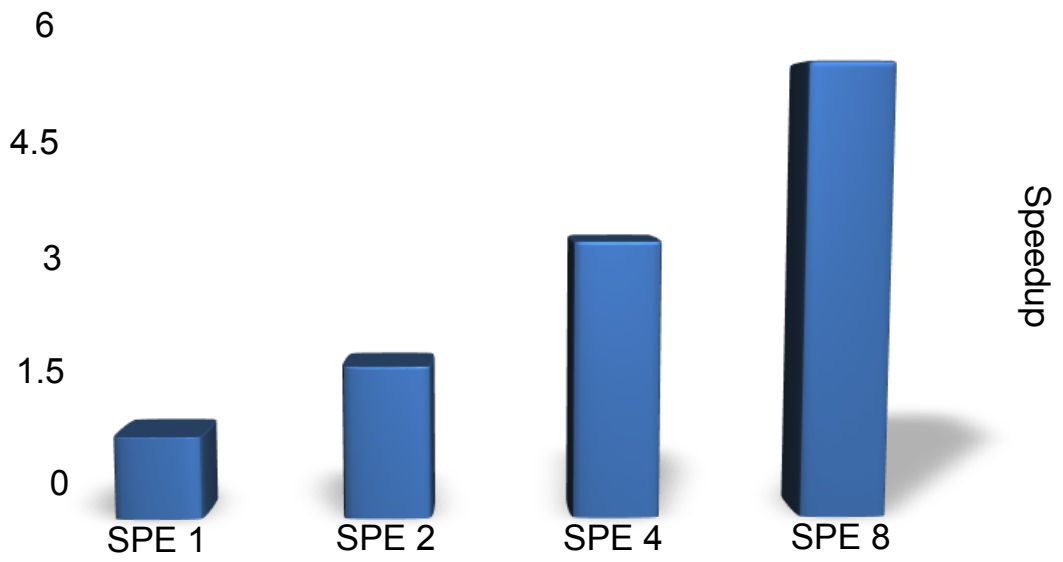


Figure 2.9: Speedup of the multi-SPE CNN simulation kernel

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 2.9. 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.

2.1.1.2 Nonlinear Dynamics

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

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

DOI:10.15774/PPKE.ITK.2011.003

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 256×256 sized cell array and 16 forward Euler iterations were used as in linear case. The global maximum finder template [21] 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 were 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 2.10. 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.

In addition, the effect of increasing the number of segments of the nonlinearity was investigated. Our experiments show that the number of required instructions

DOI:10.15774/PPKE.ITK.2011.003

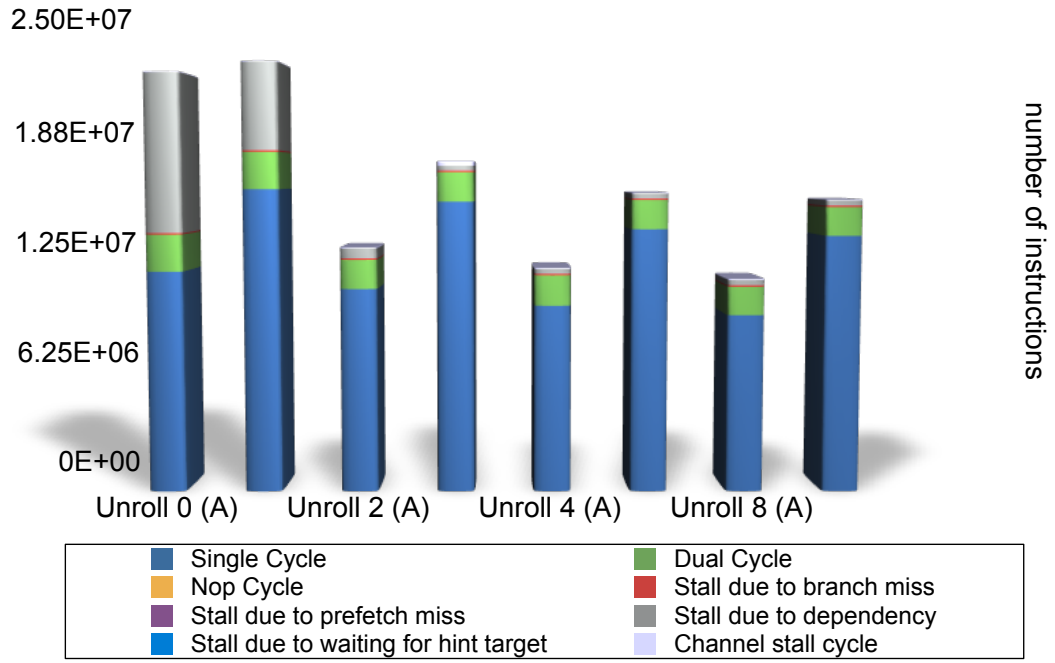


Figure 2.10: Comparison of the instruction number in case of different unrolling

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 2.10. In the four segment case (B) the number of stall cycles is smaller than in the two segment case (A) 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 2.11.

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

DOI:10.15774/PPKE.ITK.2011.003

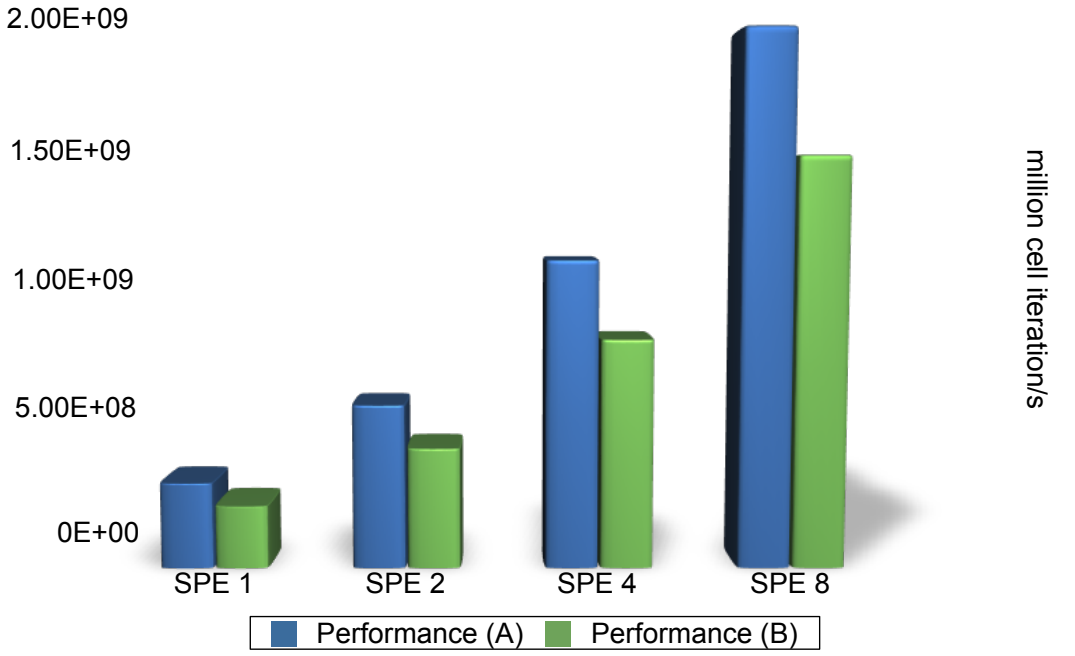


Figure 2.11: Performance comparison of one and multiple SPEs

almost 24% more than if the nonlinearity is partitioned into four parts (B) as seen in Figure 2.11. 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.

2.1.1.3 Performance comparisons

The performance of the implementation on the Cell architecture was tested by running the global maximum finder template on a 256×256 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 2.4. 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

DOI:10.15774/PPKE.ITK.2011.003

Table 2.1: Comparison of different CNN implementations: 2GHz CORE 2 DUO processor, Emulated Digital CNN running on Cell processors and on Virtex 5 SX240T FPGA, and Q-EYE with 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: 176×144, 16 forward Euler iterations)

can outperform a single Falcon Emulated Digital CNN-UM core implemented on XC5VSX240T Xilinx FPGA. 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 2.1. Let us mention that solution of a given computational problem can be much faster implemented in a Cell architecture then on FPGA.

2.2 Ocean model and its implementation

Several studies proved the effectiveness of the CNN-UM solution of different PDEs [15] [16]. But the results cannot be used in real life implementations due to 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 [45] [28] and in the solution of PDEs [46] [47] [30]. Using the CNN simulation kernel described in the previous sections helped to solve Navier-Stokes PDE on the Cell architecture.

DOI:10.15774/PPKE.ITK.2011.003

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 [48, 49, 50]. 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) [51] 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:

$$\frac{\partial \eta}{\partial t} + \frac{\partial \overline{U} D}{\partial x} + \frac{\partial \overline{V} D}{\partial y} = 0 \tag{2.1a}$$

$$\frac{\partial \overline{U} D}{\partial t} + \frac{\partial \overline{U}^2 D}{\partial x} + \frac{\partial \overline{UV} D}{\partial y} - f\overline{V} D + gD\frac{\partial \eta}{\partial x} = - < wu(0) > + < wu(-1) > - \\ - \frac{gD}{\rho_0} \int_{-1}^{0} \int_{\sigma}^{0} \left[D\frac{\partial \rho'}{\partial x} - \frac{\partial D}{\partial x}\sigma' \frac{\partial \rho'}{\partial \sigma} \right] d\sigma' d\sigma \tag{2.1b}$$

$$\frac{\partial \overline{V} D}{\partial t} + \frac{\partial \overline{UV} D}{\partial x} + \frac{\partial \overline{V}^2 D}{\partial y} + f\overline{U} D + gD\frac{\partial \eta}{\partial y} = - < wv(0) > + < wv(-1) > - \\ - \frac{gD}{\rho_0} \int_{-1}^{0} \int_{\sigma}^{0} \left[D\frac{\partial \rho'}{\partial y} - \frac{\partial D}{\partial y}\sigma' \frac{\partial \rho'}{\partial \sigma} \right] d\sigma' d\sigma \tag{2.1c}$$

DOI:10.15774/PPKE.ITK.2011.003

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 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.

The solution of equations (2.1a)-(2.1c) is based on the freely available Fortran source code of the POM [51]. The discretization in space is done according to the Arakawa-C [52] 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 surface is $1024 km^2$, the north and south boundaries are closed, the east and west boundaries are

DOI:10.15774/PPKE.ITK.2011.003

open, the grid size is 128×128 and the grid resolution is 8km. The simulation timestep is 6s and 360 iterations are computed. The results are shown in Figure 2.2. Experimental results of the average iteration time are summarized on Table 2.3.

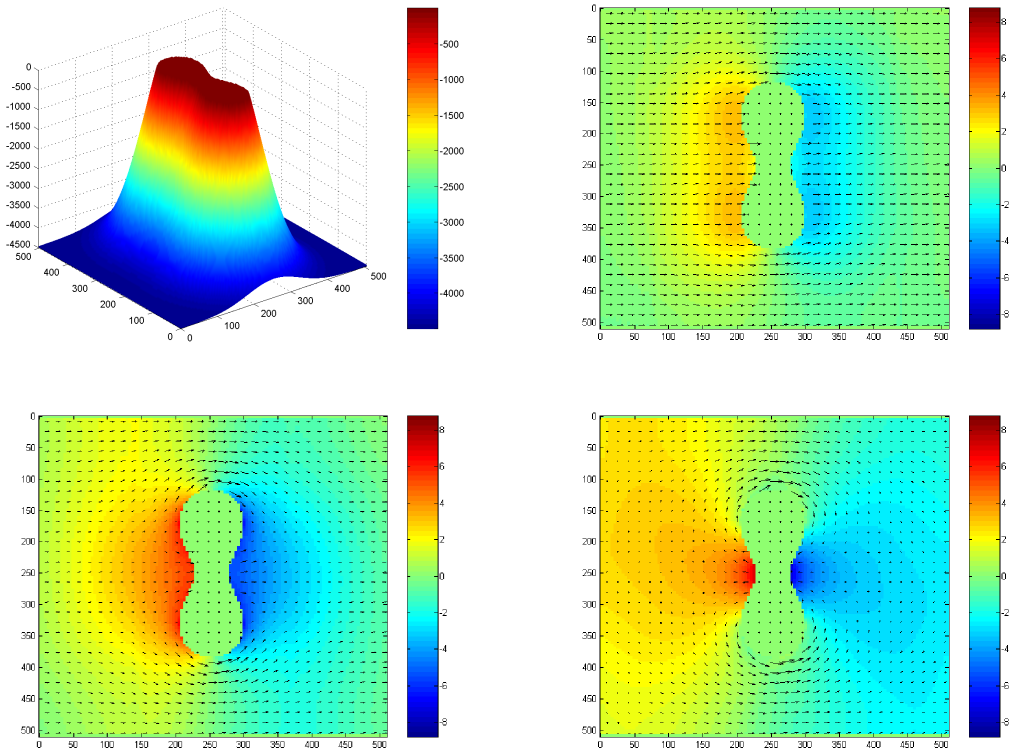


Table 2.2: The initial state, and the results of the simulation after a couple of iteration steps. Where the x- and y-axis models 1024km width ocean (1 untit is equal to 2.048km).

DOI:10.15774/PPKE.ITK.2011.003

Table 2.3: Comparison of different CNN ocean model implementations: 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
Computation time of a 72 hour simulation (s)	354.2	47.52
Power (W)	65	85
Area (mm^2)	143	253

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

2.3 Computational Fluid Flow Simulation on Body Fitted Mesh Geometry with IBM Cell Broadband Engine and FPGA Architecture

2.3.1 Introduction

In the paper of Kocsárdi et. al the authors applied the finite volume Lax-Friedrich [53, 54] scheme for solving 2D Euler equations over uniformly spaced rectangular meshes [55]. However, most real life applications of CFD require handling more complex geometries, bounded by curved surfaces. A popular and often an efficient solution to this problem is to perform the computation over non-uniform, logically structured grids. Technically, this idea can be exploited either by employing body fitted grids or by performing the computation in a curvilinear coordinate frame following the curvature of the boundaries. Although in the latter approach the standard 2D scheme over Cartesian geometry can be put to work, it is computationally much more demanding than the former one, due to the expensive operations related to coordinate transformation.

There is a number of reasons why we have chosen the IBM Cell multiprocessor system as the restricted (bounded) architecture, namely, its high computing performance, the double-precision floating point number support, the support of the standard multiprocessing libraries, such as OpenMP or MPI and the freely available software development kit.

DOI:10.15774/PPKE.ITK.2011.003

On the other hand, the development time of an optimized software solution is much shorter than designing a reconfigurable architecture [56][57], however, the computational efficiency is smaller in terms of area and power. Namely on FPGAs we can make a more specific structure for the CFD with better performance in terms of the area and dissipation with a variable accuracy considering to use it in real life applications.

2.3.2 Fluid Flows

A wide range of industrial processes and scientific phenomena involve gas or fluids flows over complex obstacles. In engineering applications the temporal evolution of non-ideal, compressible fluids is quite often modeled by Navier-Stokes [58, 59] equations. It is based on the fundamental laws of mass-, momentum- and energy conservation, extended by the dissipative effects of viscosity, diffusion and heat conduction. By neglecting all the above non-ideal processes, and assuming adiabatic variations, we obtain the Euler equations [60, 61], describing the dynamics of dissipation-free, inviscid, compressible fluids. The equations, a coupled set of nonlinear hyperbolic partial differential equations, in conservative form expressed as

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{v}) = 0 \tag{2.2}$$

$$\frac{\partial (\rho \mathbf{v})}{\partial t} + \nabla \cdot \left(\rho \mathbf{v}\mathbf{v} + \hat{I}p \right) = 0 \tag{2.3}$$

$$\frac{\partial E}{\partial t} + \nabla \cdot ((E + p)\mathbf{v}) = 0, \tag{2.4}$$

where t denotes time, ∇ is the Nabla operator, ρ is the density, u, v are the x- and y-components of velocity vector $\mathbf{v}$, respectively, p is the pressure of the fluid, $\hat{I}$ is the identity matrix, and E is the total energy density defined as

$$E = \frac{p}{\gamma - 1} + \frac{1}{2}\rho \mathbf{v} \cdot \mathbf{v}, \tag{2.5}$$

where γ is the ratio of specific heats. It is convenient to merge (2.2), (2.3) and (2.4) into hyperbolic conservation law form in terms of $U = [\rho, \rho u, \rho v, E]$ and the

DOI:10.15774/PPKE.ITK.2011.003

flux tensor

$$\mathbf{F} = \begin{pmatrix} \rho\mathbf{v} \\ \rho\mathbf{v}\mathbf{v} + Ip \\ (E+p)\mathbf{v} \end{pmatrix} \tag{2.6}$$

as

$$\frac{\partial U}{\partial t} + \nabla \cdot \mathbf{F} = 0. \tag{2.7}$$

2.3.2.1 Discretization of the governing equations

Since logically structured arrangement of data is fundamental for the efficient operation of the FPGA based implementations, we consider explicit finite volume discretizations [62] of the governing equations over structured grids employing a simple numerical flux function. Indeed, the corresponding rectangular arrangement of information and the choice of multi-level a temporal integration strategy ensure the continuous flow of data through the CNN-UM architecture. In the followings we recall the basic properties of the mesh geometry, and the details of the considered first- and second-order schemes.

2.3.2.2 The geometry of the mesh

The computational domain is composed of $n \times m$ logically structured quadrilaterals, called finite volumes (in the 2D case, the volumes are treated as planes) as shown in Figure 2.12.

Indices i and j refer to the volume situated in the i^{th} column and the j^{th} row. The corners of the volumes can be described practically by any functions X and Y of indices (i, j), provided that degenerated volumes do not appear:

$$x_{i,j} = X(i,j), y_{i,j} = Y(i,j), \tag{2.8}$$

where $x_{i,j}$ and $y_{i,j}$ stand for the $x-$ and the $y-$ component of corner point (i, j), respectively, $i \in [1, n+1]$ and $j \in [1, m+1]$. In real life applications X and Y follow from the functions describing the boundaries of the computational domain. Consider a general finite volume with indices (i, j), presented in Figure 2.12. Its volume is labeled by $V_{i,j}$, while $\mathbf{n}_f$ represents the outward pointing normal vector of face f scaled by the length of the face.

DOI:10.15774/PPKE.ITK.2011.003

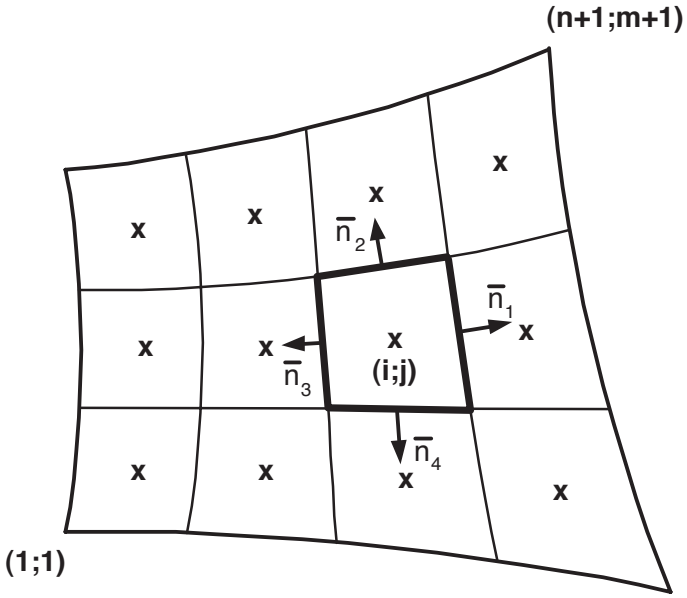


Figure 2.12: The computational domain

2.3.2.3 The First-order Scheme

The simplest algorithm we consider is first-order both in space and time [56] [57]. The application of the finite volume discretization method leads to the following semi-discrete form of governing equations (2.6)

$$\frac{dU_{i,j}}{dt} = -\frac{1}{V_{i,j}} \sum_f \mathbf{F}_f \cdot \mathbf{n}_f, \tag{2.9}$$

where the summation is meant for all the four faces of cell (i,j), $\mathbf{F}_f$ is the flux tensor evaluated at face f, and $\mathbf{n}_f$ is the outward pointing normal vector of face f scaled by the length of the face. Let us consider face f in a coordinate frame attached to the face, such that its x-axes is normal to f. Face f separates cell L (left) and cell R (right). In this case the $\mathbf{F}_f \cdot \mathbf{n}_f$ scalar product equals to the x-component of $\mathbf{F}(\mathrm{F}_x)$ multiplied by the length of the face (or area in 3D case). In order to stabilize the solution procedure, artificial dissipation has to be introduced into the scheme. Following the standard procedure, this is achieved by replacing the physical flux tensor by the numerical flux function F^N containing

DOI:10.15774/PPKE.ITK.2011.003

the dissipative stabilization term. A finite volume scheme is characterized by the evaluation of F^N, which is the function of both U_L and U_R. Applying the simple and robust Lax-Friedrichs numerical flux function defined as

$$F^N = \frac{F_L + F_R}{2} - (|\bar{u}| + \bar{c}) \frac{U_R - U_L}{2}. \tag{2.10}$$

In the last equation c is the local speed of sound, F_L=$\mathrm{F}_x(\mathrm{U}_L)$, F_R=$\mathrm{F}_x(\mathrm{U}_R)$ and bar labels speeds computed at the following averaged state

$$\bar{U} = \frac{U_L + U_R}{2}. \tag{2.11}$$

The last step concludes the spatial discretization. Finally, the temporal derivative is dicretized by the first-order forward Euler method:

$$\frac{dU_{i,j}}{dt} = \frac{U_{i,j}^{n+1} - U_{i,j}^{n}}{\Delta t}, \tag{2.12}$$

where $U_{i,j}^n$ is the known value of the state vector at time instant n, $U_{i,j}^{n+1}$ the unknown value of the state vector at time instant $n+1$, and Δt the time step. By working out the algebra described so far, leads to the following discrete form

DOI:10.15774/PPKE.ITK.2011.003

of the governing equations:

$$\begin{aligned}
\rho_C^{n+1} &= \rho_C^n - \\
&-\frac{\Delta t}{\Delta x}\left(\left(\frac{\rho u_C^n+\rho u_E^n}{2} - (|\bar{u}|+\bar{c})\,\frac{\rho_E^n-\rho_C^n}{2}\right)\right. \\
&-\left(\frac{\rho u_W^n+\rho u_C^n}{2} - (|\bar{u}|+\bar{c})\,\frac{\rho_C^n-\rho_W^n}{2}\right) \\
&+\left(\frac{\rho v_C^n+\rho v_N^n}{2} - (|\bar{u}|+\bar{c})\,\frac{\rho_N^n-\rho_C^n}{2}\right) \\
&\left.-\left(\frac{\rho v_S^n+\rho v_C^n}{2} - (|\bar{u}|+\bar{c})\,\frac{\rho_C^n-\rho_S^n}{2}\right)\right)
\end{aligned} \tag{2.13a}$$

$$\begin{aligned}
\rho u_C^{n+1} &= \rho u_C^n - \\
&-\frac{\Delta t}{\Delta x}\left(\left(\frac{\left(\rho u^2+p\right)_C^n+\left(\rho u^2+p\right)_E^n}{2} - (|\bar{u}|+\bar{c})\,\frac{\rho u_E^n-\rho u_C^n}{2}\right)\right. \\
&-\left(\frac{\left(\rho u^2+p\right)_W^n+\left(\rho u^2+p\right)_C^n}{2} - (|\bar{u}|+\bar{c})\,\frac{\rho u_C^n-\rho u_W^n}{2}\right) \\
&+\left(\frac{\rho uv_C^n+\rho uv_N^n}{2} - (|\bar{u}|+\bar{c})\,\frac{\rho u_N^n-\rho u_C^n}{2}\right) \\
&\left.-\left(\frac{\rho uv_S^n+\rho uv_C^n}{2} - (|\bar{u}|+\bar{c})\,\frac{\rho u_C^n-\rho u_S^n}{2}\right)\right)
\end{aligned} \tag{2.13b}$$

$$\begin{aligned}
\rho v_C^{n+1} &= \rho v_C^n - \\
&-\frac{\Delta t}{\Delta x}\left(\left(\frac{\rho uv_C^n+\rho uv_E^n}{2} - (|\bar{u}|+\bar{c})\,\frac{\rho v_E^n-\rho v_C^n}{2}\right)\right. \\
&-\left(\frac{\rho uv_W^n+\rho uv_C^n}{2} - (|\bar{u}|+\bar{c})\,\frac{\rho v_C^n-\rho v_W^n}{2}\right) \\
&+\left(\frac{\left(\rho v^2+p\right)_C^n+\left(\rho v^2+p\right)_N^n}{2} - (|\bar{u}|+\bar{c})\,\frac{\rho v_N^n-\rho v_C^n}{2}\right) \\
&\left.-\left(\frac{\left(\rho v^2+p\right)_S^n+\left(\rho v^2+p\right)_C^n}{2} - (|\bar{u}|+\bar{c})\,\frac{\rho v_C^n-\rho v_S^n}{2}\right)\right)
\end{aligned} \tag{2.13c}$$

$$\begin{aligned}
E_C^{n+1} &= E_C^n - \\
&-\frac{\Delta t}{\Delta x}\left(\left(\frac{(E+p)u_C^n+(E+p)u_E^n}{2} - (|\bar{u}|+\bar{c})\,\frac{E_E^n-E_C^n}{2}\right)\right. \\
&-\left(\frac{(E+p)u_W^n+(E+p)u_C^n}{2} - (|\bar{u}|+\bar{c})\,\frac{E_C^n-E_W^n}{2}\right) \\
&+\left(\frac{(E+p)v_C^n+(E+p)v_N^n}{2} - (|\bar{u}|+\bar{c})\,\frac{E_N^n-E_C^n}{2}\right) \\
&\left.-\left(\frac{(E+p)v_S^n+(E+p)v_C^n}{2} - (|\bar{u}|+\bar{c})\,\frac{E_C^n-E_S^n}{2}\right)\right)
\end{aligned} \tag{2.13d}$$

Complex terms in the equations were marked with only one super- and subscript for better understanding, for example $(\rho u^2+p)_C^n$ is equal to $\rho_C^n\left(u_C^n\right)^2+p_C^n$.

DOI:10.15774/PPKE.ITK.2011.003

Notations $|\bar{u}|$ and $|\bar{c}|$ represent the average value of the u velocity component and the speed of sound at an interface, respectively.

A vast amount of experience has shown that these equations provide a stable discretization of the governing equations if the time step obeys the following Courant-Friedrichs-Lewy condition (CFL condition):

$$\Delta t \leq \min_{(i,j)\in([1,M]\times[1,N])} \frac{\min(\Delta x, \Delta y)}{|u_{i,j}| + c_{i,j}}. \tag{2.14}$$

2.3.2.4 The Second-order Scheme

The overall accuracy of the scheme can be raised to second-order if the spatial and the temporal derivatives are calculated by a second-order approximation. One way to satisfy the latter requirement is to perform a piecewise linear extrapolation of the primitive variables P_L and P_R at the two sides of the interface in (2.10). This procedure requires the introduction of additional cells with respect to the interface, i.e. cell LL (left to cell L) and cell RR (right to cell R). With these labels the reconstructed primitive variables are

$$P_L = P_L + \frac{g_L(\delta P_L, \delta P_C)}{2}, P_R = P_R - \frac{g_R(\delta P_C, \delta P_R)}{2}, \tag{2.15}$$

with

$$\delta P_L = P_L - P_{LL}, \delta P_C = P_R - P_L, \delta P_R = P_{RR} - P_R \tag{2.16}$$

while g_L and g_R are the limiter functions.

The previous scheme yields acceptable second-order time-accurate approximation of the solution, only if the variations in the flow field are smooth. However, the integral form of the governing equations admits discontinuous solutions as well, and in an important class of applications the solution contains shocks. In order to capture these discontinuities without spurious oscillations, in (2.15) we apply the *minmod* limiter function, also:

$$g_L(\delta P_L, \delta P_C) = \begin{cases} \delta P_L & \text{if} \quad |\delta P_L| < |\delta P_C| \\ & \text{and} \quad \delta P_L \delta P_C > 0 \\ \delta P_C & \text{if} \quad |\delta P_C| < |\delta P_L| \\ & \text{and} \quad \delta P_L \delta P_C > 0 \\ 0 & \text{if} \quad \delta P_L \delta P_C \leq 0 \end{cases} \tag{2.17}$$

The function $g_R(\delta P_C, \delta P_R)$ can be defined analogously.

DOI:10.15774/PPKE.ITK.2011.003

2.3.2.5 Implementation on the Cell Architecture

By using the previously described discretization method, a C based CFD solver is developed which is optimized for the SPEs of the Cell architecture.

The simplest way to define a body fitted mesh is to define the coordinates of the vertices. By using these coordinates, the length and the normal vector of the faces and the volume of the quadrilateral can be computed. This solution, however, requires the lowest memory bandwidth. This is not efficient, due to the lack of hardwired floating point divider and square root unit inside the SPE. Computing these values on the SPE requires as many instructions as computing the derivative and updating the state value of one cell. Additionally, these values are constant during the computation and can be computed in advance by the PPU. Eight constant variables are required to describe the quadrilaterals, namely the length of the interfaces, the normal vector components, the volume and the mask. Even though, using these precomputed values, the memory I/O bandwidth requirement of the algorithm is increased by 45%, the number of required instructions is nearly halved, therefore, the computing performance is doubled.

Since the relatively small local memory of the SPEs does not allow to store all the required data, an efficient buffering method is necessary to save memory bandwidth. In our solution a belt of 5 rows is stored in the local memory from the array: 2 rows are required to form the local neighborhood of the currently processed row, one line is required for data synchronization, and two lines to allow the overlap of the computation and communication as shown in Figure 2.13. Additionally, 2 rows are needed to store the constant values and temporarily store the primitive variables (u, v, p) computed from the conservative variables $(\rho, \rho u, \rho v, E)$. During implementation the environment of the CNN simulation kernel was used. Template operations are optimized according to the discretized equations (2.7) to improve performance. The optimized kernel requires about 32KB memory from the local store of the SPE leaving approximately 224KB for the row buffers. Therefore, the length of the buffer is of maximum 1000 grid points, while the number of rows is only limited by the size of the main memory. Storing the precomputed constant values requires 27% more local store per grid point, but this overhead is affordable due to the doubled performance.

DOI:10.15774/PPKE.ITK.2011.003

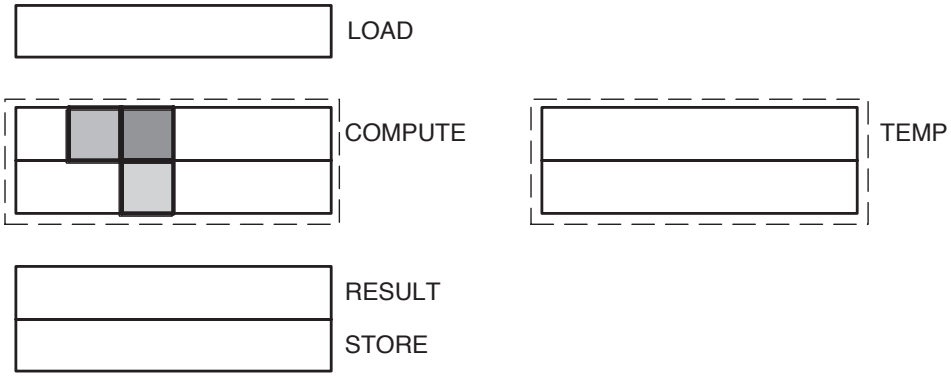


Figure 2.13: Local store buffers

To utilize the power of the Cell architecture computation work should be distributed between the SPEs. In spite of the large memory bandwidth of the architecture, the memory bus can be easily saturated. Therefore, an appropriate arrangement of data between SPEs can greatly improve computing performance. One possible solution is to distribute grid data between the SPEs. In this case each SPE work on a narrow horizontal slice of the grid, similarly to the first row of SPEs in Figure 2.14.

Though the above data arrangement is well suited for the architecture of array processors and simplifies the inter-processor communication, the eight SPEs access the main memory in parallel, which might require very high memory bandwidth. If few instructions are executed on large data sets, the memory system is saturated resulting in low performance. One possible solution for this problem is to form a pipeline using the SPEs to compute several iterations in parallel as shown in Figure 2.14. In this case continuous data flow and synchronization are required between the neighboring SPEs but this communication pattern is well suited for the ring structure of the EIB.

The static timing analysis of the optimized CFD solver kernel showed that a grid point can be updated in approximately 250 clock cycles. Each update requires moving 64byte data (4x4byte conservative state value, 4x4byte updated state value, 8x4byte constant value) between the main memory and the local store of the SPE. The Cell processor runs on 3.2GHz clock frequency, therefore, in an ideal case the expected performance of the computation kernel using one

DOI:10.15774/PPKE.ITK.2011.003

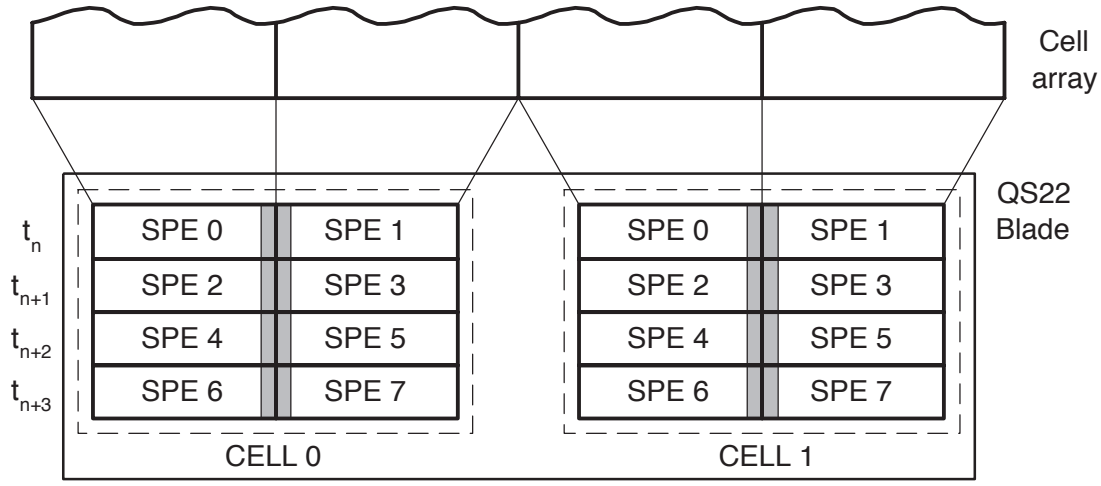


Figure 2.14: Data distribution between SPEs

SPE is 50.5 million update/s. The estimated memory bandwidth requirement is 3.23GByte/s, which is slightly more than the $1/8^{th}$ of the available memory bandwidth. Therefore, SPEs should be arranged in a 4×2 logical array where the 4 columns work on the 4 slices of the cell array and each row computes different iteration.

2.3.2.6 Implementation on Falcon CNN-UM Architecture

The Falcon architecture [30] is an emulated digital implementation of CNN-UM array processor which uses the full signal range model (Section 1.1). On this architecture the flexibility of simulators and computational power of analog architectures are mixed. Not only the size of templates and the computational precision can be configured but space-variant and non-linear templates can also be used.

The Euler equations are solved by a modified Falcon processor array in which the arithmetic unit was redesigned according to the discretized governing equations. Since each CNN cell has only one real output value, four layers are required to represent the variables ρ, ρu, ρv and E in case of Lax-Friedrichs approximation. In the first-order case the non-linear CNN templates acting on the ρu layer can easily be taken from (2.13b). Equations (2.18)-(2.20) show templates, in which

DOI:10.15774/PPKE.ITK.2011.003

cells of different layers are connected to the cell of layer ρu at position (i, j).

$$A_1^{\rho u} = \frac{1}{2\Delta x} \begin{bmatrix} 0 & 0 & 0 \\ \rho u^2 + p & 0 & -(\rho u^2 + p) \\ 0 & 0 & 0 \end{bmatrix} \tag{2.18}$$

$$A_2^{\rho u} = \frac{1}{2\Delta x} \begin{bmatrix} 0 & -\rho uv & 0 \\ 0 & 0 & 0 \\ 0 & \rho uv & 0 \end{bmatrix} \tag{2.19}$$

$$A_3^{\rho u} = \frac{1}{2\Delta x} \begin{bmatrix} 0 & \rho v & 0 \\ \rho u & -2\rho u - 2\rho v & \rho u \\ 0 & \rho v & 0 \end{bmatrix} \tag{2.20}$$

The template values for ρ, ρv and E layers can be defined analogously.

The $\rho uu + p$, ρuv, ρu and ρv terms can be reused during the computation of the neighboring cells and they should be computed only once in each iteration step. This solution requires additional memory elements but greatly reduces the area requirement of the arithmetic unit.

Other trick can be applied if we choose Δt to be integer power of two because the multiplication in this case can be done by shifts so we can eliminate several multipliers from the hardware and additionally the area requirements will be greatly reduced.

In the second-order case limiter function should be used on the primitive variables and the conservative variables are computed from these results. The limited values will be different for the four interfaces and cannot be reused in the computation of the neighboring cells. Therefore, this approach does not make it possible to derive CNN templates for the solution. However a specialized arithmetic unit still can be designed to solve it directly.

2.3.2.7 Results and performance

The previous results solving the Euler equations on a rectangular grid shows that only few specialized arithmetic unit can be implemented even on the largest FPGAs. In the body fitted case additional area is required to take into account the geometry of the mesh during the computation. Symmetrical nature of the problem, which can be seen on the templates (2.18)-(2.20), enable further optimization of the arithmetic unit which compensates the area increase due to

DOI:10.15774/PPKE.ITK.2011.003

coordinate transformations. The number of slices, multipliers and block-RAMs of the arithmetic unit can be seen in Figure (2.15)-(2.17) respectively.

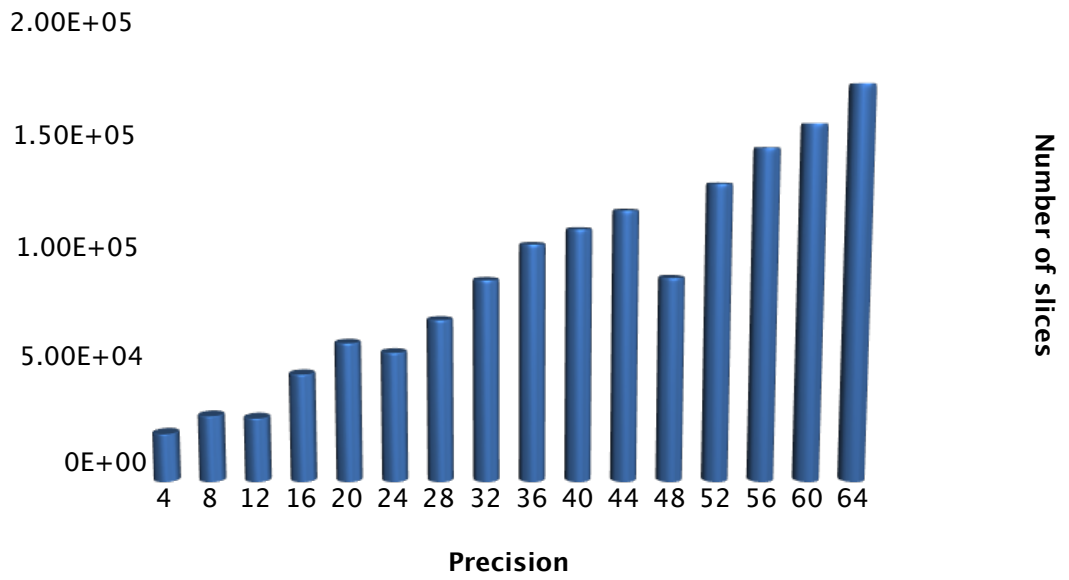


Figure 2.15: Number of slices in the arithmetic unit

To show the efficiency of our solution a complex test case was used, in which a Mach 3 flow around a cylinder was computed. The direction of the flow is from left to right and the speed of the flow at the left boundary is 3-times the speed of sound constantly. The solution contains a symmetrical bow shock flow around the cylinder. Therefore, only the upper half of the region should be simulated. This problem was solved on a 128×256 grid, which was bent over the cylinder using 2ms timestep. Result of the computation after 1s of simulation time is shown in Figure 2.18.

The experimental results of the average computation time are compared to a Intel Core2Duo microprocessor is shown on Table 2.4.

The Cell based solution is 35 times faster compared to that of a high performance microprocessor, even using a single SPE during the computation. Utilizing all the 16 SPEs of the IBM CellBlade the computation can be carried out two orders of magnitude faster, while the FPGA solution is three order of magnitude

DOI:10.15774/PPKE.ITK.2011.003

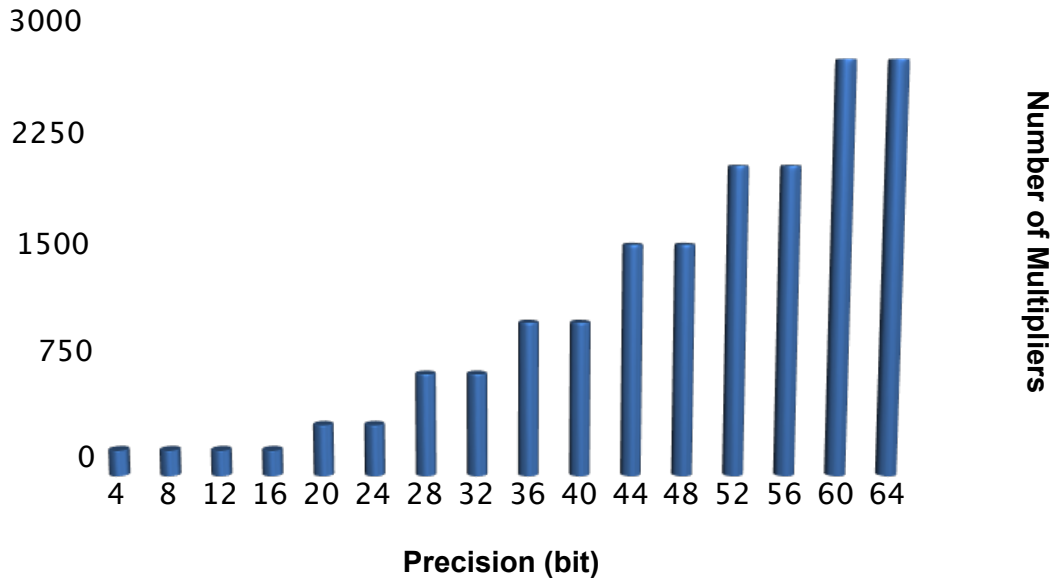


Figure 2.16: Number of multipliers in the arithmetic unit

faster. The power dissipation and the area of the architectures are in the same range.

2.3.3 Conclusion

Complex spatio-temproral dynamical problems are analyzed by a topographic array processor. A CNN simulation kernel was implemented on the Cell architecture and was optimized according to the special requirements of the IBM Cell processor. By using this kernel both linear and nonlinear CNN arrays can be simulated. Based on the basic CNN simulation kernel a framework was developed to compare the optimal mapping of the simulation of a complex spatio-temporal dynamics on Xilinx Virtex FPGA and on IBM Cell architecture. The framework has been tested by the acceleration of a computational fluid dynamics (CFD) simulation. During the implementation the goal was to reach the highest possible computational performance.

The governing equations of two dimensional compressible Newtonian flows on body fitted mesh geometry were solved by using different kind of Xilinx Virtex 5 FPGAs and by using the IBM QS22 and LS22 systems. The Falcon proces-

DOI:10.15774/PPKE.ITK.2011.003

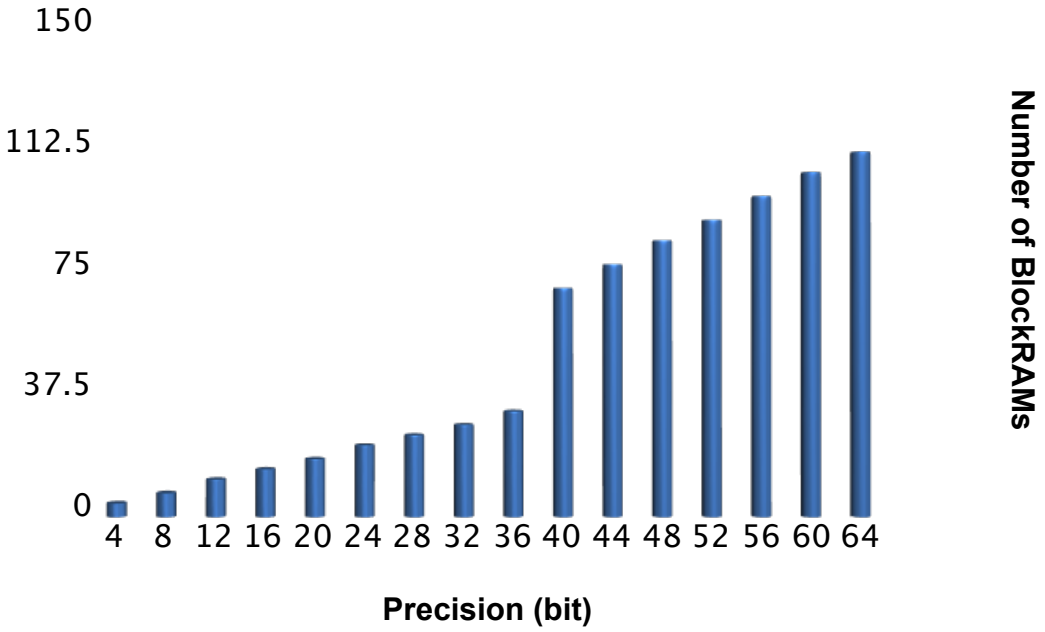


Figure 2.17: Number of block-RAMs in the arithmetic unit

sor was redesigned according to the discretized version of the partial differential equations optimized for the dedicated elements (BlockRAM, multiplier) of the FPGA. A process is developed for the optimal bandwidth management between the processing elements and the memory on Xilinx Virtex and on IBM Cell architectures. It turned out, that placing a memory element close to the processor results in a beneficial effect on the computing speed, which provides a minimum one order of magnitude higher speedup independently from the dimension of the problem.

One order of magnitude speedup can be achieved between an inhomogenous architecture, like the IBM Cell, and a custom architecture optimized for Xilinx Virtex FPGA using the same area, dissipated power and precision. During the simulation of CFD on body fitted mesh geometry the Xilinx Virtex 5 SX240T running on 410 MHz is 8 times faster, than the IBM Cell architecture with 8 synergistic processing element running on 3.2 GHz. Their dissipated power and area are in the same range, 85 Watt, 253mm^2 and 30 Watt, 400 mm^2 respectively. Considering the IBM Cell processor's computing power per watt performance as

DOI:10.15774/PPKE.ITK.2011.003

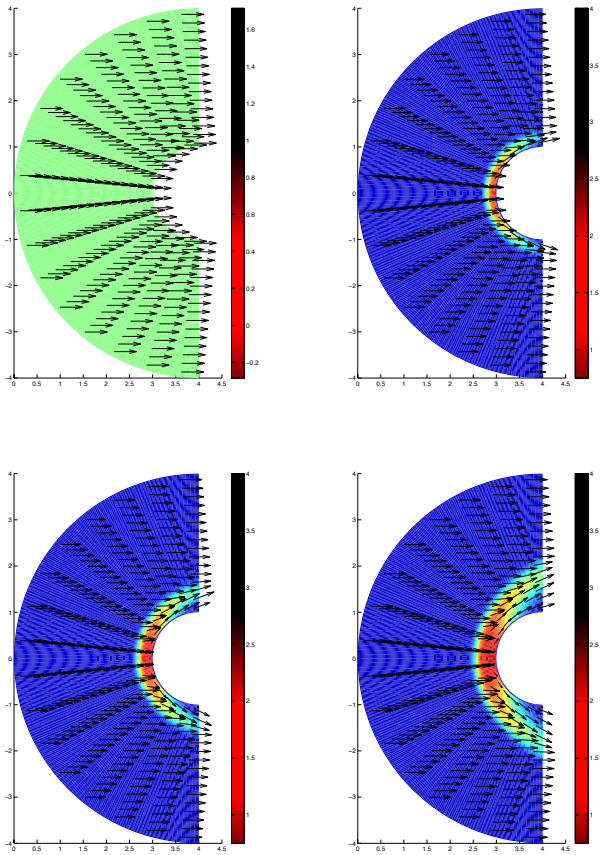


Figure 2.18: Simulation around a cylinder in the initial state, 0.25 second, 0.5 second and in 1 second

DOI:10.15774/PPKE.ITK.2011.003

Table 2.4: Comparison of different hardware implementations

	Implementations			
	Intel	Cell Processor		FPGA
	Core2Duo	1 SPE	16 SPEs	SX240T
Clock Frequency (MHz)	2000	3200	3200	410
Million cell iteration/s	1.05	37.3206	529.9525	2500
Computation Time on 128×512 1 step (ms)	62.41524	1.756028	0.123664	0.01311
Computation Time on 128×512 65536 steps (s)	4294.967	120.8372	8.50966	0.86
Speedup	1	35.54343	504.7167	1922.448
Power Dissipation (W)	65	85	2×85	~ 30
Area (mm^2)	143	-	2×253	389

a unit, computational efficiency of the Xilinx Virtex 5 SX240T FPGA is 22 times higher, while providing 8 times higher performance. The one order of magnitude speedup of the FPGA is owing to the arithmetic units working fully parallel and the number of implementable arithmetic units. During CFD simulation, the IBM Cell processor and the FPGA based accelerator can achieve 2 and 3 order of magnitude speedup respectively compared to a conventional microprocessor (e.g.: Intel x86 processors).

DOI:10.15774/PPKE.ITK.2011.003

Chapter 3

Investigating the Precision of PDE Solver Architectures on FPGAs

Reconfigurable devices seems to be the most versatile devices to implement array processors. Flexibility of the FPGA devices enable to use different computing precisions during the solution of PDEs and evaluate different architectures quickly [30]. Higher computing precision requires wider mantissa which results in larger implementation area. For that reason it is important to determine the minimal required computational precision of the algorithm. It is a simple common practice to use fixed wordlength during the implementation of the datapath on FPGA. The required silicon area are consumed by a given implementation can be estimated by the area of the processing units, which is determined mainly by the number of operations.

In this chapter the optimal precisions with fixed-point and floating-point arithmetic units on a simple test case is investigated. A single uniform wordlength was determined in order to have the required accurate solution. There are more elegant ways to find a more area efficient solutions, like the multiple wordlength selection problem, but it is proven to be a NP-hard problem [63].

The main motivation of the investigation of the precision was the finite number of resources. Because different type of problems requires different computational precision it is necessary to know which problem can be mapped into the FPGA. In the next section a simple PDE is investigated in order to find a method to

DOI:10.15774/PPKE.ITK.2011.003

determine the optimal computational precision of the advection equation solver architecture.

3.1 The Advection Equation

For the test the simple advection equation (3.1) is selected, where the analytical solution is known and can be easily generated.

$$\frac{\partial u}{\partial t}+\mathrm{c}\frac{\partial u}{\partial x}=0 \tag{3.1}$$

where t denotes time, u is conserved property, c is the advection speed. The initial condition is a proper periodical function with the property of:

$$u(x,t_0)=u_0(x) \tag{3.2}$$

the range of the function is from 0 to 1. Periodic boundary condition are used on the boundaries. With this initial conditions the architecture can easily be tested, because of the periodicity, the result should be in the same range as the initial condition. The analytical solution is

$$\tilde{u}(x,t)=u_0(x-c(t-t_0)) \tag{3.3}$$

The numerical approximation of the advection equation is not easy, especially if the initial condition u_0 is discontinuous function. Its structure and solution method is similar to those equations which used in Fluid Flow simulation [55]. This is a good starting equation which helps us investigating the precision of the arithmetic unit on FPGA to reach the predefined accuracy of the solution. However this chapter deals only with the one-dimensional case, and this 1D can not be used in real life applications, it gives us experience for the two- and three dimensional cases.

3.2 Numerical Solutions of the PDEs

In order to solve the continuous space-time PDE the equations has to be discretized. Since logically structured arrangement of data is fundamental for the

DOI:10.15774/PPKE.ITK.2011.003

efficient operation of the FPGA based implementations, we consider explicit finite volume discretizations of the governing equations over structured grids employing a simple numerical flux function. In the following we recall the details of the considered first- and second-order schemes.

3.2.1 The First-order Discretization

The simplest algorithm we consider is first-order both in space and time. Application of the finite volume discretization method leads to the following semi-discrete form of governing equations

$$\frac{dU_i}{dt} = -\frac{1}{V_{i,j}} \sum_f \mathbf{F}_f, \tag{3.4}$$

where the summation is meant for the two faces of cell i, $\mathbf{F}_f$ is the flux tensor evaluated at face f. Face f separates cell L (left) and cell R (right).

In order to stabilize the solution procedure, artificial dissipation has to be introduced into the scheme. Following the standard procedure, this is achieved by replacing the physical flux tensor by the numerical flux function F^N containing a dissipative stabilization term. The finite volume scheme is characterized by the evaluation of F^N, which is the function of both U_L and U_R. In the paper we apply the simple and robust Lax-Friedrichs numerical flux function defined as

$$F^N = \frac{F_L + F_R}{2} - \bar{c} \cdot \frac{U_R - U_L}{2} \tag{3.5}$$

and bar labels speeds computed at the following averaged state

$$\bar{U} = \frac{U_L + U_R}{2}. \tag{3.6}$$

In equation (3.5) c is the velocity component, F_L=$\mathrm{F}_x(\mathrm{U}_L)$, F_R=$\mathrm{F}_x(\mathrm{U}_R)$.

A vast amount of experience has shown that these equations provide a stable discretization of the governing equations if the time step satisfies the Courant–Friedrichs–Lewy condition (CFL condition):

$$\frac{u \cdot \Delta t}{\Delta x} \leq c \tag{3.7}$$

where u is the velocity, Δt is the time-step, Δx is the length interval.

DOI:10.15774/PPKE.ITK.2011.003

3.2.2 The Second-order Limited Scheme

The overall accuracy of the scheme can be raised to second-order if the spatial and the temporal derivatives are calculated by a second-order approximation. One way to satisfy the latter requirement is to perform a piecewise linear extrapolation of the primitive variables P_L and P_R at the two sides of the interface in (3.5). This procedure requires the introduction of additional cells with respect to the interface, i.e. cell LL (left to cell L) and cell RR (right to cell R). With these labels the reconstructed primitive variables are

$$P_L = P_L + \frac{g_L\left(\delta P_L, \delta P_C\right)}{2}, P_R = P_R - \frac{g_R\left(\delta P_C, \delta P_R\right)}{2}, \tag{3.8}$$

with

$$\delta P_L = P_L - P_{LL}, \delta P_C = P_R - P_L, \delta P_R = P_{RR} - P_R \tag{3.9}$$

while g_L and g_R are the limiter functions.

The previous scheme yields acceptable second-order time-accurate approximation of the solution, only if the variations in the flow field are smooth. In case of discontinuous initial conditions this simple second order approximation can not be used, because the method can be unstable. In order to capture these discontinuities without spurious oscillations, in (3.8) we apply the *minmod* limiter function, also:

$$g_L\left(\delta P_L, \delta P_C\right) = \begin{cases} \delta P_L & \text{if} \quad |\delta P_L| < |\delta P_C| \\ & \text{and} \quad \delta P_L \delta P_C > 0 \\ \delta P_C & \text{if} \quad |\delta P_C| < |\delta P_L| \\ & \text{and} \quad \delta P_L \delta P_C > 0 \\ 0 & \text{if} \quad \delta P_L \delta P_C \leq 0 \end{cases} \tag{3.10}$$

The function $g_R\left(\delta P_C, \delta P_R\right)$ can be defined analogously.

3.3 Testing Methodology

The goal of the experiment was to assign an adequate step size and an optimal precision for the problem, which is described by equation (3.1). During the solution different fixed-point and floating point numbers are used. The methodology of the experiment is the investigation of different precision on different grid resolution.

DOI:10.15774/PPKE.ITK.2011.003

In fixed-point case, the precision means the length of the whole number, where 2 bits represents the integer part and the rest is for the fraction, because the numbers are between 0 and 1. In the floating point the exponent length is constant 11 bit wide and the rest is the mantissa, which will be varied during the examination. Current computing architectures are supporting 32 bit floating point or 64 bit double precision numbers for computations. Fixed-point computations can be carried out by using 32 or 64 bit integers. The precision of the fixed- and floating point numbers on FPGA can be defined in much smaller steps. For the computer simulation of the algorithm we used the Mentor Graphics Algorithmic C Datatypes [64], which is a class-based C++ library that provides arbitrary-length fixed-point data types. For different types of floating-point numbers the GNU MPFR library [65] can be used. Despite of the fact, the MPFR contains low-level routines, the first floating-point computations, which we planned to make, took a very long time. Therefore an efficient arithmetic unit were created on FPGA to overcome the slow speed of the arbitrary precision floating-point computations, where the floating point operators from Xilinx CoreGenerator Library [37] are used.

The initial condition is a sinus wave and periodic boundary condition is used. The initial conditions was tested because of the simplicity of the sinus function with different precisions, where the result function should match the initial condition after one period. The resolution of the grid is from 100 to 100000 point, the speed of advection is 12/13 and ΔT is 16/17 times the optimal ΔT, which can be computed by the CFL condition. These parameters are selected to meet some critical property, namely it shall not be represented in finite length binary number, on the other hand the number of time steps to compute one period shall be an integer to overcome phase errors. The result was simply compared to the initial condition. The error of the solution is measured by the L_∞ (infinity) norm defined by the following equation:

$$|x|_\infty = \max_i |x_i| \tag{3.11}$$

The method to find a minimal bit width of the arithmetic unit is heuristic. Investigation of different precision operating units are done empirically. With the increase of the bit width a more accurate result can be achieved as can be seen

DOI:10.15774/PPKE.ITK.2011.003

in Figure (3.1) in the floating point first order discretization case. The slope of the error curves in different precision in fixed point first order and second order discretization case are similar to the floating point first order case in different grid resolution. This increasing should be done until the required accuracy of the solution is reached. There are two cases, when the selected arithmetic precision is not satisfactory:

- With the increase of the grid resolution may results, that the truncation and roundoff error overcomes the method error before the required accuracy is reached. In this case a higher precision arithmetic should be chosen, and the iterative grid refining should be restarted.
- If the required accuracy of the solution is reached before a grid resolution (truncation and roundoff error domination), then a lower precision arithmetic unit can be selected in order to use a smaller architecture.

To find the proper arithmetic precision for the required accuracy, the minimum point of the curve should be determined with the pre defined grid resolution (e.g.: the 38 bit width with 10^4 grid resolution in Figure (3.1)) where the larger arithmetic precision does not result in higher accuracy. With this method the minimal grid resolution can be determined for an expected accuracy with the investigation of the roundoff and truncation error in different arithmetic precision.

After the first couple of floating-point simulations using moderate grid size it turned out, that the complete testing of even the lowest 29 bit precision case will take more than 200 hours. That is why we decided to make an arithmetic unit on FPGA.

3.4 Properties of the Arithmetic Units on FPGA

The purpose was not to gain the highest speed with an acceleration board, but to investigate the precision of a computation, there for a familiar development board is used. The architecture was developed on a Xilinx XC2V6000 FPGA [37], which takes place on an Alpha Data [66] board. It contains 33,792 slices, 144 18×18 bit multipliers and 144 18Kb BlockRAMs. It is not the latest FPGA, but it represents well todays low cost, low power FPGA.

DOI:10.15774/PPKE.ITK.2011.003

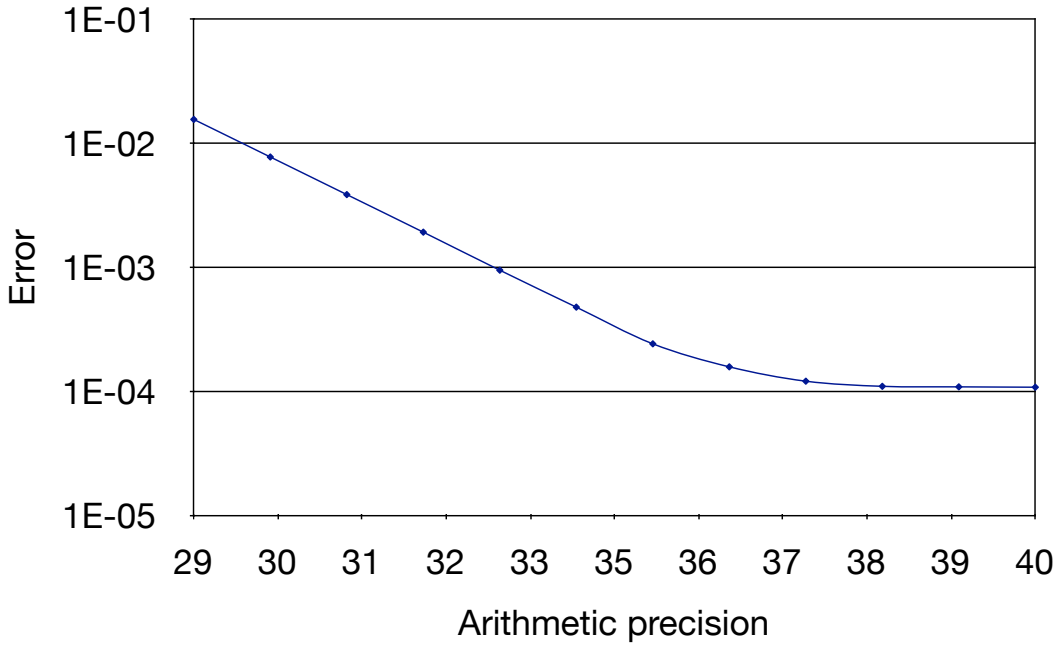


Figure 3.1: Error of the 1st order scheme in different precision with 10^4 grid resolution

The arithmetic unit of the first order scheme is shown in Figure (3.2). The architecture of the second order solution is build from two first order arithmetic units and extended by some floating point units according to equations (3.10) (see Figure 3.3).

The whole architecture can be seen in Figure 3.4, where the advection equation solver unit is the accelerator. Area requirements of the arithmetic unit is shown in Figure (3.5). The required number of slices for the arithmetic unit is increased by 25-30% as the precision is increasing from 29 bit to 40 bit in both the 1st and 2nd order cases. The low slope of the area increase derives from the size of the dedicated elements of the FPGA, whereas the multiplication of two 29 bit numbers requires the same amount of 18×18 dedicated multiplier block as in the two 35 bit number multiplication case. The main source of the area increase is the higher area requirements of the more accurate addition/subtraction operators.

Computation of the first and second order method is well suited to a conventional microprocessor (e.g.: 3GHz Intel Core 2 Duo [67]) when the native (32, 64) bit width is used. Even the data set of the finest resolution grid can be stored

DOI:10.15774/PPKE.ITK.2011.003

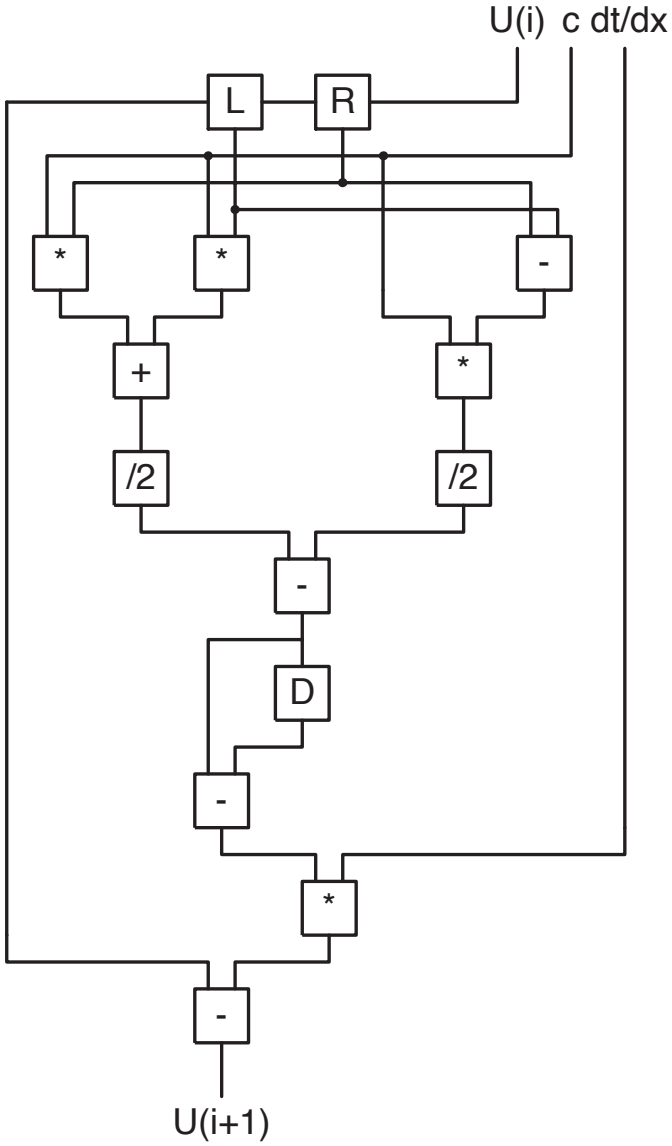


Figure 3.2: The arithmetic unit of the first order scheme

DOI:10.15774/PPKE.ITK.2011.003

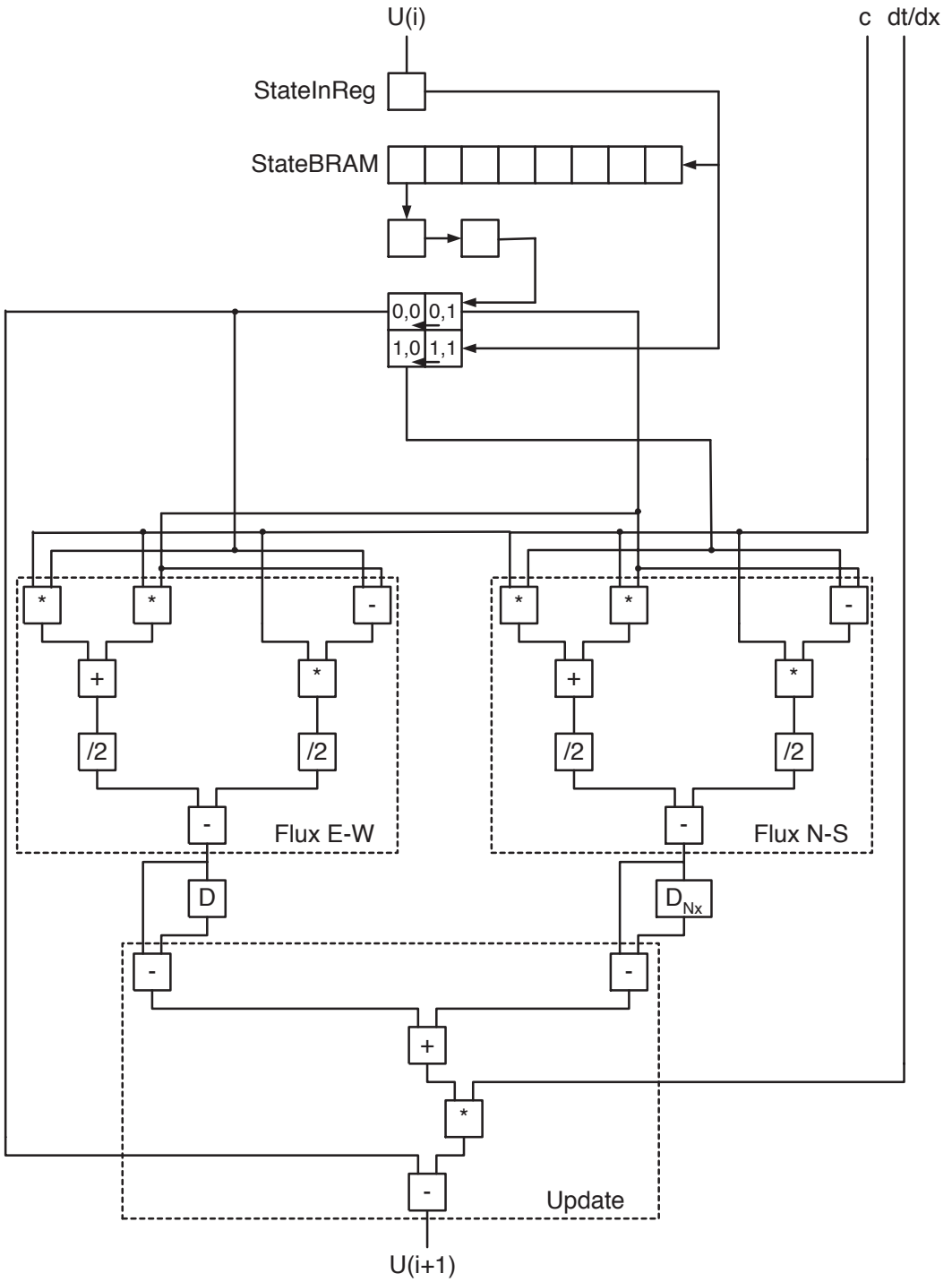


Figure 3.3: The arithmetic unit of the second order scheme

DOI:10.15774/PPKE.ITK.2011.003

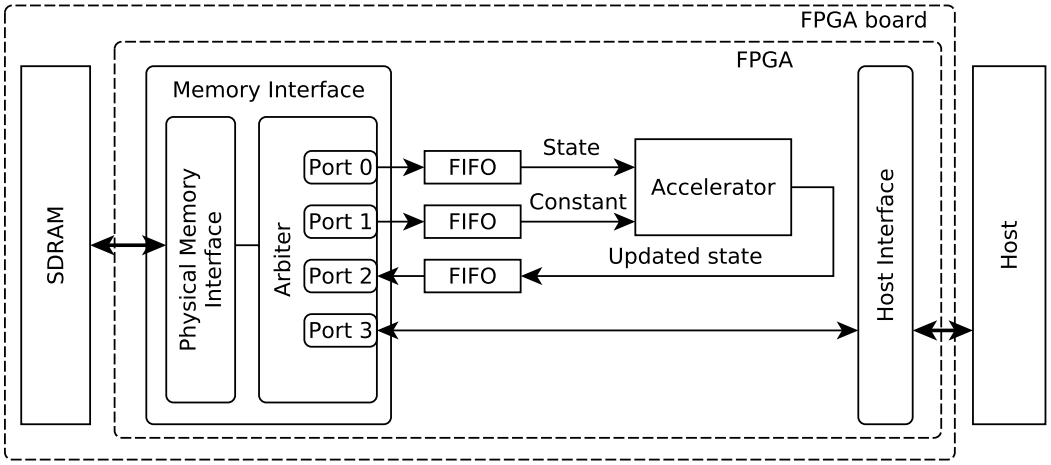


Figure 3.4: Structure of the system with the accelerator unit

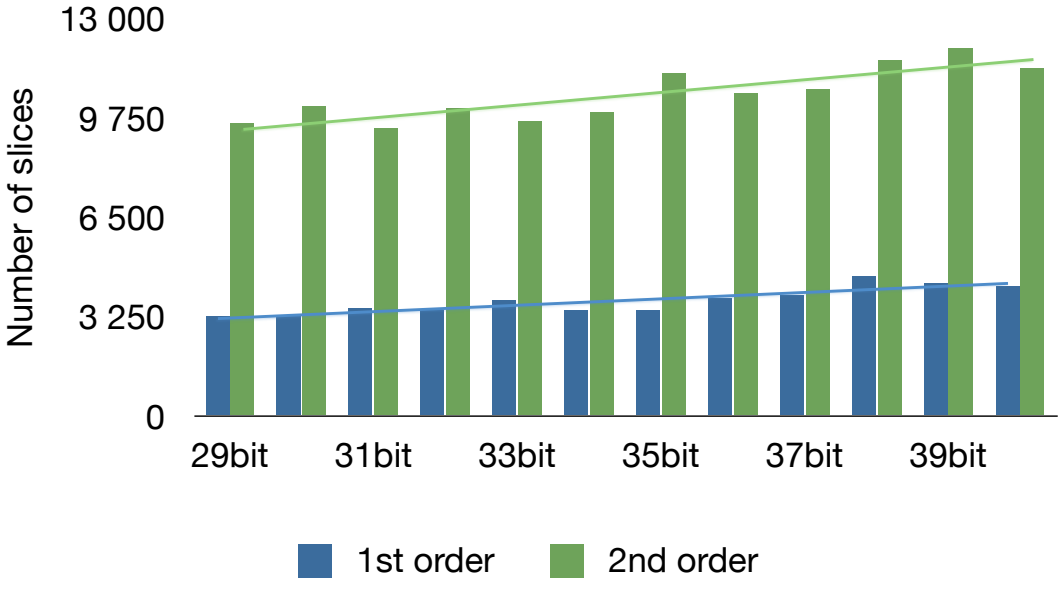


Figure 3.5: Number of slices of the Accelerator Unit in different precisions

DOI:10.15774/PPKE.ITK.2011.003

in the L2 cache of the processor comfortably. However the architecture implemented on the FPGA was not optimized for speed and runs only on 100MHz, its performance is comparable or in some cases slightly higher than a 3GHz Intel Core 2 Duo microprocessor.

3.5 Results

The L_∞ norm of the error in case of different grid resolution and different fixed-point and floating-point precisions are compared in Figure (3.6) – (3.8). As the resolution is increased, the accuracy of the solution can be improved. Finer grid resolution requires smaller time-step, according to the CFL condition, which results in more operations. After a certain number of steps the round-off error of the computation will be comparable or larger than the truncation error of the numerical method. Therefore the step size can not be decreased to zero in a given precision. The precision of the arithmetic units should be increased when the grid is refined.

The arithmetic unit in the 2nd order case (see Figure 3.7) is twice as large as in 1st order arithmetic unit, additionally the step-size should be halved therefore four times more computation is required. The slope of the error is higher as shown in Figure 3.9, therefore computation on a coarser grid results in higher accuracy.

The L_∞ norm of fix-point and floating point solutions with the same mantissa width are compared in Figure 3.9. Because we use 11 bit for the exponent in floating-point numbers the 40 bit floating point number compared to the 29 bit fix-point number. As it can be seen, the floating-point computation results in more accurate solution than the fix-point. But just adding four bits to the 29bit fixed-point number and using 33bit fixed-point number the L_∞ norm of the error is comparable to the error of the 40bit floating point computations. In addition 7 bits are saved, which results in lower memory bandwidth and has a reduced area requirement. The fixed-point arithmetic unit requires 22 multipliers while the floating-point arithmetic unit requires only 16 multipliers, the number of required slices is far more less in the fixed-point case. While the 34 bit fixed-point

DOI:10.15774/PPKE.ITK.2011.003

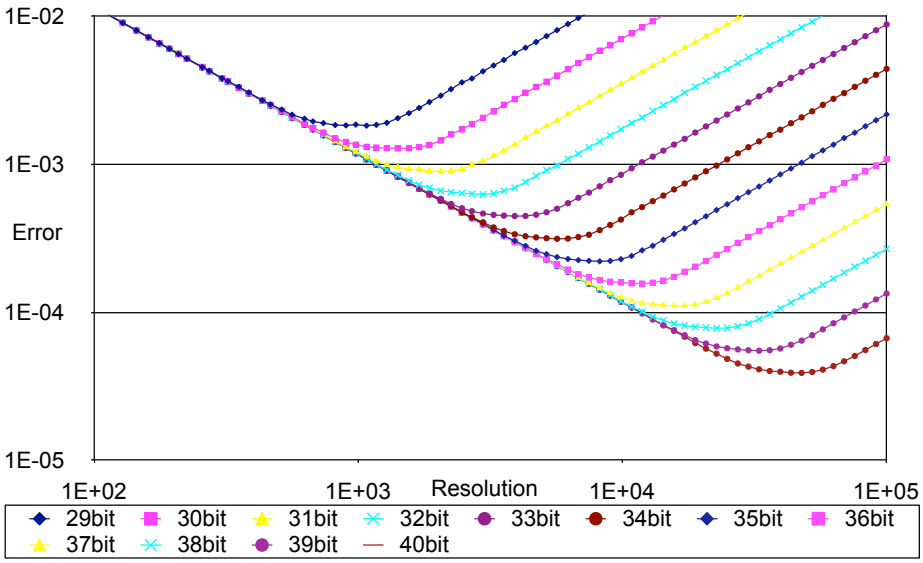


Figure 3.6: Error of the 1st order scheme in different precisions and step sizes using floating point numbers

DOI:10.15774/PPKE.ITK.2011.003

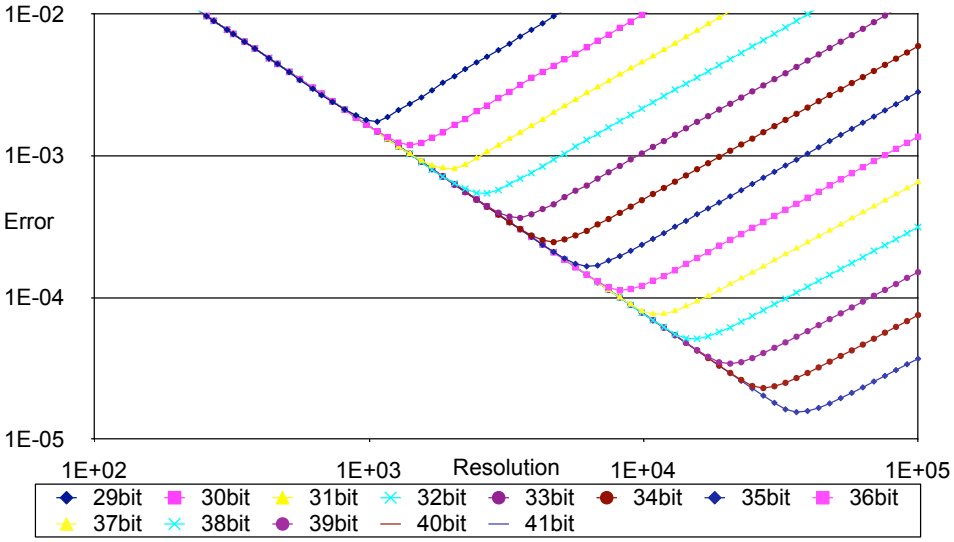


Figure 3.7: Error of the 2nd order scheme in different precisions and step sizes using floating point numbers

DOI:10.15774/PPKE.ITK.2011.003

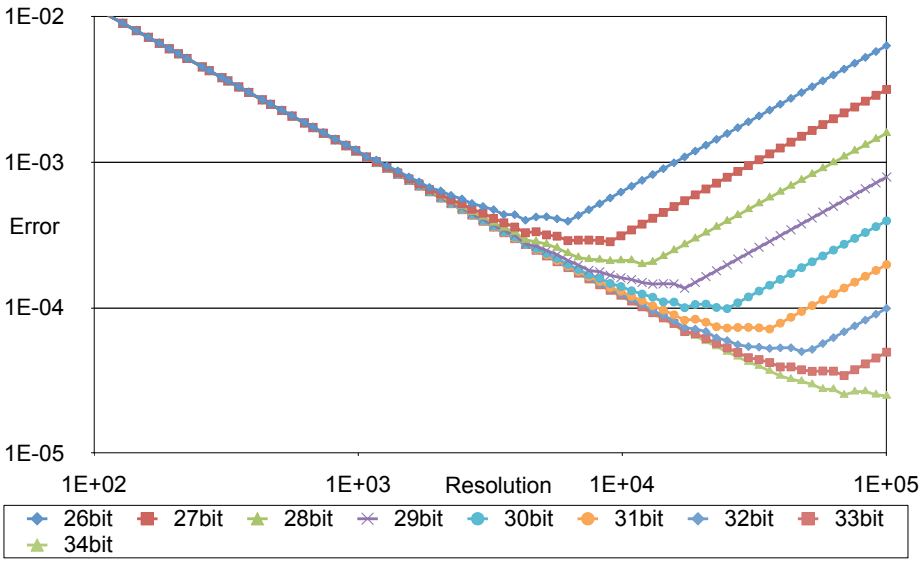


Figure 3.8: Error of the 1st order scheme in different precisions and step sizes using fixed-point numbers

DOI:10.15774/PPKE.ITK.2011.003

arithmetic unit requires only 774 slices from the FPGA, the 40 bit floating-point (which has only 29 bit mantissa) uses 11718 slices.

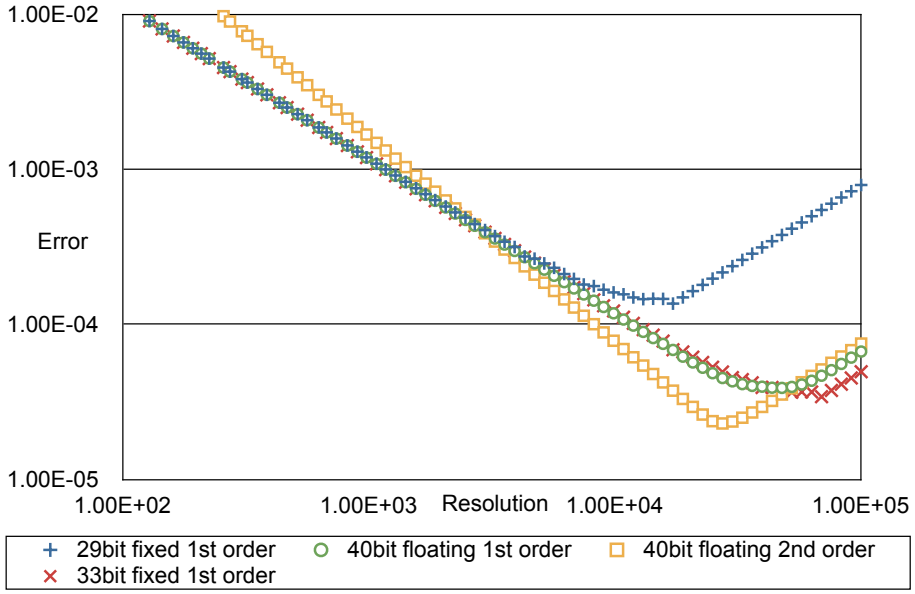


Figure 3.9: Comparison of the different type of numbers and the different discretization

The goal is to find the finest grid resolution for a given precision where the roundoff and truncation errors are in the same range. This can be easily determined by examining Figure 3.9, where the optimal grid size is located at the minimum point of the error function.

The second goal is to find the optimal precision for a given grid resolution. Starting from a low precision simulation and increasing the precision by one bit, the error is halved in each step when the roundoff error is larger than the truncation error. The precision should be increased until the error term can not be decreasing any more. In this point the error of the solution will be dominated by the truncation error and the effect of roundoff error will be negligible.

DOI:10.15774/PPKE.ITK.2011.003

3.6 Conclusion

During engineering computations usually 64 bit floating point numbers are used to reach an appropriate accuracy. However solution of several complex computational problems using 64 bit numbers consumes huge computing power. It is worth to examine the required precision, if the computing resources, power dissipation or size are limited or the computation should be carried out in real time.

Significant speedup can be achieved by decreasing the state precision. Engineering applications usually does not require 14-15 digit accuracy, therefore the decreased computational precision can be acceptable. Reduction of the state precision makes it possible to map some particularly complex problems onto an FPGA. A methodology was developed to specify the minimal required computational precision to reach the maximal computing performance on FPGA where the accuracy of the solution and the grid resolution is given a-priori. The required computational precision can only be determined precisely in infrequent cases, when the exact solution is known.

A method was elaborated to find the minimum required computing precision of the arithmetic units when the step size, spatial resolution and the required accuracy is defined. A tested method was given to find the precision of the arithmetic unit of a problem, which has analytic solution. For problems without analytic solution, the reduced precision results can be compared to the 64 bit floating point reference precision. The finest resolution of the grid can also be determined by the method if the desired accuracy is defined.

During the investigation of the arithmetic unit of the advection equation solver the precision is decreased from 40 bit to 29 bit, while area requirements of the architecture are decreased by 20-25% independently from the applied discretization method. Clock frequency of the arithmetic units does not increase significantly due to the decreased precision, the main source of speedup is the increased number of implementable arithmetic units on the FPGA.

The area requirements of the arithmetic units can be significantly reduced by using properly normalized fixed point numbers. During the investigation of the advection equation solver architecture, error of the solution of the 33 bit fixed

DOI:10.15774/PPKE.ITK.2011.003

point and the 40 bit floating point (29 bit mantissa) arithmetic unit is in the same order, but the area required for the arithmetic unit is decreased by 15 times. The main source of speedup is the increased number of implementable arithmetic units on the FPGA, when fixed point arithmetic is used.

DOI:10.15774/PPKE.ITK.2011.003

DOI:10.15774/PPKE.ITK.2011.003

Chapter 4

Implementing a Global Analogic Programming Unit for Emulated Digital CNN Processors on FPGA

4.1 Introduction

Cellular Neural/Nonlinear Networks (CNN) are defined as locally connected, analog, stored programmable processor arrays [68], visual microprocessors. The topographic, sensory, Cellular Wave Computer architectures, based on the CNN-UM (Universal Machine) principle, have been implemented in various physical forms [69] such as mixed-mode CMOS VLSI, emulated-digital (both on ASIC and FPGA), DSP, and optical implementations.

The analog VLSI implementation of the extended CNN-UM architectures ([22], [70], [23], [25], [71], and [24]) exhibit very high speed (few TeraOP/s) with low power dissipation, but these architectures have some disadvantages: they are known to have relative low accuracy (about 7-8 bit) and moderate flexibility, moreover the number of cells is limited (e.g., 128×128 on ACE16k, or 176×144 on eye-RIS). On the one hand their cost is high, and the development time is long, due to the utilization of full-custom VLSI technology. On the other hand, the highly flexible software solutions (running on host processors or DSPs) are usually insufficient, considering the performance of computations.

DOI:10.15774/PPKE.ITK.2011.003

To overcome the former problems of the analog solutions, the emulated-digital CNN-UM implementations, as aggregated arrays of processing elements, provide non-linear template operations and multi-layer structures of high accuracy with somewhat lower performance [31], [26], and [45].

Among the CNN-UM implementations the emulated-digital approaches based on FPGAs (Field Programmable Gate Arrays) have been proved to be very efficient in the numerical solution of various complex spatio-temporal problems described by PDEs (Partial Differential Equations) [46]. Several FPGA implementations of the specialized emulated-digital CNN architecture, for example a real-time, multi-layer retina model [18], non-linear template runner [72], 2D seismic wave propagation models [19], and solution of 2D Navier-Stokes equations, such as barotropic ocean model [73], and compressible flow simulation [74] have been successfully designed and tested.

Hence, the elaborated Falcon architecture on FPGA [45] seems to be a good trade-off between the high-speed analog VLSI CNN-UM and the versatile software solutions. In order to provide high flexibility in CNN computations, it is interesting how we can reach large performance by connecting locally a lot of simple and relatively low-speed parallel processing elements, which are organized in a regular array. The large variety of configurable parameters of this architecture (such as state- and template-precision, size of templates, number of rows and columns of processing elements, number of layers, size of pictures, etc.) allows us to arrange an implementation, which is best suited to the target application (e.g. image/video processing or fluid flow simulation). So far, without the proper controller (Global Analog Programming Unit, GAPU) extension, when solving different types of PDEs, a single set of CNN template operations has been implemented on the host PC: by downloading the image onto the FPGA board (across a quite slow parallel port), computing the transient, and finally uploading the result back to the host computer where logical, arithmetic and program organizing steps were executed.

If, howewer, we want to run analogic CNN algorithms with template statements, the Falcon architecture has to be extended with this proposed GAPU implementation. It simplifies the downloading of the input picture(s)/image(s) (e.g.: pictures for image processing, or surfaces for tactile sensing, etc.) along

DOI:10.15774/PPKE.ITK.2011.003

with the sequence of analogic instructions (the program) onto the FPGA, through merely asserting a 'start' signal. All the above may be done either without a host PC, thus our architecture is capable of achieving stand-alone operation, which is desirable in many industrial contexts. The complex analogic algorithms require classical program organization elements, i.e., sequential-, iterative- and conditional execution of instructions. Consequently, the embedded GAPU must be supplied to the Falcon architecture to extend it to a fully functional CNN-UM implementation on a reconfigurable FPGA.

4.2 Computational background and the optimized Falcon architecture

The Falcon architecture implemented on reconfigurable FPGA iterates the forward-Euler discretized CNN equation by using FSR (Full-Signal Range) model, which is derived from the original Chua-Yang model [68]. The equations for the Euler method are as follows:

$$x_{m,ij}(n+1) = x_{m,ij}(n) + \sum_{n=1}^{p} \sum_{kl \in S_r(ij)} A'_{mn,ij,kl} \cdot x_{n,kl}(n) + g_{m,ij} \tag{4.1}$$

$$g_{m,ij} = \sum_{n=1}^{p} \sum_{kl \in S_r(ij)} B'_{mn,ij,kl} \cdot u_{n,kl}(n) + h \cdot z_{m,ij}, \tag{4.2}$$

where the number of layers is denoted by p, the state of the cell is equal to its output and limited in the [-1, +1] range. It contains processing elements in a square grid, and the time-step value h is inserted into the A' and B' template matrices. Moreover, supposing that input is constant or changing slowly, g_{ij} can be treated as constant and should be calculated only once at the beginning of the computation.

This multi-layer extension of the elaborated Falcon architecture based on FPGA [45] can be used for emulating a fully connected multi-layer CNN structure: both the number of layers and the computing accuracy are configurable. The main blocks of the Falcon architecture are extended with a low-level Control unit, which is optimized for GAPU extension, and they are shown in Figure 4.1.

DOI:10.15774/PPKE.ITK.2011.003

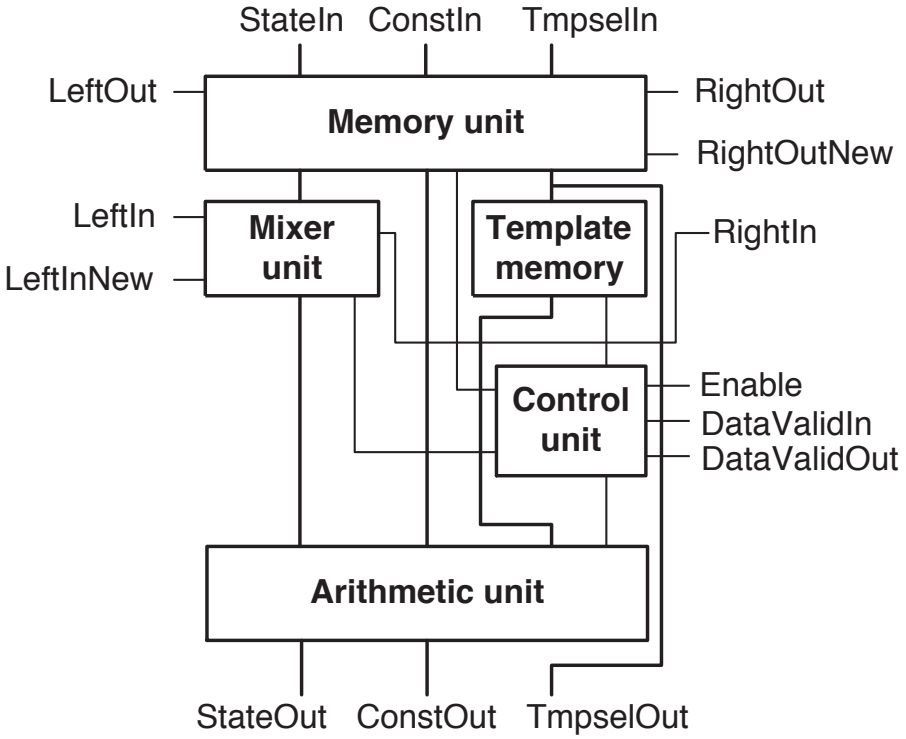


Figure 4.1: The structure of the general Falcon processor element is optimized for GAPU integration. Main building blocks and signals with an additional low-level Control unit are depicted.

The integrated Control logic has three different control signals: the Enable allows the operation of the Falcon processor element (FPE) for one clock cycle, the DataValidIn denotes that valid input data arrived, while DataValidOut indicates the first computed result.

Depending on the template size, several rows from the given layer should be stored in the Memory unit. It must be supplied with input data from the main memory (RAM) of the proposed CNN-UM implementation. The arithmetic unit is modified and optimized for GAPU. Which can be built up from $(2r + 1) \times (2r + 1)$ multipliers instead of r dedicated elements. This means that it is capable of computing the new state value within one clock cycle, supposing nearest neighborhood (r=1) templates and a single-layer structure. The hardware complexity of the application increases quadratically as the number of layers are

DOI:10.15774/PPKE.ITK.2011.003

increased, so for multi-layer structures the arithmetic core must perform r^2 times more computations than in the single-layer case, and the templates may be treated as $r \times r$ pieces of single-layer templates.

The input picture is partitioned among the physical FPEs: each column of the processors works on vertical stripe of the image, while each row executes one iteration step in the discretized time. The inter-processor communication is solved on processor level. However, certain timing and control signals must be supplied to the first processor row in order to indicate the processing elements when the first input data are available. On the other hand, the input pipeline gets the value of the last pixel in a row, as pixel values are stored continuously in the RAM. These signals ensure the duplication of the first and last rows of pixels satisfying the Neumann-type (zero-flux) boundary condition.

One possible problem might be that prototyping boards usually have different memory-bus width than the Falcon array has. For this reason an Input and Output Interface are necessary to be implemented between the on-board RAM memory and the Falcon array (in Figure 4.2).

The Falcon interface multiplexes and transforms multiple words of data (width of the data bus) into a w wide data word, according to the following equation:

$$w = n \times (w_s + w_c + w_{ts}) \,, \tag{4.3}$$

where n means columns of processors, each one works with w_s bits of state, w_c bits of constant, and w_{ts} bits of template width. This might be a bottle-neck in speed, but it can be eliminated by the careful selection of the number of processor elements. An additional Arbiter unit provides the communication with necessary bus control and gating logic.

4.3 Implementation

4.3.1 Objectives

Since the Falcon processor is only capable of computing the result of one iteration step on the picture stored in the on-board RAM, it should be extended with an additional high-level control unit, called GAPU, when more iteration steps are

DOI:10.15774/PPKE.ITK.2011.003

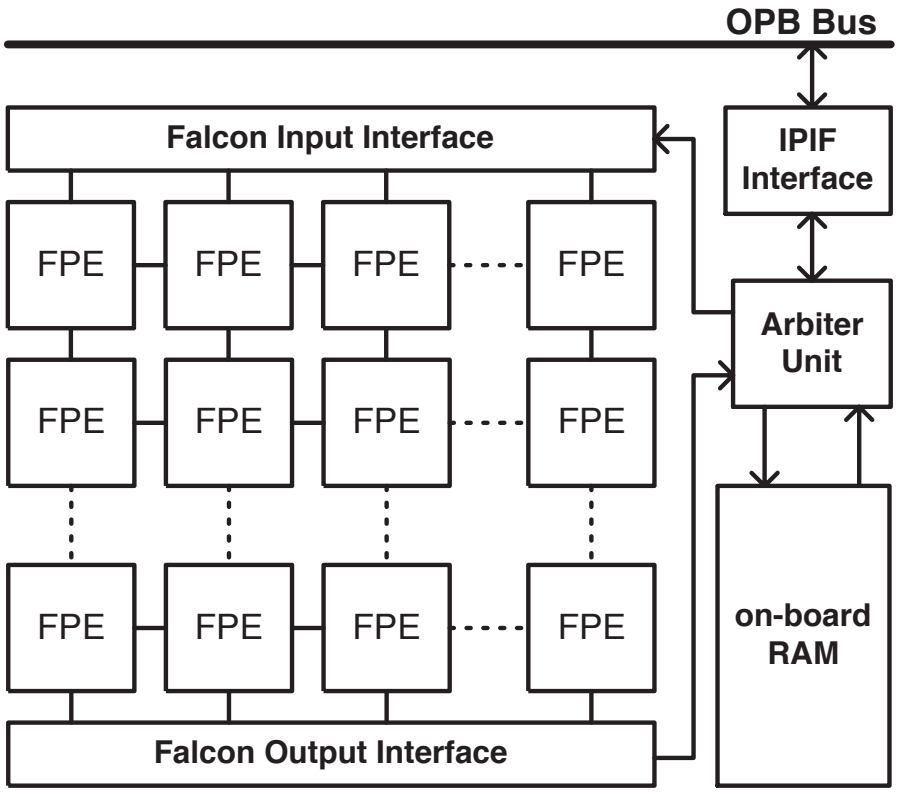


Figure 4.2: Structure of the Falcon array based on locally connected processor elements.

DOI:10.15774/PPKE.ITK.2011.003

required. Analogical operations are considered - a dual computational structure, in which algorithms are implemented with analog and logical (arithmetical) operations. Consequently, the GAPU must control not only program organizing constructions and I/O instructions, but basic local logical (e.g., bitwise AND, OR, EQ etc.) and elementary arithmetic operations (e.g., addition, subtraction, multiplication) between two pictures. On the other hand, the GAPU should run analog instructions, as well. Timing and control signals (including template values, i.e., the 'analog' program) must be supplied to the Falcon architecture to be feasible for a low-cost programmable CNN-UM. In order to accelerate the arithmetic and logical operations a Vector processor element (VPE) should be implemented. It receives the same data structure as an FPE. The number of VPEs depends on the number of columns of the Falcon processors. The main objective during the GAPU implementation was, that the GAPU must incur a minimal overhead in area and no overhead in time at all.

To achieve these goals, the Xilinx MicroBlaze [37] architecture has been used for implementing most of the complex sophisticated CNN algorithms. MicroBlaze is an embedded soft processor core optimized for Xilinx FPGAs, meaning that it is implemented using general logic primitives rather than hard, dedicated blocks (as in case of the IBM PowerPC architecture, or the ARM processor architecture). It is designed to be flexible, providing control of a number of features: the 32-bit RISC architecture of Harvard-style with separated in instruction- and data buses running at full speed to execute programs and access data from both on-chip and external memory.

The MicroBlaze system design relies on a soft CPU core, a number of slave peripherals place on the CPU's On-chip Peripheral Bus (OPB) and RAM- or DMA-controllers with the addition of an Arbiter Unit for multi-processor and multi-bus support. This can be strongly application-specific, which we add and configure within the soft processor core. Consequently, during the implementation process we eliminated several unnecessary components from the MicroBlaze core, and the blocks needed to perform the analogical operations were kept.

DOI:10.15774/PPKE.ITK.2011.003

4.3.2 Implementation of GAPU

The architecture of the GAPU is built up from five main components as shown in Figure 4.3:

- MicroBlaze core,
- OPB bus (On-Chip Peripheral Bus),
- IPIF interface (Intellectual Property Interface),
- BRAM memories,
- Controller unit.

Additionally, it can be integrated with not only one, but several Falcon and Vector processor elements in an array, which significantly increases the computing performance. The number of the implementable processor is limited by the resources of the development board.

The State-, Const- and Template-BRAM memories store the state, constant and template values, respectively, while four special registers in the Controller module (called Command, Status, IterCounter and IterLeft) implement common functions in the instruction set processing. The Command register stores the actual instruction, the Status register shows the status of the process, IterCounter register stores the maximal number of iterations and the IterLeft register stores the number of remaining iterations. These control signals controls the work of the processors which are connected to the GAPU.

In our experimental system a Xilinx MicroBlaze core is attached to a Controller unit across an IPIF interface (Intellectual Property InterFace). According to the desired function, the Controller module generates control signals for operating both Falcon and Vector processors. These processor elements can obtain data from StateBRAM, ConstBRAM and TemplateBRAM memories implemented on the dedicated on-chip BRAM memories, but storing larger pictures the external on-board ZBT-SRAM modules might be used. The previously elaborated Falcon processor is capable of performing a number of analog operations, while arithmetical and logical instructions on the CNN array can be carried out by

DOI:10.15774/PPKE.ITK.2011.003

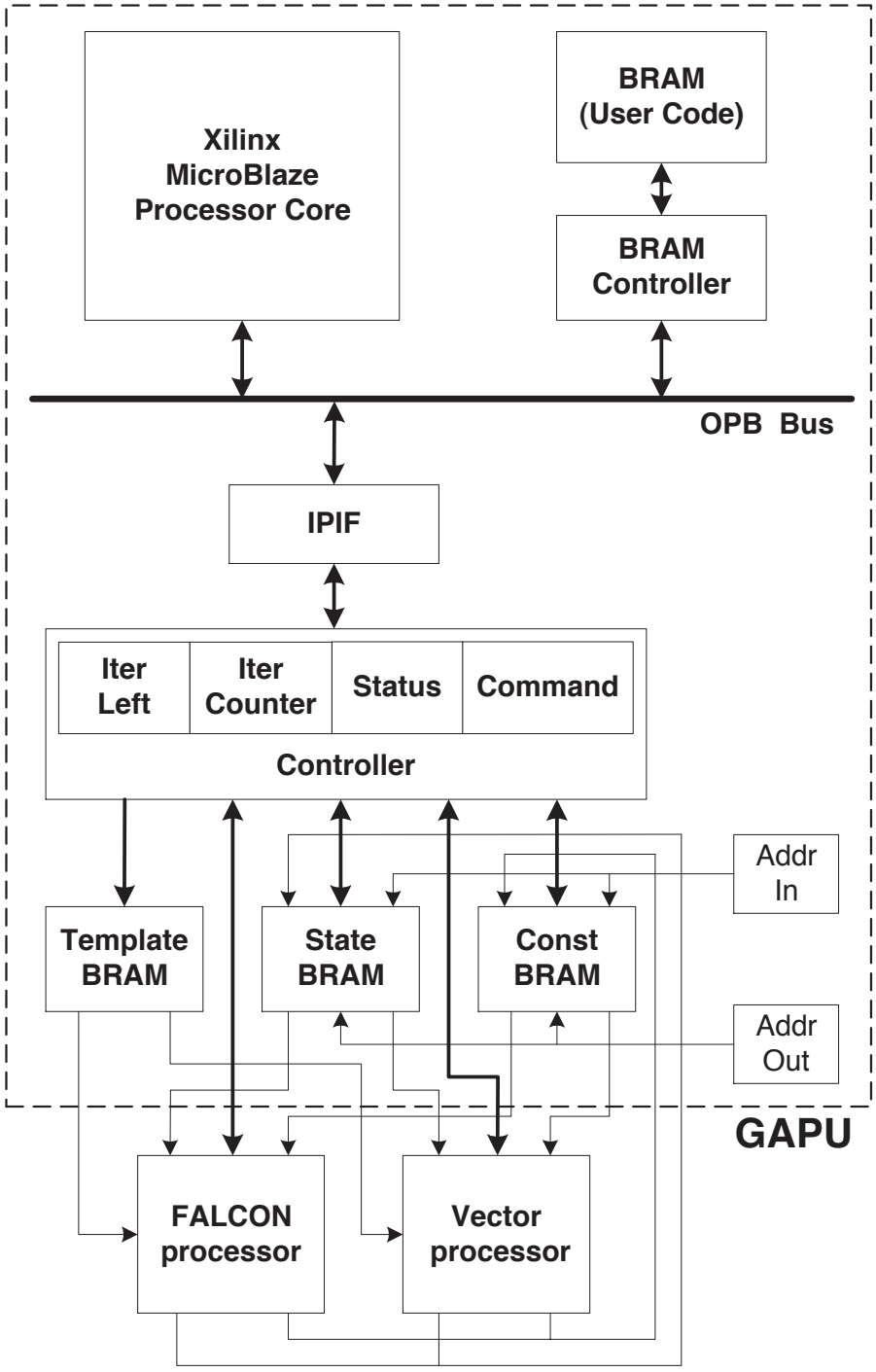


Figure 4.3: Detailed structure of the implemented experimental system (all blocks of GAPU are located within the dashed line).

DOI:10.15774/PPKE.ITK.2011.003

additional Vector processor units. In this structure - somewhat similar to mixed-signal processors - the duality (mentioned in [68], in the context of analog and logical computing) is expressed in the way timing and control signals are generated for all components by Controller block. The MicroBlaze core is connected across an IPIF interface to the OPB bus, which makes it possible to supply more Falcon processors and Vector processor elements in an array without any significant modifications, and perform operations simultaneously.

The entire computation can be started by a single writing operation of a register element. According to the given analog or logical/arithmetical operation, the Command register stores the instructions for Falcon and Vector processors. The Status register shows the current state of the CNN-array. The IterLeft denotes the actual number of the iteration, while the IterCount denotes the maximal number of iterations in the picture, which can be adjusted at the beginning of the computation. The AddrIn and AddrOut counters have to store the start addresses of the ConstBRAM and StateBRAM memories. Each bit-width (e.g., state-, constant-, template-widths) of the entire structure are configurable, thus can be adjusted to the corresponding bit-widths of the Falcon architecture for a specific application. Moreover, owing to the modular structure of the proposed GAPU if the development of a new analogical operation is required, the Command register can be easily extended with a given instruction set of the new function.

In order to utilize the high computing power of the modern FPGAs several modification need to perform on the GAPU architecture. The dedicated arithmetic units of the new generation FPGAs becomes faster, while the embedded microprocessors and the used bus systems evolves slower. The implemented Falcon PEs can work much faster than the MicroBlaze processor and its buses. I developed a new architecture in order to work the FPE, the embedded microprocessor, the controller circuit and the memory in different clock speed. This involves the modification of the StateBRAM memories. In spite of using a dual-ported BRAM, we need to use several single-ported ones. A few to serve the FPEs with datas and one for the MicroBlaze. In this case the relative slow MicroBlaze can monitor the state memories too, while the fast FPE can work continuously. With an additional FIFO element the StateBRAM memory can be reached from

DOI:10.15774/PPKE.ITK.2011.003

outside the GAPU architecture. The new architecture makes it possible for the MicroBlaze, for the control units and for the FPEs to reach the state memories from the outer memory at the same time.

4.3.3 Operating Steps

In practice the GAPU works as follows:

1. If we want to compute with an analog operation on an arbitrary sized picture, first a given template should be loaded into the Template BRAM memory. The maximal number of iterations can be set in the IterCount register, and in the Command register the analog operation should be given. The selected template is stored in Template BRAM.

2. The template is loaded from the Template BRAM into the Template memory unit of the Falcon processor.

3. In the next step the AddrIn and AddrOut pixel counters should be initialized along with the iteration counter storing the current step of iteration (IterLeft). The IterLeft gets the value of the IterCount register.

4. When a start (compute) signal is asserted, the GAPU sets the flag of the Command register and Status register to '1'. Depending on the content of the Command register, it will enable the Falcon processor to work and obtain data from ConstBRAM and StateBRAM memories. During processing the analog operation the Status register shows the operating status of the Falcon processor continuously.

5. If the signal AddrIn_En (a valid data is on the input of the processor) is 'true', the the AddrIn counter is incrementing with the processing of the datas. This counter helps us to monitor the processed datas. If the counter reaches the edge of the picture (the last pixel), its content is deleted.

6. If the Falcon processor has the first results, it sets the DataValidOut signal to 'true'. The way to write the memory is similar to reading the memory. The AddrOut register counts the number of computed pixels, and its content is reseted after the last computed pixel.

DOI:10.15774/PPKE.ITK.2011.003

7. If the computation reaches the last row of the picture, the number of iterations should be decreased by 1, and the AddrIn and AddrOut registers should be deleted.

The above steps will be iteratively performed, until the given number of iteration step is obtained. At this point the analog operation will be accomplished.

In the other case if we want to perform a logical or an arithmetic operation on an arbitrary sized binary picture, in the Command register a flag of the given instruction should be set, enabling the selected operation of the Vector processor element. The function of registers is similar to the above analog operation, but in case of logical processing, it is not necessary to use the iteration counter registers, because only one iteration should be performed.

These low-level operations may be hidden by a higher level software API, which makes it possible to control the communication between the MicroBlaze core and the Falcon or Vector processor cores. In order to provide suitable user interface, in addition, these functions simplify the download of input pictures and template values onto the processor elements, starting the CNN computation with given templates together with an arbitrary number of iterations, and finally uploading, furthermore, displaying the computed results.

4.4 The real image processing system

In the real image processing system, based on the RC203 FPGA development board [75], several interfaces are implemented to handle the memory, parallel port, camera input and video output parts, as shown in Figure 4.4.

Storing the partial results of calculation, the processors can access the on-board memory through the implemented ZBT interface. A Memory Arbitration unit decides on the component that can be used by the memory. The FIFO elements need to match the bit-width differences between the ZBT memory and other functional elements. The Parallel Port interface is applied to attach the system to the host PC for configuring (downloading / uploading) the processor. The VGA interface is necessary to connect the monitor which can display the computed results, while the video input of the system supports on-line video

DOI:10.15774/PPKE.ITK.2011.003

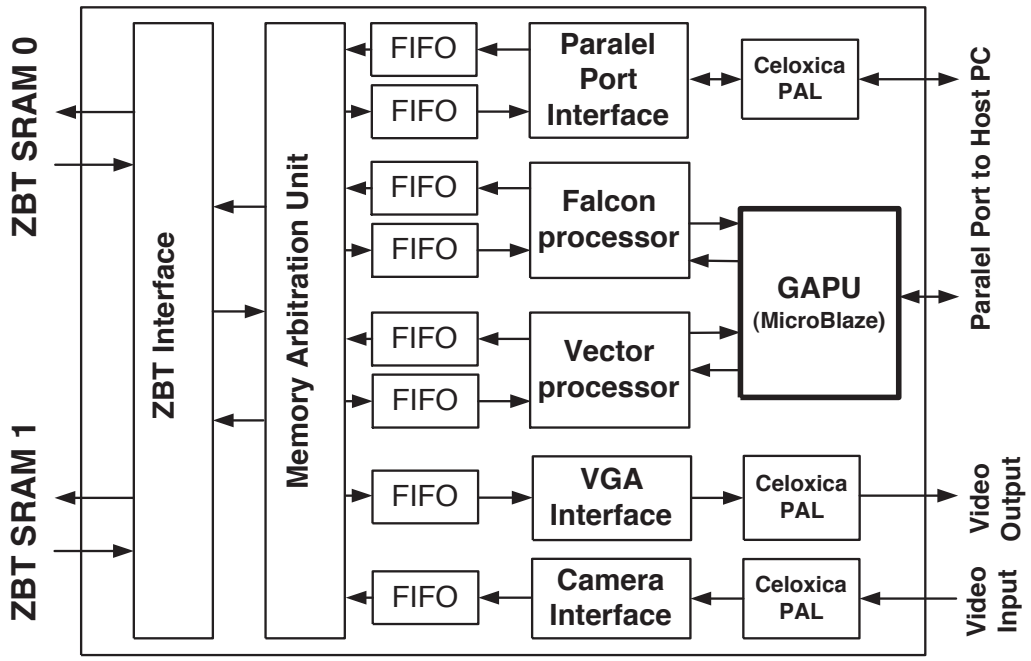


Figure 4.4: Block diagram of the experimental system. The embedded GAPU is connected with Falcon and Vector processing elements on FPGA.

DOI:10.15774/PPKE.ITK.2011.003

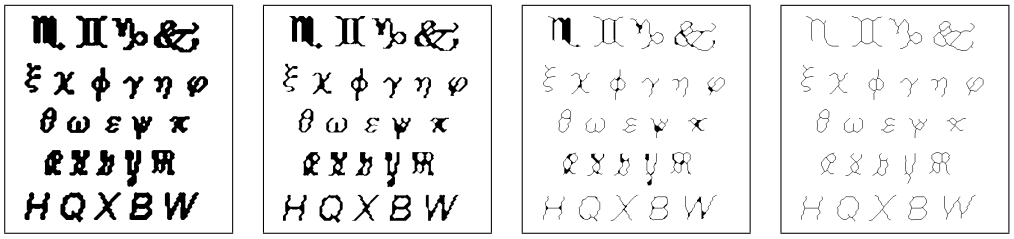
signal processing coming from PAL or NTSC video cameras, as well. Several parts of these predefined interfaces are provided from the Platform Abstraction Layer (PAL) API of Celoxica [32].

4.5 An Example

The example shows the functionality of GAPU, by using a skeletonization algorithm where a lot of template replacements should be performed in each iteration step. This may increase significantly the communication (download/upload) time between the host PC and FPGA, and this will be the great bottleneck when calculating the full processing time. Consequently, if we apply the embedded GAPU, it will reduce the communication time, and it provides a more efficient utilization of the Falcon processor.

The analogic algorithm in this example finds the skeleton of a black-and-white object. The 8 different templates should be applied circularly, always feeding the output result back to the input before using the next template. The templates of the algorithm are as follows (using the up-to-date Cellular Wave Computing Library [76]):

DOI:10.15774/PPKE.ITK.2011.003



Input 1^{st} iteration 5^{th} iteration 10^{th} iteration

Figure 4.5: Results after some given iteration steps of the black and white skeletonization.

SKELBW1:

$$B_1 = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix} \quad z_1 = -1$$

SKELBW2:

$$B_2 = \begin{bmatrix} 2 & 2 & 2 \\ 0 & 9 & 0 \\ -1 & -2 & -1 \end{bmatrix} \quad z_2 = -2$$

SKELBW3:

$$B_3 = \begin{bmatrix} 0 & 1 & 1 \\ -1 & 5 & 1 \\ 0 & -1 & 0 \end{bmatrix} \quad z_3 = -1$$

SKELBW4:

$$B_4 = \begin{bmatrix} -1 & 0 & 2 \\ -2 & 9 & 2 \\ -1 & 0 & 2 \end{bmatrix} \quad z_4 = -2$$

SKELBW5:

$$B_5 = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & 1 \\ 0 & 1 & 1 \end{bmatrix} \quad z_5 = -1$$

SKELBW6:

$$B_6 = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 9 & 0 \\ 2 & 2 & 2 \end{bmatrix} \quad z_6 = -2$$

SKELBW7:

$$B_7 = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 5 & -1 \\ 1 & 1 & 0 \end{bmatrix} \quad z_7 = -1$$

SKELBW8:

$$B_8 = \begin{bmatrix} 2 & 0 & -1 \\ 2 & 9 & -2 \\ 2 & 0 & -1 \end{bmatrix} \quad z_8 = -2$$

FEEDBACK TEMPLATES:

$$A_n = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad (n = 1...8)$$

where B_i denotes the control templates, while A_n denotes the feedback matrix (the same for all directions). The algorithm runs on the SKELBWI.bmp and the size of the image is 128×128. The results of the algorithm are plotted below, in Figure 4.5.

The above analogic CNN algorithm had been implemented on the experimental system, based on Virtex-II FPGA with high-level software API (collection of functions), which makes it possible to control the communication of the GAPU

DOI:10.15774/PPKE.ITK.2011.003

between the MicroBlaze core and the array of Falcon / Vector processor elements.

4.6 Device utilization

The experimental system is implemented on the RC203 development board from Celoxica [75], which is equipped with a Xilinx Virtex-II 3000 FPGA including 14 336 slices, 96 18×18 bit signed multipliers, 96 BRAMs and 2×2 MB ZBT SSRAM memory. Using rapid prototyping techniques and high-level hardware description languages such as Handel-C from Celoxica makes it possible to develop optimized architectures much faster, compared to the conventional VHDL or Verilog based RTL-level approaches. During the implementation of the GAPU, Handel-C is located at the top level of the design, while the MicroBlaze core and its modules are wrapped as a low-level system processor macro. Using the Platform Studio integrated development environment [37] from Xilinx supports both the MicroBlaze soft-core and IBM PowerPC hard processor core designs.

The required number of resources of the Falcon Processor Element and the proposed GAPU in different precision are examined (see Figure 4.6 and 4.7).

As shown in Table 4.1, the proposed GAPU, based on a Xilinx MicroBlaze core, requires minimal additional area on the available chip resources at 18-bit state precision, which accuracy is best suited for dedicated multipliers (MULT18$\times$ 18) and block-RAM (BRAM) modules. Though, the GAPU controller occupies four-times as many slices and BRAMs as one Falcon PE does, but only a small fraction of the available resources on the moderate-sized Virtex-II FPGA is used. Due to this consideration, the embedded GAPU does not decrease the number of implementable Falcon processor elements significantly. The number of the implementable FPEs and VPEs is configurable. Hence, by using our XC2V3000 FPGA only 12% of the available slices are utilized, which makes it possible to implement 15 FPE cores, depending on the limited number of BlockRAMs. We can save some additional area by using external ZBT-SRAM modules instead of on-chip BRAMs. Moreover, the speed of the GAPU is close to the clock frequency of a Falcon processing element on Virtex-II architecture (the maximum realizable clock frequency of the MicroBlaze core is shown in case of the state-of-the-art Virtex-6 architecture [37]). If we want to attain the best performance,

DOI:10.15774/PPKE.ITK.2011.003

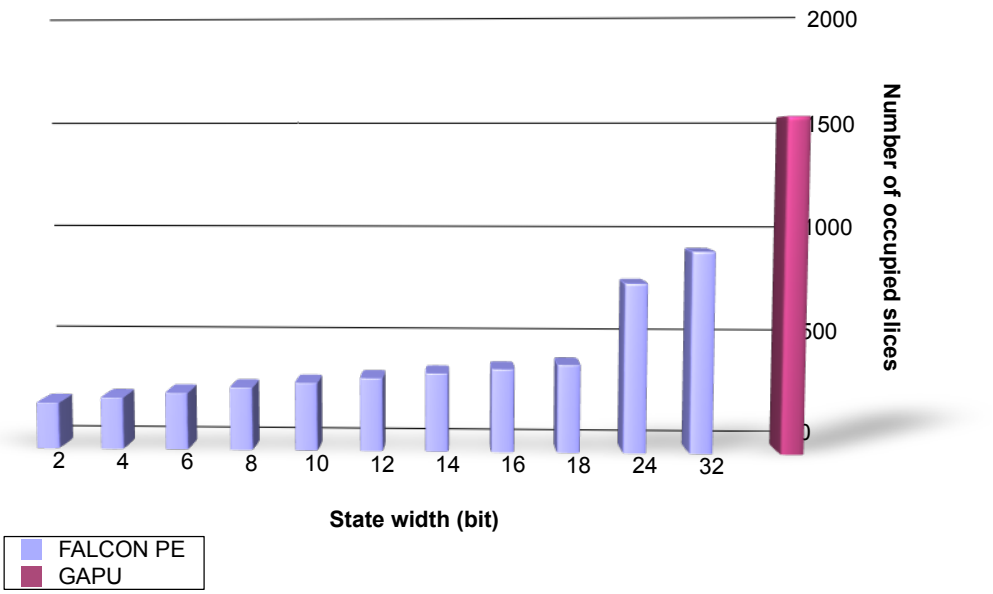


Figure 4.6: Number of required slices in different precision

DOI:10.15774/PPKE.ITK.2011.003

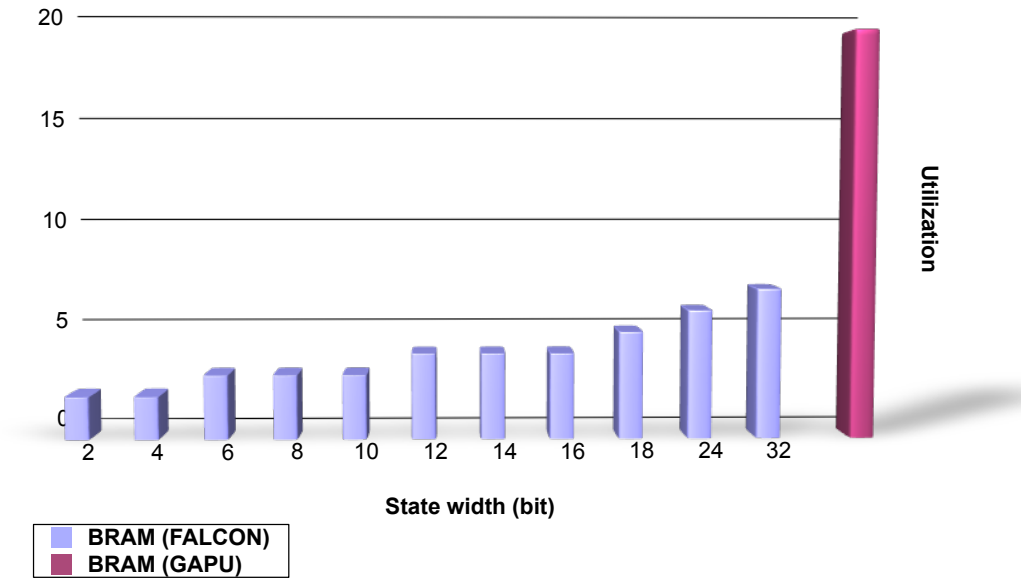


Figure 4.7: Number of required BlockRAMs in different precision

DOI:10.15774/PPKE.ITK.2011.003

Table 4.1: Comparison of a modified Falcon PE and the proposed GAPU in terms of device utilization and achievable clock frequency. The configuration of state width is 18 bit. (The asterisk denotes that in Virtex-5 FPGAs, the slices differently organized, and they contain twice as much LUTs and FlipFlops as the previous generations).

Device utilization and speed	**MicroBlaze GAPU (@18 bit)**	**Falcon PE (@18 bit)**	**Available on XC2V3000**	**Available on XC6VSX475T**
Num. of occupied slices	**1780**	452	14 336	74400*
Num. of BRAMs	**18**	5	96	2128
Num. of MULT18×18s	**0**	9	96	2016
Core frequency [MHz]	**100**	133	133	600 (210)

the currently available largest DSP-specialized Virtex-6 SX475T FPGA has to be used where the GAPU occupies only a minimal additional area (about 2.4% of slices and only 1% of BRAMs). With the use of this device, 160 Falcon PE cores can be implemented in an array, as well. The other large-sized Virtex-II Pro or Virtex4 FX or Virtex5 FX platform FPGAs provide embedded IBM PowerPC405 hard processor core(s) at a higher speed, which may be a good alternative to implement GAPU in further development. There are only rumors about the newest 7th series Xilinx FPGAs, which will embed ARM processor core.

4.7 Results

Considering consecutive analog operations (e.g., the black and white skeletonization above) on a 64×64 image with 18-bit state-, constant- and 9-bit template-precision, the Falcon PE cores perform 10 iterations within 0.307 ms. Without the proposed GAPU extension, due to the slow communication via the parallel port (downloading / uploading the sequence of instructions, templates, and results), the data transfer requires approximately 204.8 ms, while the full computing time

DOI:10.15774/PPKE.ITK.2011.003

Table 4.2: Comparison of the different host interfaces.

	Parallel Port (without GAPU)	**PCI 2.1 (without GAPU)**	**MicroBlaze-OPB (with embedded GAPU)**
Bus speed (MByte/s)	0.4	133	266
CNN core speed (MHz)	133	133	133
Picture (Width×Height)	64×64	64×64	64×64
Num. of iterations	10	10	10
Data size (byte)	4	4	4
Comm. time (ms)	204.8	0.49	0.246
Computing. time (ms)	0.307	0.307	0.307
Full time (ms)	**409.907**	**1.287**	**0.799**
Effectiveness of FPEs	0.30%	38.46%	55.56%

is about 410 ms, as you can see in Table 4.2. In this case, the effective utilization of the Falcon processor is only 0.3%. If we use a prototyping board equipped with a PCI interface, the communication time is reduced to 0.49 ms, while the computation is performed within 1.287 ms.

We can have the best alternative by applying the embedded GAPU where the MicroBlaze core can communicate directly across the OPB bus at a speed of 133 MByte/s, which means that data transfer is accomplished within 0.246 ms. Hence, the Falcon PE can be more effectively (in about 55.56% of the full computing time) utilized when working on analogic operations. Using the embedded GAPU implementation, stand-alone operation and high performance with minimal additional cost in area can be achieved.

With larger images (eg.: 128×128) and with the same 18-bit state-, constant- and 9-bit template-precision, the Falcon PE cores perform 10 iterations within 1.23 ms. In this case, with the proposed GAPU, the Falcon PE works in 71.43% of the full computing time.

4.8 Conclusions

During the long research with my colleague, Dr. Zsolt Vörösházi, we have extended the Falcon architecture with the proposed GAPU on FPGA.

The Falcon architecture extended with the proposed GAPU was successfully

DOI:10.15774/PPKE.ITK.2011.003

designed on FPGA. It provides a fully programmable CNN-UM, on which the most sophisticated and complex analogic algorithms can be executed. It has shown that integrating this embedded GAPU is highly advantageous in case sequences of template operations in an analogic CNN algorithm are performed on large-sized (e.g., 128×128, or even 512×512) images. Due to the modular structure, the GAPU makes the integration easy with the different elaborated, emulated-digital CNN-UM implementations on FPGA, e.g., computing single layer dynamic, globally connected multi-layer computations, or running non-linear templates, as well.

I made recommendations for the structure of the GAPU (precision) to develop an emulated digital CNN-UM. The Falcon processor should be extended with the GAPU, according to the original CNN-UM architecture, in order to execute a more complex algorithm time efficiently. The implemented GAPU should consume minimal area while providing high operating speed to avoid slow down of the Falcon processor, to gain the largest possible computational performance. The GAPU can be built from a properly configured MicroBlaze, or a dedicated PPC, or ARM processor. I made further considerations on the structure of the controller's state registers and configuration of the template and state memory in order to adopt the system for the different kind of Falcon Processing Units. E.g.: Different Falcon units are optimal for black and white or grayscale image processing.

I have developed a new memory organization methodology in order to exploit the possibilities of the latest FPGAs. The dedicated arithmetic units of the new generation FPGAs become faster, but the speed of the embedded processor and bus architecture are evolving slower. The Falcon processor can work on higher operating frequency than the embedded microprocessor and the bus system on the latest FPGAs. I have developed a new architecture, where the embedded microprocessor, the controller circuit, the memory and the Falcon processing unit can be operated on different clock speed. In addition to the internal structural modifications the external memory can be accessed via a dedicated FIFO element. The new architecture makes concurrent access to the external memory possible for the MicroBlaze, the control unit and the Falcon processor.

DOI:10.15774/PPKE.ITK.2011.003

The implemented GAPU architecture, with the modified emulated digital Falcon architecture defines an organic like computing device, which builds up from simply, but large numbered of locally connected elements, which provides a distributed, decentralized, failure tolerant operation of the information processing.

DOI:10.15774/PPKE.ITK.2011.003

Chapter 5

Summary of new scientific results

1. Development of an efficient mapping of the simulation of partial differential equations on inhomogenous and reconfigurable architectures: I have compared the optimal mapping of the simulation of a complex spatio-temporal dynamics on Xilinx Virtex FPGA and on IBM Cell architecture, and I made a framework for that. The framework has been successfully tested by the acceleration of a computational fluid dynamics (CFD) simulation. During the implementation my goal was always to reach the highest possible computational performance. The structure of the accelerator was designed according to this goal while considering the hardware specifications of the different architectures.

   - **I have implemented an effective architecture, in the aspect of area, speed, dissipated power, bandwidth, for solving partial differential equations on structured grid. I have redesigned the arithmetic unit of the Falcon processor according to the discretized version of the partial differential equations optimized for the dedicated elements (BlockRAM, multiplier) of the FPGA.**

     I have developed a process for the optimal bandwidth management between the processing elements and the memory on Xilinx Virtex and on IBM Cell architectures, which makes it possible to continuously supply the processing elements with data.

DOI:10.15774/PPKE.ITK.2011.003

I have successfully confirmed experimentally in both cases, that placing a memory element close to the processor results in a beneficial effect on the computing speed, which provides a minimum one order of magnitude higher speedup independently from the dimension of the problem.

- **I have proved experimentally that one order of magnitude speedup can be achieved between an inhomogenous architecture, like the IBM Cell, and a custom architecture optimized for Xilinx Virtex FPGA using the same area, dissipated power and precision.** During the simulation of CFD on body fitted mesh geometry the Xilinx Virtex 5 SX240T running on 410 MHz is 8 times faster, than the IBM Cell architecture with 8 synergistic processing element running on 3.2 GHz. Their dissipated power and area are in the same range, 85 Watt, 253mm^2 and 30 Watt, 400 mm^2 respectively. **Considering the IBM Cell processor's computing power per watt performance as a unit, computational efficiency of the Xilinx Virtex 5 SX240T FPGA is 22 times higher, while providing 8 times higher performance. The one order of magnitude speedup of the FPGA is owing to the arithmetic units working fully parallel and the number of implementable arithmetic units.** During CFD simulation, the IBM Cell processor and the FPGA based accelerator can achieve 2 and 3 order of magnitude speedup respectively compared to a conventional microprocessor (e.g.: Intel x86 processors).

2. Examination of the precision and the accuracy of partial differential equation solver architectures on FPGA: I have shown in my thesis, that significant speedup can be achieved by decreasing the state precision on FPGA. Engineering applications usually does not require 14-15 digit accuracy, therefore the decreased computational precision can be acceptable. Reduction of the state precision makes it possible to map some particularly complex problems onto an FPGA. I have developed a methodology to specify the minimal required computational precision to reach the maximal computing

DOI:10.15774/PPKE.ITK.2011.003

performance on FPGA where the accuracy of the solution and the grid resolution is given a-priori. The required computational precision can only be determined precisely in infrequent cases, when the exact solution is known.

- **I have elaborated a method to find the minimum required computing precision of the arithmetic units when the step size, spacial resolution and the required accuracy is defined. I have given a tested method to find the precision of the arithmetic unit of a problem, which has analytic solution.** For problems without analytic solution, the reduced precision results can be compared to the 64 bit floating point reference precision. The finest resolution of the grid can also be determined by the method if the desired accuracy is defined.
- **I have shown during the solution of the advection equation (5.2), that higher computing power can be achieved at the expense of the precision.**

$$\frac{\partial u}{\partial t}+\mathrm{c}\frac{\partial u}{\partial x}=0 \tag{5.1}$$

  where t denotes time, u is a conserved property, c is the advection speed. **During the investigation of the arithmetic unit of the advection equation solver the precision is decreased from 40 bit to 29 bit, while area requirements of the architecture are decreased by 20-25% independently from the applied discretization method.** Clock frequency of the arithmetic units does not increase significantly due to the decreased precision, the main source of speedup is the increased number of implementable arithmetic units on the FPGA.
- **I have proved experimentally that area requirements of the arithmetic units can be significantly reduced by using properly normalized fixed point numbers. During the investigation of the advection equation solver architecture, error of the solution of the 33 bit fixed point and the 40 bit floating point (29 bit mantissa) arithmetic unit is in the same order, but**

DOI:10.15774/PPKE.ITK.2011.003

the area required for the arithmetic unit is decreased by 15 times. The main source of speedup is the increased number of implementable arithmetic units on the FPGA, when fixed point arithmetic is used.

3. Implementation of a Global Analogical Programming Unit for emulated digital CNN-UM processor on FPGA architecture: The dynamics of the CNN can be emulated by the Falcon processor with different computing precision, arbitrary sized template on many layers. It should be extended with Global Analogical Programming Unit (GAPU) in order to execute a more complex analogical algorithm time efficiently, additionally a Vector Processor should be attached to accelerate arithmetic and logic operations. The GAPU is not only used during program organizing and I/O peripheral management tasks but it should execute local logic, arithmetic and analog instructions as well. Furthermore, timing and control signals of the Falcon processor should be set correctly by the GAPU.

   The proposed modifications were implemented and verified with a testing example. Due to the implemented modifications and the extension with the GAPU and the Vector Processor, a real image processing system, a Cellular Wave Computer can be developed.

   - **I made recommendations for the structure of the GAPU (precision) to develop an emulated digital CNN-UM.** The Falcon processor should be extended with the GAPU, according to the original CNN-UM architecture, in order to execute a more complex algorithm time efficiently. **The implemented GAPU should consume minimal area while providing high operating speed to avoid slow down of the Falcon processor, to gain the largest possible computational performance. The GAPU can be built from a properly configured MicroBlaze, or a dedicated PPC, or ARM processor. I made further considerations on the structure of the controller's state registers and configuration of the template and state memory in order to adopt the system**

DOI:10.15774/PPKE.ITK.2011.003

for the different kind of Falcon Processing Units. E.g.: Different Falcon units are optimal for black and white or grayscale image processing.

- **I have developed a new architecture, where the embedded microprocessor, the controller circuit, the memory and the Falcon processing unit can be operated on different clock speed. In addition to the internal structural modifications the external memory can be accessed via a dedicated FIFO element. The new architecture makes concurrent access to the external memory possible for the MicroBlaze, the control unit and the Falcon processor.**

  The dedicated arithmetic units of the new generation FPGAs become faster, but the speed of the embedded processor and bus architecture are evolving slower. The Falcon processor can work on higher operating frequency than the embedded microprocessor and the bus system on the latest FPGAs.

5.1 Új tudományos eredmények (*magyar nyelven*)

1. Parciális differenciál egyenletek numerikus szimulá-ciójának optimális leképezése inhomogén és újrakonfigurálható architektúrára: Összehasonlítottam egy komplex tér-időbeli dinamika szimulációjának optimális (felület, idő, disszipált teljesítmény) leképezését Xilinx Virtex FPGA-án és IBM Cell architektúrán, és erre egy keretrendszert alkottam. A keretrendszert sikeresen teszteltem egy CFD szimuláció gyorsításával. Célom mindvégig a lehető leggyorsabb feldolgozás volt. Az architektúra ennek megfelelően lett kialakítva figyelembe véve a hardware sajá-tosságait.

   - **Létrehoztam egy új felület, idő, disszipált teljesítmény, sávszélesség szempontjából hatékony architektúrát parciális differenciál egyenletek strukturált rácson történő megoldására. Újraterveztem a Falcon processzor aritmetikai egységeit a dis-**

DOI:10.15774/PPKE.ITK.2011.003

zkretizált parciális differenciál egyenleteknek megfelelően az FPGA dedikált erőforrásaira (BlockRAM, szorzó) optimalizálva.

Eljárást adtam a processzáló elemek és a memória között a sávszélesség optimális kezelésének problémájára Xilinx Virtex és IBM Cell architektúrákon, amely lehetővé teszi a processzáló elemek folyamatos ellátását adatokkal.

Mindkét esetben kísérletileg sikerült igazolnom, hogy a működési sebességre jótékonyan hat egy processzor közeli tárterület kialakítása, mely a feladat dimenziójától függetlenül legalább egy nagyságrendnyi sebességnövekedést biztosít.

- **Kísérletileg igazoltam, hogy egy kötött architektúra, mint az IBM Cell és egy a Xilinx Virtex FPGA-ra tervezett, optimalizált architektúra között egy nagyságrendi gyorsulást lehetséges elérni azonos felület, disszipált teljesítmény és pontosság esetén.** A Xilinx Virtex 5 SX240T 410 MHz-en 8-szor gyorsabb volt, mint a 8 db szinergikus processzor elemet tartalmazó IBM Cell architektúra 3.2 GHz-en CFD-t szimulálva görbült hálón. A disszipált teljesítményük és felületük azonos nagyság-rendbe tartozik, rendre 85 Watt, 253 mm^2 és 30 Watt, 400 mm^2. **Az IBM Cell processzor egy wattra jutó számítási teljesítményét egységnyinek tekintve a Xilinx Virtex 5 SX240T FPGA 8-szoros sebességnövekedés mellett a számítás hatékonysága 22 szeres. Az egy nagyságrendnyi sebességkülönbség köszönhető az FPGA teljesen párhuzamosan működő műveletvégző egységeinek, illetve a megvalósítható aritmetikai egységek számának.** A ma használatos általános célú mikroprocesszorokhoz (pl.: Intel x86 processzorok) képest az IBM Cell processzor CFD-t szimulálva 2, az FPGA alapú gyorsító 3 nagyság-rendnyi gyorsulást ért el.

2. Parciális differenciál egyenleteket megoldó architek-túrák pontosságának vizsgálata FPGA-n: Tézisemben megmutattam, hogy az állapot pontosságának csökkentésével jelentős sebességnövekedés érhető el. Ez a számítási

DOI:10.15774/PPKE.ITK.2011.003

pontosság csökkentés elfogadható lehet, hiszen nem minden mérnöki alkalmazás követel meg 14-15 helyiértéknyi pontosságot. A pontosság csökkentésével néhány különösen bonyolult probléma is leképezhető az FPGA-ra. A megoldás előírt pontossága és a rácstávolság ismeretének tükrében kidolgoztam egy eljárást, amivel a szükséges minimális számábrázolás pontosság megadható, ezáltal az FPGA-val adható legnagyobb számí-tási teljesítmény érhető el. Természetesen a szükséges pontosság meghatározása csak egzakt megoldás esetén adató meg pontosan és ez kevés esetben áll rendelkezésünkre.

- **Eljárást adtam arra vonatkozólag, hogy hogyan határozható meg az aritmetika minimális szükséges pontossága ismert lépésköz, térbeli felbontás és a megoldás elvárt pontossága esetén. Analitikus megoldással rendelkező probléma vizsgálata esetén tesztelt eljárást adtam a problémát megoldó architektúra aritmetikai egységeinek pontosságára.** Azon problé-mák esetén, ahol nincs analitikus megoldása a prob-lémának, ezen csökkentett pontosságú eredményeket a referenciának tekinthető 64 bites lebegőpontos pontossághoz lehet viszonyítani. A módszer továbbá alkalmas arra is, hogy a kerekítési és a levágási hibák ismeretével meghatározható adott pontosság mellett a rács legfinomabb felbontása.

- **Megmutattam, hogy az advekciót leíró parciális differenciálegyenlet (5.2) megoldása során hogyan lehet a pontosság rovására kevesebb erőforrás felhasz-nálásával nagyobb teljesítményt elérni.**

$$\frac{\partial u}{\partial t}+\mathrm{c}\frac{\partial u}{\partial x}=0 \tag{5.2}$$

  ahol a t az időt, u egy fentartósági tulajdonságot, c az advekció sebességét jelenti. **A pontosság vizsgá-latára használt advekciós egyenletet megoldó architektúra esetén, az aritmetikai egységek pontossá-gának 40 bitről 29 bitre csökkentésével a felhasznált felületigénye 20-25%-al csökkent az alkalmazott diszkretizációs**

DOI:10.15774/PPKE.ITK.2011.003

eljárástól függetlenül. Sebességnövekedés jelentős része az FPGA-ra implementálható művelet-végző egységeknek nagyobb számának köszönhető, az órajel pedig számottevően nem növekszik a pontosság csökkentésével.

- **Kísérlettel igazoltam, hogy megfelelő normalizálás esetén a fixpontos aritmetika adott pontosság mellett további felületnyereséggel jár. A pontosság vizsgálatára használt advekciós egyenletet megoldó architektúra esetén a 33 bit pontos fix és 40 bit pontos lebegőpontos (29 bit mantissza) aritmetikai egység megoldás hibája ugyanabba a nagyságrendbe esik, ellenben az aritmetikai egység felülete fixpontos esetben 15-ödére csökken.** Fixpontos aritmetikát használva a sebességnövekedés jelentős része az FPGA-ra implementálható műveletvégző egységek nagyobb számának köszönhető.

3. Globális Analogikai Vezérlő Egység implementációja emulált-digitális CNN processzorhoz FPGA architektúrán: A Falcon processzor különböző számábrá-zolási pontossággal, különböző méretű template-kkel, több rétegben tudja a CNN dinamikát kiszámolni. Annak érdekében, hogy komplexebb analogikai algoritmusokat is időben hatékonyan végre lehessen hajtani, ki kellett egészíteni egy Globális Analogikai Vezérlő Egységgel (GAPU), továbbá az aritmetikai és logikai műveletek elvégzésére egy Vektor Processzort kellett készíteni. A GAPU-nak nemcsak programszervezési és I/O periféria kezelési feladata van, hanem lokális logikai és aritmetikai műveleteket, valamint analóg utasításokat is tudnia kell kezelni. A GAPU feladata továbbá a Falcon processzor megfelelő időzítő- és vezérlő-jeleinek beállítása is.

   Az általam javasolt módosítások implementálásra is kerültek és egy pédán keresztül tesztelve is lettek. Az eszközölt módosításoknak és a GAPU, illetve a Vektor Processzorral való kiegészítésnek köszön-hetően létrehozható egy önálló képfeldolgozó rendszer, egy Celluláris Hullámszámítógép.

   - **Emulált digitális CNN-UM kialakításához javaslatokat tettem a GAPU felépítésére (pontosságára) vonatkozólag.** A Fal-

DOI:10.15774/PPKE.ITK.2011.003

con processzor kibővítése a GAPU-val, annak érdekében, hogy komplexebb algoritmusokat is végre tudjon időben hatékonyan hajtani, kézenfekvő volt az eredeti CNN-UM-nek megfelelően. **A GAPU-t úgy kellett implementálni, hogy ne foglaljon el sok helyet az FPGA-n, és ne lassítsa számot-tevően a Falcon processzor működési sebességét, hiszen a cél a lehető legnagyobb számítási teljesítmény elérése. A GAPU szerepének betöltésére egy jól konfigurált MicroBlaze-t, dedikált PPC-t, vagy ARM-ot javasoltam használni. Továbbá megfontolásokat tettem a vezérlő és állapot regiszterek fajtájára és a template és állapot memória konfigurálhatóságára is annak érdekében, hogy a rendszer adaptálható legyen a hozzá csatlakoztatott Falcon Processzáló Egység fajtájához.** Pl.: ha csak fekete-fehér képen dolgozunk, vagy szürkeárnyalatos képeken is akarunk műveleteket végezni, különböző Falcon egységet célszerű használni.

- **Olyan új architektúrát dolgoztam ki ami lehetővé teszi, hogy a beágyazott mikroprocesszor, a vezérlő áram-körök, a memória és a Falcon processzáló egység különböző órajeleken működhessen. A belső struktúra átalakítása mellet a külső elérést is biztosítottam egy dedikált FIFO-n keresztül. Az új architektúra lehetővé teszi a MicroBlaze, a vezérlő egység és a Falcon processzor számára a külső memória konkurrens hozzáférését.**

  Az új generációs FPGA-k dedikált műveletvégző egységei gyorsulnak, de ezt nem követi a beágyazott processzor-architektúra és az alkalmazott busz sebessége. Az új FPGA-k esetében a Falcon processzor nagyobb működési sebességre is képes, mint a mellette be-ágyazott mikroprocesszor- és busz-rendszer.

DOI:10.15774/PPKE.ITK.2011.003

5.2 Application of the results

5.2.1 Application of the Fluid Flow Simulation

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), being a sigma coordinate model in which the vertical coordinate is scaled on the water column depth.

Computational Fluid Dynamics (CFD) is the scientific modeling the temporal evolution of gas and fluid flows by exploiting the enormous processing power of computer technology. Simulation of fluid flow over complex shaped objects currently requires several weeks of computing time on high performance supercomputers. The developed CFD simulation architecture, implemented on FPGA, is several order of magnitude faster than todays microprocessors.

5.2.2 Examining the accuracy of the results

In real life engineering application double precision floating point numbers are used for computations to avoid issues of roundoff error. However it is worth to examine the required precision, if the computing resources, power dissipation or size is limited or the computation should be carried out in real time. The speed of the partial differential equation solver architecture implemented on FPGA can be greatly increase, if we decrease the precision of the solver architecture, consequently more processing unit can be implemented on the same area. This thesis is useful if we want to investigate the limitation of a real time computation. I have examined a simplified advection equation solver architecture, where the analytic

DOI:10.15774/PPKE.ITK.2011.003

solution is known. With the minimal modification of such problems (which has analytic solution), the computed precision is remaining probably acceptable with a similar problem, which has no analytic solution.

5.2.3 The importance of Global Analogic Programming Unit

In order to provide high flexibility in CNN computations, it is interesting how we can reach large performance by connecting locally a lot of simple and relatively low-speed parallel processing elements, which are organized in a regular array. The large variety of configurable parameters of this architecture (such as state- and template-precision, size of templates, number of rows and columns of processing elements, number of layers, size of pictures, etc.) allows us to arrange an implementation, which is best suited to the target application (e.g. image/video processing). So far, without the GAPU extension, when solving different types of PDEs, a single set of CNN template operations has been implemented on the host PC: by downloading the image onto the FPGA board (across a quite slow parallel port), computing the transient, and finally uploading the result back to the host computer where logical, arithmetic and program organizing steps were executed.

Reconfigurable CNN-UM implementation on FPGAs may also mean a possible breakthrough point towards industrial applications, due to their simplicity, high computing power, minimal cost, and fast prototyping.

DOI:10.15774/PPKE.ITK.2011.003

DOI:10.15774/PPKE.ITK.2011.003

References

The author's journal publications

[1] Z. Nagy, L. Kék, Z. Kincses, A. Kiss, and P. Szolgay, "Toward Exploitation of Cell Multi-processor Array in Time-consuming Applications by Using CNN Model," *International Journal of Circuit Theory and Applications*, vol. 36, no. 5-6, pp. 605–622, 2008.

[2] Z. Vörösházi, A. Kiss, Z. Nagy, and P. Szolgay, "Implementation of Embedded Emulated-Digital CNN-UM Global Analogic Programming Unit on FPGA and its Application," *International Journal of Circuit Theory and Applications*, vol. 36, no. 5-6, pp. 589–603, 2008.

The author's international conference publications

[3] Z. Vörösházi, Z. Nagy, A. Kiss, and P. Szolgay, "An Embedded CNN-UM Global Analogic Programming Unit Implementation on FPGA," in *Proceedings of the 10th IEEE International Workshop on Cellular Neural Networks and their Applications*, (Istanbul, Turkey), CNNA2006, August 2006.

[4] Z. Vörösházi, A. Kiss, Z. Nagy, and P. Szolgay, "FPGA Based Emulated-Digital CNN-UM Implementation with GAPU," in *Proc. of CNNA'2008*, (Santiago de Compostella), pp. 175–180, 2008.

[5] Z. Nagy, L. Kék, Z. Kincses, A. Kiss, and P. Szolgay, "Toward Exploitation of Cell Multi-Processor Array in Time-Consuming Applications by Using CNN Model," in *Proc. of CNNA'2008*, (Santiago de Compostella), pp. 157–162, 2008.

DOI:10.15774/PPKE.ITK.2011.003

[6] Z. Vörösházi, A. Kiss, Z. Nagy, and P. Szolgay, "A Standalone FPGA Based Emulated-Digital CNN-UM System," in *Proc. of CNNA'2008*, (Santiago de Compostella), 2008.

[7] Z. Nagy, A. Kiss, S. Kocsárdi, and Á. Csík, "Supersonic Flow Simulation on IBM Cell Processor Based Emulated Digital Cellular Neural Networks," in *Proc. of ISCAS'2009*, (Taipei, Taiwan), pp. 1225–1228, 2009.

[8] Z. Nagy, A. Kiss, S. Kocsárdi, and Á. Csík, "Computational Fluid Flow Simulation on Body Fitted Mesh Geometry with IBM Cell Broadband Engine Architecture," in *Proc. of ECCTD'2009*, (Antalya, Turkey), pp. 827–830, 2009.

[9] Z. Nagy, A. Kiss, S. Kocsárdi, M. Retek, Á. Csík, and P. Szolgay, "A Supersonic Flow Simulation on IBM Cell Processor Based Emulated Digital Cellular Neural Networks," in *Proc. of CMFF'2009*, (Budapest, Hungary), pp. 502–509, 2009.

[10] A. Kiss and Z. Nagy, "Computational Fluid Flow Simulation on Body Fitted Mesh Geometry with FPGA Based Emulated Digital Cellular Neural Networks," in *Proceedings of 12th International Workshop on Cellular Nanoscale Networks and their Applications*, (Berkeley, CA, USA), CNNA2010, 2010.

[11] L. Füredi, Z. Nagy, A. Kiss, and P. Szolgay, "An Improved Emulated Digital CNN Architecture for High Performance FPGAs," in *Proceedings of the 2010 International Symposium on Nonlinear Theory and its Applications*, (Krakow, Poland), pp. 103–106, NOLTA2010, 2010.

[12] C. Nemes, Z. Nagy, M. Ruszinkó, A. Kiss, and P. Szolgay, "Mapping of High Performance Data-Flow Graphs into Programmable Logic Devices," in *Proceedings of the 2010 International Symposium on Nonlinear Theory and its Applications*, (Krakow, Poland), pp. 99–102, NOLTA2010, 2010.

DOI:10.15774/PPKE.ITK.2011.003

Publications connected to the dissertation

[13] "IBM Inc." [Online] http.ibm.com. 1

[14] "INTEL Inc." [Online] http.intel.com. 1

[15] P. Szolgay, G. Vörös, and G. Erőss, "On the Applications of the Cellular Neural Network Paradigm in Mechanical Vibrating System," *IEEE Transactions Circuits and Systems-I, Fundamental Theory and Applications*, vol. 40, no. 3, pp. 222–227, 1993. 1, 2.2

[16] Z. Nagy and P. Szolgay, "Numerical Solution of a Class of PDEs by Using Emulated Digital CNN-UM on FPGAs," *Proceeding of the 16th European Conference on Circuits Theory and Design, Krakow, Poland*, vol. II, pp. 181–184, 2003. 1, 2.2

[17] T. Roska and L. O. Chua, "The CNN Universal Machine: an Analogic Array Computer," *IEEE Transactions on Circuits and Systems-II*, vol. 40, pp. 163–173, 1993. 1.1, 1.1.1, 2.1.1

[18] Zs.Vörösházi, Z.Nagy, and P.Szolgay, "An advanced emulated digital retina model on fpga to implement a real-time test environment," *ISCAS*, pp. 1949–1952, 2006. 1.1, 4.1

[19] P. Sonkoly, P. Kozma, Z. Nagy, and P. Szolgay, "Elastic wave propagation modeling on emulated digital cnn-um architecture," *9th IEEE International Workshop on Cellular Neural Networks and their Applications*, pp. 126–129, 2005. 1.1, 4.1

[20] S. E. et.al., "A VLSI-Oriented Continuous-Time CNN Model," *International Journal of Circuit Theory and Applications*, vol. 24, pp. 341–356, 1996. 1.1.1, 1.3

[21] "Cellular Wave Computing Library." [Online] http://cnn-technology.itk.ppke.hu/, 2007. 1.1.2, 1.1.2, 2.1.1.2

DOI:10.15774/PPKE.ITK.2011.003

[22] T. Roska and L. O. Chua, "The cnn universal machine: An analogic array computer," *IEEE Trans. on Circuits and Systems-II.*, vol. 40, pp. 163–173, march 1993. 1.2, 1.3, 4.1

[23] G. Linán, S. Espejo, R. Dominguez-Castro, and A. Rodriguez-Vázquez, "Ace4k: an analog i/o 64 x 64 visual microprocessor chip with 7-bit analog accuracy," *Int J. CTA*, vol. 30, pp. 89–116, 2002. 1.3, 4.1

[24] AnaFocus Ltd, [online] http://www.anafocus.com, *eye-RIS v1.0/v2.0 Datasheet*, 2004. 1.3, 4.1

[25] G. Linán, R. Dominguez-Castro, S. Espejo, and A. Rodriguez-Vázquez, "Ace16k: A programmable focal plane vision processor with 128x128 resolution," *Proc. of the 15th European Conference on Circuit Theory and Design*, vol. 1, pp. 345–348, 2001. 1.3, 4.1

[26] T. Hidvegi, P. Keresztes, and P. Szolgay, "An accelerated digital cnn-um (castle) architecture by using the pipe-line technique," *Proc. of the 15th European Conference on Circuit Theory and Design*, 2002. 1.3, 4.1

[27] T. Roska, G. Bartfai, P. Szolgay, T. Sziranyi, A. Radvanyi, T. Kozek, and Z. Ugray, "A hardware accelerator board for cellular neural networks: Cnn-hac," in *IEEE International Workshop on Cellular Neural Networks and their Applications*, pp. 160 – 168, 1990. 1.3

[28] Z. Nagy, Z. Kincses, L. Kék, and P. Szolgay, "CNN Model on Cell Multiprocessor Array," *Proceedings of the 18th European Conference on Circuit Theory and Design, Seville, Spain*, pp. 276–279, 2007. 1.3, 2.2

[29] B. G. Soós, Á. Rák, J. Veres, , and G. Cserey, "Gpu boosted cnn simulator library for graphical flow-based programmability," *EURASIP Journal on Advances in Signal Processing*, vol. 2009, no. Article ID 930619, p. 11, 2009. 1.3

[30] Z. Nagy, Z. Vörösházi, and P. Szolgay, "Emulated Digital CNN-UM Solution of Partial Differential Equations," *International Journal of Circuit Theory and Applications*, vol. 34, no. 4, pp. 445–470, 2006. 1.3, 2.2, 2.3.2.6, 3

DOI:10.15774/PPKE.ITK.2011.003

[31] P. Keresztes, Á. Zarándy, T. Roska, P. Szolgay, T. Hídvégi, P. Jónás, and A. Katona, "An emulated digital cnn implementation," *International Journal of VLSI Signal Processing*, September 1999. 1.3, 4.1

[32] "Celoxica ltd. homepage." [online] http://www.celoxica.com, 2010. 1.3, 4.4

[33] M. B. Gokhale and P. S. Graham, *Reconfigurable Computing : Accelerating Computation with Field-Programmable Gate Arrays*. Springer, 2005. 1.4

[34] "International roadmap for semiconductors." [Online] http://public.itrs.net. 1.4

[35] "Microsemi SoC Products Group." [Online] www.actel.com, 2011. 1.4.2

[36] "Quicklogic." [Online] www.quicklogic.com, 2011. 1.4.2

[37] "Xilinx Inc.." [Online] http://www.xilinx.com/, 2011. 1.4.2, 3.3, 3.4, 4.3.1, 4.6, 4.6

[38] "Altera Corporation." [Online] www.altera.com, 2011. 1.4.2

[39] 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*, vol. 49, no. 4, pp. 589–604, 2005. 1.5.1

[40] "ClearSpeed Inc.." [Online] http://www.clearspeed.com/, 2011. 1.6

[41] "MathStar Inc.." [Online] http://www.mathstar.com/, 2008. 1.6

[42] "Tilera Inc.." [Online] http://www.tilera.com/, 2011. 1.6

[43] T. Kozek, L. O. Chua, T. Roska, D. Wolf, R. Tetzlaff, F. Puffer, and K. Lotz, "Simulating nonlinear waves and partial differential equations via cnn - part ii: typical examples," *IEEE, Trans. Circuits Syst.*, vol. 42, no. 816-820, 1995. 2.1

[44] T. Roska, L. O. Chua, D. Wolf, T. Kozek, R. Tetzlaff, and F. Puffer, "Simulating nonlinear waves and partial differential equations via cnn - part i: basic techniques," *IEEE, Trans. Circuits Syst.-I*, vol. 42, pp. 807–815, 1995. 2.1

DOI:10.15774/PPKE.ITK.2011.003

[45] 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. 2.2, 4.1, 4.2

[46] Z. Nagy and P. Szolgay, "Solving Partial Differential Equations on Emulated Digital CNN-UM Architectures," *Functional Differential Equations*, vol. 13, pp. 61–87, 2006. 2.2, 4.1

[47] 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. 2.2

[48] "Harvard ocean prediction system." [Online] http://www.gfdl.noaa.gov/ocean-model, 2011. 2.2

[49] "Parallel ocean program." [Online] http://climate.lanl.gov/Models/POP/, 2011. 2.2

[50] "Mit general circulation model." [Online] http://mitgcm.org/, 2011. 2.2

[51] "The Princeton Ocean Model (POM)." [Online] http://www.aos.princeton.edu/aos, 2007. 2.2, 2.2

[52] V. Arakawa, A.; Lamb *Methods of Computational Physics*, pp. 173–165, 1977. 2.2

[53] C. K. Chu, "Numerical methods in fluid mechanics," *Advances in Applied Mechanics*, 1978. 2.3.1

[54] J. W. Thomas, "Numerical partial differential equations: Finite difference methods," *Texts in Applied Mathematics*, 1995. 2.3.1

[55] S. Kocsárdi, Z. Nagy, Á. Csík, and P. Szolgay, "Simulation of 2D inviscid, adiabatic, compressible flows on emulated digital CNN-UM," *International Journal of Circuit Theory and Applications*, vol. 37, no. 4, pp. 569–585, 2009. 2.3.1, 3.1

DOI:10.15774/PPKE.ITK.2011.003

[56] S. Kocsárdi, Z. Nagy, Á. Csík, and P. Szolgay, "Simulation of two-dimensional inviscid, adiabatic, compressibleflows on emulated digital CNN-UM," *International Journal of Circuit Theory and Applications*, vol. DOI:10.1002/cta.565, 2008. 2.3.1, 2.3.2.3

[57] S. Kocsárdi, Z. Nagy, Á. Csík, and P. Szolgay, "Two-dimensional compressible flow simulation on emulated digital CNN-UM," in *Proc. IEEE 11th International Workshop on Cellular Neural Networks and their Applications (CNNA'08)*, (Santiago de Compostella, Spain), pp. 169–174, jul 2008. 2.3.1, 2.3.2.3

[58] G. K. Batchelor, *An Introduction to Fluid Dynamics*. Cambridge University Press, 1967. 2.3.2

[59] D. J. Acheson, "Elementary fluid dynamics," *Oxford Applied Mathematics and Computing Science Series*, 1990. 2.3.2

[60] J. D. Anderson, *Computational Fluid Dynamics - The Basics with Applications*. McGraw Hill, 1995. 2.3.2

[61] T. J. Chung, *Computational Fluid Dynamics*. Cambridge University Press, 2002. 2.3.2

[62] K. Morton and D. Mayers, *Numerical Solution of Partial Differential Equations, An Introduction.* Cambridge University Press, 2005. 2.3.2.1

[63] G. A. Constantinides and G. J. Woeginger, "The complexity of multiple wordlength assignment," *Applied Mathematics Letters*, vol. 15, pp. 137 – 140, 2002. 3

[64] "Mentor Graphics Inc.." [Online] http://www.mentor.com, 2011. 3.3

[65] "The GNU MPFR Library." [Online] http://www.mpfr.org, 2011. 3.3

[66] "Alpha Data Inc.." [Online] http://www.aplha-data.com, 2011. 3.4

[67] "Intel performance libraries homepage." [online] http://www.intel.com/software/product/perflib. 3.4

DOI:10.15774/PPKE.ITK.2011.003

[68] L. O. Chua and L. Yang, "Cellular neural networks: Theory and applications," *IEEE Trans. On Circuits and Systems*, vol. 35, pp. 1257–1290, 1988. 4.1, 4.2, 4.3.2

[69] Á. Zarándy, P. Földesy, P. Szolgay, S. Tõkés, and C. R. T. Roska, "Various implementations of topographic, sensory, cellular wawe computers," *ISCAS, IEEE international symposium on circuits and systems*, pp. 5802–5804, 2005. 4.1

[70] T. Roska, J. M. Cruz, and L. O. Chua, "A fast, complex and efficient test implementation of the cnn universal machine," *Proc. of the IEEE CNNA-94*, pp. 61–66, 1994. 4.1

[71] C. Rekeczky, T. Serrano-Gotarredona, T. Roska, and A. Rodríguez-Vázquez, "A stored program 2nd order/3-layer complex cell cnn-um," *Proc. of the Sixth IEEE International Workshop on Cellular Neural Networks and their Applications*, pp. 219–224, may 2000. 4.1

[72] Z. Kincses, Z. Nagy, and P. Szolgay, "Implementation of nonlinear template runner emulated digital cnn-um on fpga," *10th IEEE International Workshop on Cellular Neural Networks and their Applications*, pp. 126–129, 2006. 4.1

[73] Z. Nagy and P. Szolgay, "Emulated digital cnn-um implementation of a barotropic ocean model," *Proceedings of the International Joint Conference on Neural Networks*, july 2004. 4.1

[74] S. Kocsárdi, Z. Nagy, and P. Szolgay, "Emulated digital cnn solution for two dimensional compressible flows," *Proc. of the 18th European Conference on Circuit Theory and Design*, vol. 1, pp. 288–291, 2007. 4.1

[75] "Celoxica rc203 datasheet." [Online] http://www.celoxica.com/techlib/files/CEL-W0504181A26-52.pdf. 4.4, 4.6

[76] L. Kék, K.Karacs, and T.Roska, *Cellular Wave Computing Library: Templates, Algorithms, and Programs*. Budapest: MTA-SZTAKI, version 2.1 ed., 2007. 4.5