Hindawi Publishing Corporation
EURASIP Journal on Advances in Signal Processing
Volume 2009, Article ID 923404, 11 pages
doi:10.1155/2009/923404

Research Article

Simulation of Two-Dimensional Supersonic Flows on Emulated-Digital CNN-UM

Sándor Kocsárdi,[1] Zoltán Nagy,[2] Árpád Csík,[3] and Péter Szolgay[2, 4]

[1] *Department of Image Processing and Neurocomputing, Faculty of Information Technology, University of Pannonia, Egyetem 10, 8200 Veszprém, Hungary*
[2] *Cellular Sensory and Wave Computing Laboratory, Computer and Automation Research Institute, Hungarian Academy of Sciences, 1518 Budapest, Hungary*
[3] *Department of Mathematics and Computational Sciences, Széchenyi István University, 9026 Győr, Hungary*
[4] *Faculty of Information Technology, Pázmány Péter Catholic University, 1083 Budapest, Hungary*

Correspondence should be addressed to Sándor Kocsárdi, skocso@vision.vein.hu

Received 25 September 2008; Accepted 7 January 2009

Recommended by Victor M. Brea

Computational fluid dynamics (CFD) is the scientific modeling of 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. A CNN-UM-based solver of 2D inviscid, adiabatic, and compressible fluids will be presented. The governing partial differential equations (PDEs) are solved by using first- and second-order numerical methods. Unfortunately, the necessity of the coupled multilayered computational structure with nonlinear, space-variant templates does not make it possible to utilize the huge computing power of the analog CNN-UM chips. To improve the performance of our solution, emulated digital CNN-UM implemented on FPGA has been used. Properties of the implemented specialized architecture is examined in terms of area, speed, and accuracy.

1. Introduction

The CNN paradigm is a natural framework to describe the behavior of locally interconnected dynamical systems which have an array structure [1]. Therefore, it possesses an inherent potential in the fields of computational fluid dynamics and numerical analysis [2]. Unfortunately, analog CNN-UM chips suffer from technical limitations diminishing their efficiency in such practical applications. Their most notable deficiencies are the low precision (8 bits) and restricted usability in applications requiring nonlinear, space-variant templates in a multilayered structure. However, by implementing the concepts behind the CNN-UM technology on reconfigurable architectures, the cell model can be modified according to the numerical simulation of the physical phenomena under consideration [3, 4]. Simulation of a 2D compressible flow on CNN-UM was reported in [5] but this solution used customized floating-point number representation inside the arithmetic unit. Unfortunately, area requirements of the floating-point arithmetic units are quite high, therefore, parallelism of the arithmetic unit needs to be reduced which has a negative impact on computing performance.

In this paper, we focus on the numerical solution of the same hyperbolic system of the nonlinear Euler equations but using fixed-point numbers. Our aim is to find some optimal computational architecture satisfying the functional requirements with minimal required precision, while driving computing power toward its maximum level. Thus, we intend to perform the operations with the highest possible parallelism.

The structure of the paper is the following. In Section 2, we recall the theoretical bases of compressible, adiabatic fluid flows. The details of the numerical discretization technique are described in Section 3. The optimized Falcon processor with the CNN templates and the optimized fixed-point arithmetic unit are given in Sections 4 and 5. In Section 6, the accuracy analysis of the fixed- and floating-point solutions is presented and the features of their implementation on FPGA units are investigated. Finally, conclusions are drawn in Section 7.

2. Fluid Flows

A wide range of industrial processes and scientific phenomena involve gas or fluids flows over complex obstacles, for example, air flow around vehicles and buildings and the flow of water in the oceans or liquid in BioMEMS. In engineering applications, the temporal evolution of nonideal, compressible fluids is quite often modeled by the system of Navier-Stokes 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 these nonideal processes and assuming adiabatic variations, we obtain the Euler equations [6, 7], describing the dynamics of dissipation-free, inviscid, compressible fluids. They are a coupled set of nonlinear hyperbolic partial differential equations, in conservative form expressed as

$$\begin{aligned}\frac{\partial\rho}{\partial t}+\nabla\cdot(\rho\mathbf{v})&=0,\\ \frac{\partial(\rho\mathbf{v})}{\partial t}+\nabla\cdot(\rho\mathbf{v}\mathbf{v}+\hat{I}p)&=0,\\ \frac{\partial E}{\partial t}+\nabla\cdot((E+p)\mathbf{v})&=0,\end{aligned}\tag{1}$$

where t denotes time, ∇ is the nabla operator, ρ is the density, u, v are the x- and y-component of the 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

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

In (2), the value of the ratio of specific heats is taken to be $\gamma = 1.4$. For later use, we introduce the conservative state vector $\mathbf{U}=[\rho,\rho u,\rho v,E]^T$, the set of primitive variables $\mathbf{P}=[\rho,u,v,E]^T$, and the speed of sound $c=\sqrt{\gamma p/\rho}$. It is also convenient to merge (1) into hyperbolic conservation law form in terms of U and the flux tensor,

$$\mathbf{F}=\begin{pmatrix}\rho\mathbf{v}\\ \rho\mathbf{v}\mathbf{v}+Ip\\ (E+p)\mathbf{v}\end{pmatrix},\tag{3}$$

as

$$\frac{\partial U}{\partial t}+\nabla\cdot\mathbf{F}=0.\tag{4}$$

3. 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 discretization of the governing equations over structured grids employing a simple numerical flux function. Indeed, the corresponding rectangular arrangement of information and the choice of multilevel 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.

3.1. The Geometry of the Mesh. For the sake of simplicity, in this paper, we only consider rectangular computational domains labeled by Ω. The sides of the rectangle are a and b units long. We divide Ω into $M\times N$ nonoverlapping rectangular finite volumes (cells) of equal sizes. The volume situated in the ith column and the jth row is indexed by (i,j). The resolution of the mesh in the x- and the y-directions coinciding with the length of the cells' edges are $\Delta x=a/M$ and $\Delta y=b/N$, thus the volume of the cell (i,j) is $V_{i,j}$. Following the finite volume methodology, we store all components of the volume-averaged state vector $U_{i,j}$ at the mass center of cell (i,j).

3.2. The Discretization Scheme. Application of the finite volume discretization method leads to the following semidiscrete form of governing equations (4)

$$\frac{dU_{i,j}}{dt}=-\frac{1}{V_{i,j}}\sum_f\mathbf{F}_f\cdot\mathbf{n}_f,\tag{5}$$

where the summation is meant for all 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-axis is normal to f (see Figure 1). 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 $\mathbf{F}(F_x)$ multiplied by the area of the face. In order to stabilize the solution procedure, artificial dissipation has to be introduced into the scheme. According to the standard procedure, this is achieved by replacing the physical flux tensor by the numerical flux function F^N containing 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. In this paper, we employ the simple and robust Lax-Friedrichs numerical flux function defined as

$$F^N=\frac{F_\mathrm{L}+F_\mathrm{R}}{2}-(|\overline{u}|+\overline{c})\frac{U_\mathrm{R}-U_\mathrm{L}}{2}.\tag{6}$$

In the last equation, $F_L=F_x(U_L)$ and $F_R=F_x(U_R)$ and notations $|\overline{u}|$ and $|\overline{c}|$ represent the average value of the u velocity component and the speed of sound at an interface, respectively. The temporal derivative is discretized by the first-order forward Euler method

$$\frac{dU_{i,j}}{dt}=\frac{U_{i,j}^{n+1}-U_{i,j}^{n}}{\Delta t},\tag{7}$$

where $U_{i,j}^n$ is the known value of the state vector at time level n, $U_{i,j}^{n+1}$ is the unknown value of the state vector at time level $n+1$, and Δt is the time step.

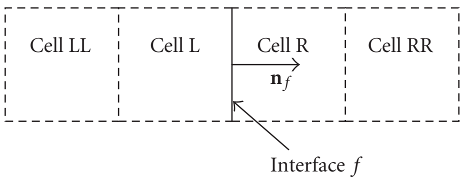

FIGURE 1: Interface with the normal vector and the cells required in the computation.

By working out the algebra described so far, it leads to the discrete form of the governing equations to compute the numerical flux term F and the dissipation term D,

$$\begin{aligned}
F_i^{\rho,n} &= \frac{\rho u_C^n + \rho u_i^n}{2}, \quad i = E, W,\\
F_i^{\rho u,n} &= \frac{(\rho u^2+p)_C^n + (\rho u^2+p)_i^n}{2}, \quad i = E, W,\\
F_i^{\rho u,n} &= \frac{\rho u v_C^n + \rho u v_i^n}{2}, \quad i = N, S,\\
F_i^{\rho v,n} &= \frac{\rho u v_C^n + \rho u v_i^n}{2}, \quad i = E, W,\\
F_i^{\rho v,n} &= \frac{(\rho v^2+p)_C^n + (\rho v^2+p)_i^n}{2}, \quad i = N, S,\\
F_i^{E,n} &= \frac{(E+p)u_C^n + (E+p)u_i^n}{2}, \quad i = E, W,\\
F_i^{E,n} &= \frac{(E+p)v_C^n + (E+p)v_i^n}{2}, \quad i = N, S,\\
D_i^{\rho,n} &= (|\overline{u}| + \overline{c})\frac{\rho_i^n - \rho_C^n}{2}, \quad i = E, N,\\
D_i^{\rho,n} &= (|\overline{u}| + \overline{c})\frac{\rho_C^n - \rho_i^n}{2}, \quad i = W, S,\\
D_i^{\rho u,n} &= (|\overline{u}| + \overline{c})\frac{\rho u_i^n - \rho u_C^n}{2}, \quad i = E, N,\\
D_i^{\rho u,n} &= (|\overline{u}| + \overline{c})\frac{\rho u_C^n - \rho u_i^n}{2}, \quad i = W, S,\\
D_i^{\rho v,n} &= (|\overline{u}| + \overline{c})\frac{\rho v_i^n - \rho v_C^n}{2}, \quad i = E, N,\\
D_i^{\rho v,n} &= (|\overline{u}| + \overline{c})\frac{\rho v_C^n - \rho v_i^n}{2}, \quad i = W, S,\\
D_i^{E,n} &= (|\overline{u}| + \overline{c})\frac{E_i^n - E_C^n}{2}, \quad i = E, N,\\
D_i^{E,n} &= (|\overline{u}| + \overline{c})\frac{E_C^n - E_i^n}{2}, \quad i = W, S.
\end{aligned} \tag{8}$$

Complex terms in the equation were marked with only one super- and subscript for better understanding, for example, $(\rho u^2 + p)_C^n$ is equal to $\rho_C^n(u_C^n)^2 + p_C^n$. Additionally, in the subscripts E, W, N, and S denote the eastern, western, northern, and southern interfaces of the examined cell.

Finally, in (9), the update scheme for each layer can be seen based on (8),

$$\begin{aligned}
\rho_C^{n+1} &= \rho_C^n - \frac{\Delta t}{\Delta x}(F_E^{\rho,n} - F_W^{\rho,n} + D_E^{\rho,n} - D_W^{\rho,n})\\
&\quad - \frac{\Delta t}{\Delta y}(F_N^{\rho,n} - F_S^{\rho,n} + D_N^{\rho,n} - D_S^{\rho,n}),\\
\rho u_C^{n+1} &= \rho u_C^n - \frac{\Delta t}{\Delta x}(F_E^{\rho u,n} - F_W^{\rho u,n} + D_E^{\rho u,n} - D_W^{\rho u,n})\\
&\quad - \frac{\Delta t}{\Delta y}(F_N^{\rho u,n} - F_S^{\rho u,n} + D_N^{\rho u,n} - D_S^{\rho u,n}),\\
\rho v_C^{n+1} &= \rho v_C^n - \frac{\Delta t}{\Delta x}(F_E^{\rho v,n} - F_W^{\rho v,n} + D_E^{\rho v,n} - D_W^{\rho v,n})\\
&\quad - \frac{\Delta t}{\Delta y}(F_N^{\rho v,n} - F_S^{\rho v,n} + D_N^{\rho v,n} - D_S^{\rho v,n}),\\
E_C^{n+1} &= E_C^n - \frac{\Delta t}{\Delta x}(F_E^{E,n} - F_W^{E,n} + D_E^{E,n} - D_W^{E,n})\\
&\quad - \frac{\Delta t}{\Delta y}(F_N^{E,n} - F_S^{E,n} + D_N^{E,n} - D_S^{E,n}).
\end{aligned} \tag{9}$$

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 (6). This procedure requires the introduction of additional cells with respect to the interface, that is, cell LL (left to cell L) and cell RR (right to cell R) as shown in Figure 1. With these labels, the reconstructed primitive variables are

$$\begin{aligned}
P_L &= P_L + \frac{g_L(\delta P_L, \delta P_C)}{2},\\
P_R &= P_R - \frac{g_R(\delta P_C, \delta P_R)}{2},
\end{aligned} \tag{10}$$

with

$$\begin{aligned}
\delta P_L &= P_L - P_{LL},\\
\delta P_C &= P_R - P_L,\\
\delta P_R &= P_{RR} - P_R.
\end{aligned} \tag{11}$$

while g_L and g_R are the limiter functions. The scheme without limitation 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 (10) we apply the minmod limiter function, also

$$g_L(\delta P_L, \delta P_C) = \begin{cases} \delta P_L, & \text{if } |\delta P_L| < |\delta P_C|,\ \delta P_L \delta P_C > 0,\\ \delta P_C, & \text{if } |\delta P_C| < |\delta P_L|,\ \delta P_L \delta P_C > 0,\\ 0, & \text{if } \delta P_L \delta P_C \le 0. \end{cases} \tag{12}$$

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

The temporal derivative is discretized by the standard two-stage Runge-Kutta method [8]. During the second-order update procedure, the primitive variables (ρ, u, v, and p) are computed from the conservative variables (ρ, ρu, ρv, and E) and extrapolated by using the limiter function. The resulting variables are used to compute the spatial derivatives (9) and time is advanced by half time step according to the second-order Runge-Kutta method. Finally, the whole procedure is repeated to compute the next timestep.

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 (CFL) condition:

$$\Delta t \leq \min_{(i,j)\in([1,M]\times[1,N])} \frac{\min(\Delta x, \Delta y)}{|u_{i,j}| + c_{i,j}}. \quad (13)$$

4. Implementation on Falcon CNN-UM Architecture

The Falcon architecture [9] is an emulated digital implementation of CNN-UM array processor which uses the full signal range model. 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 nonlinear templates can also be used.

The Euler equations were solved by a modified Falcon processor array in which the arithmetic unit has been changed 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 a simple first-order forward Euler temporal discretization, the nonlinear CNN templates acting on the ρu layer can easily be taken from the discretized equations. Equations (14) show templates in which cells of different layers at positions (k, l) are connected to the cell of layer ρu at position (i, j),

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

$$A_2^{\rho u} = \frac{1}{2\Delta x}\begin{bmatrix} 0 & -\rho uv & 0 \\ 0 & 0 & 0 \\ 0 & \rho uv & 0 \end{bmatrix}, \quad (14)$$

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

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

In accordance with (9), we have designed four complex circuits. These are able to update the values of the conservative state vector of a cell in every clock cycle using emulated digital CNN-UM architecture. The arithmetic unit for the computation of the ρu layer is shown in Figure 2. 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 the ratio of Δt and Δx or Δy to be integer power of two because the multiplication with $\Delta t/\Delta x$ and $\Delta t/\Delta y$ can be done by shifts so we can eliminate several multipliers from the hardware and additionally the area requirements will be greatly reduced.

Unfortunately, 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 compute the second-order update scheme described in the previous section directly.

In accordance with the discretized governing equations, we have designed a complex circuit which is able to update the values of the conservative state vector of a cell in every clock cycle using emulated digital CNN-UM architecture. The main building blocks of the proposed unit are shown in Figure 3(a). From the blocks, two identical arithmetic cores can be built according to the two steps of the second-order Runge-Kutta method. In order to get the conservative state values at time level $n + 1$, the two identical units need to be applied successively. The arithmetic core computing ρu value after the first step can be seen in Figure 3(b). Two similar units (F_N and F_E) are required to compute the flux value at the North and South or East and West interfaces while four instances of the third unit (D_E) is required to compute the artificial diffusion term. Inputs of these units are connected to the output of the appropriate limiter units. In order to achieve the highest possible clock speed during the computation, pipelining technique and parallel working hardware units have been used.

5. Fixed-Point Arithmetic Unit

FPGA implementation of the previously described arithmetic unit using floating-point IP cores was reported in [5]. The results show that even computing with 32-bit single precision numbers, the currently available largest FPGAs are required for the implementation. Size of the arithmetic unit is greatly increased by the area requirements of the floating-point adders.

Some previous studies proved the effectiveness of fixed-point numbers during the solution of simple PDEs [10]. In case of simple PDEs, all bits computed during the evaluation of the derivative are kept and rounding is carried out at the last step when the state value is updated. Unfortunately, this method cannot be used in our case because the bit width of the partial results is growing quickly as shown in Figure 4(a). To reduce the bit width inside the arithmetic unit and reduce

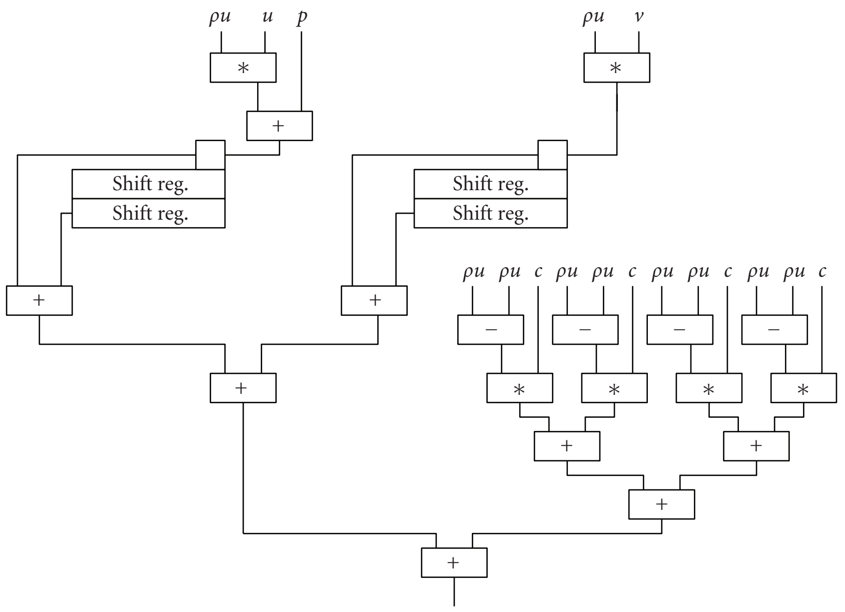
ρu u p
ρu v
Shift reg.
Shift reg.
Shift reg.
Shift reg.
ρu ρu c ρu ρu c ρu ρu c ρu ρu c

FIGURE 2: The proposed arithmetic unit to compute the derivative or ρu layer in the solution using first-order Lax-Friedrichs approximation method.

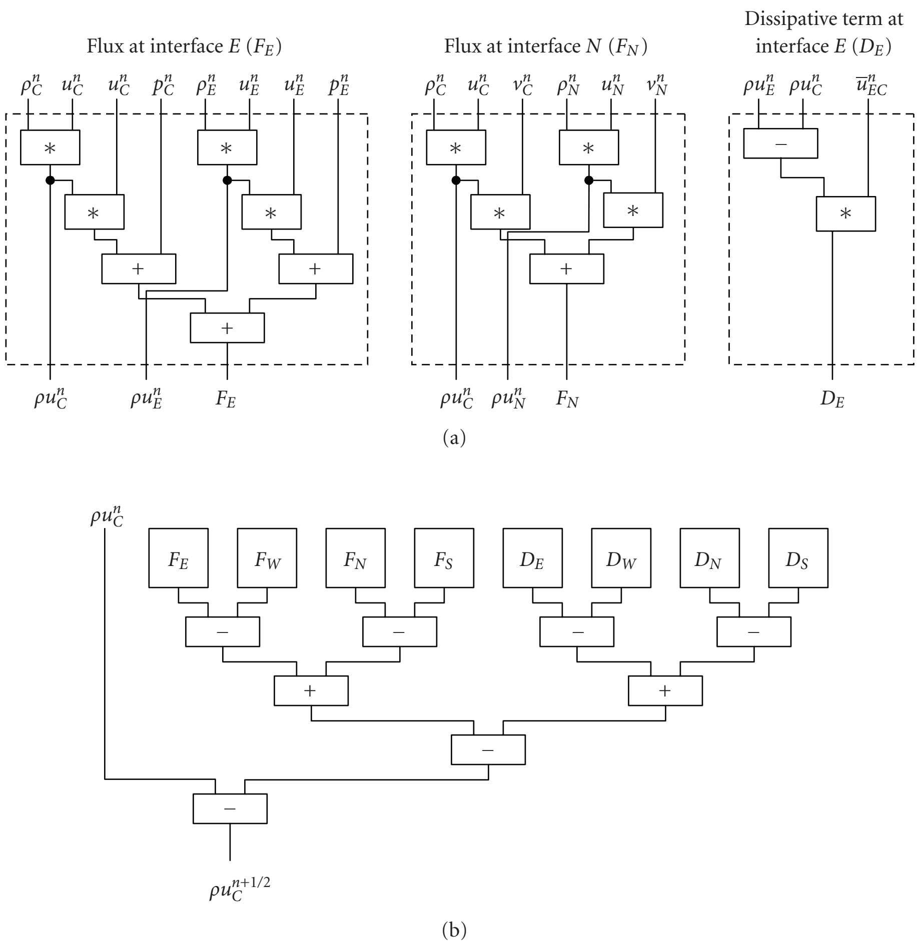
Flux at interface E (F_E)
Flux at interface N (F_N)
Dissipative term at interface E (D_E)
ρ_C^n u_C^n u_C^n p_C^n ρ_E^n u_E^n u_E^n p_E^n
ρ_C^n u_C^n v_C^n ρ_N^n u_N^n v_N^n
ρu_E^n ρu_C^n $\overline{u}_{EC}^n$
ρu_C^n ρu_E^n F_E
ρu_C^n ρu_N^n F_N
D_E
(a)
ρu_C^n
F_E F_W F_N F_S D_E D_W D_N D_S
$\rho u_C^{n+1/2}$
(b)

FIGURE 3: (a) The main building blocks of the proposed arithmetic unit, (b) the whole arithmetic unit built from the main blocks.

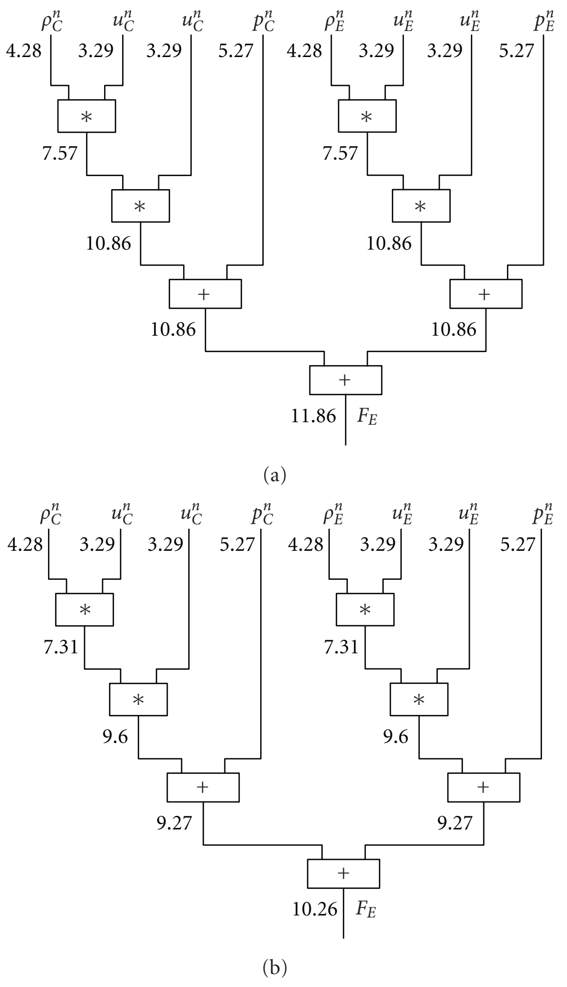

Figure 4: Bit width of the fixed-point arithmetic unit to compute F_E, (a) without optimization, (b) optimized by using interval arithmetic (bit width is denoted by (integer width) . (fractional width)).

area requirements, rounding is required. However, it should be carried out very carefully because important information required to accurately compute the derivative of a state value may be lost during improper rounding.

One possible solution to determine the number of fractional bits required during the computation is to use interval arithmetic [11] and compute the error of the operation along with the result. The basic arithmetic operations computed in interval arithmetic have the following form (m: computer representation of the number, ε: computer representation of the error):

$$m_1 \pm \varepsilon_1 + m_2 \pm \varepsilon_2 = (m_1 + m_2) \pm (\varepsilon_1 + \varepsilon_2), \tag{15a}$$

$$m_1 \pm \varepsilon_1 - m_2 \pm \varepsilon_2 = (m_1 - m_2) \pm (\varepsilon_1 + \varepsilon_2), \tag{15b}$$

$$m_1 \pm \varepsilon_1 \times m_2 \pm \varepsilon_2 = (m_1 \times m_2) \pm (\varepsilon_1 |m_2| + |m_1| \varepsilon_2 + \varepsilon_1\varepsilon_2), \tag{15c}$$

$$m_1 \pm \varepsilon_1 \div m_2 \pm \varepsilon_2 = \left(\frac{m_1}{m_2}\right) \pm \left(\frac{\varepsilon_1 + |m_1/m_2|\varepsilon_2}{|m_2| - \varepsilon_2}\right). \tag{15d}$$

The error of the addition and subtraction is simply the sum of the error of the operands while in the case of multiplication and division, the error of the results also depends on the value of the operands.

In our case, we assume that a priori information is available about the maximum value of the input variables (this is usually true in engineering applications), which can be used to determine the number of integer and fractional bits. We also assume that the least significant bit (LSB) of the input values is erroneous, therefore, ε is set to $2^{-\text{LSB}}$. Error of the additions and subtractions can be easily determined by using (15a)-(15b). However, to determine the error of the multiplication and division, the value of the operands are also required which is not known in advance. Therefore, a worst case analysis of the accuracy of the arithmetic unit should be carried out by computing the minimum and maximum values and the minimum and maximum errors of each partial result. The number of integer bits is computed from the maximal value while the number of fractional bits can be computed form the minimum error value by using the following equations:

$$\begin{aligned} \text{int} &= \lceil \log_2(2\cdot\max) \rceil, \\ \text{frac} &= \lceil -\log_2(\varepsilon_{\min}) \rceil, \end{aligned} \tag{15}$$

where int is the number of integer bits, frac is the number of fractional bits, and max is the computed maximal value of the partial result, while its minimum error is denoted by $\varepsilon_{\min}$. The computed minimum error values represent the theoretically achievable accuracy of the computation. The LSB of the variable (and the smallest representable number $2^{-\text{LSB}}$) should be set to be in the same range as the computed minimal error. If the number of fractional bits is smaller, valuable information is lost. On the other hand, using more fractional bits does not really improve the results. A small part of the arithmetic unit after the optimization (assuming $\rho_{\min} = 0.2$) is shown in Figure 4(b).

Without optimization, the results of the multiplications are stored on 64 and 96 bits and the output of the arithmetic unit (F_E) is 97-bit wide. If the results are used later during multiplications, the bit width is further increased and quickly hits an unpractical size. Using the previously described method, the width of the partial results can be significantly reduced. The width of the multiplications is decreased by 26 bits while the width of the final result is reduced to 36 bits from 97 bits. Area requirements of the arithmetic units are significantly decreased by using these optimizations while the operating frequency is improved.

6. Results and Performance

6.1. Area Requirements. During the implementation of the first- and second-order method, customized precision fixed-point arithmetic cores from Xilinx [12] are used. Implementation and testing of the previously described arithmetic unit can be very time-consuming but using rapid prototyping techniques and high-level hardware description languages such as Handel-C from agility [13] make it possible to

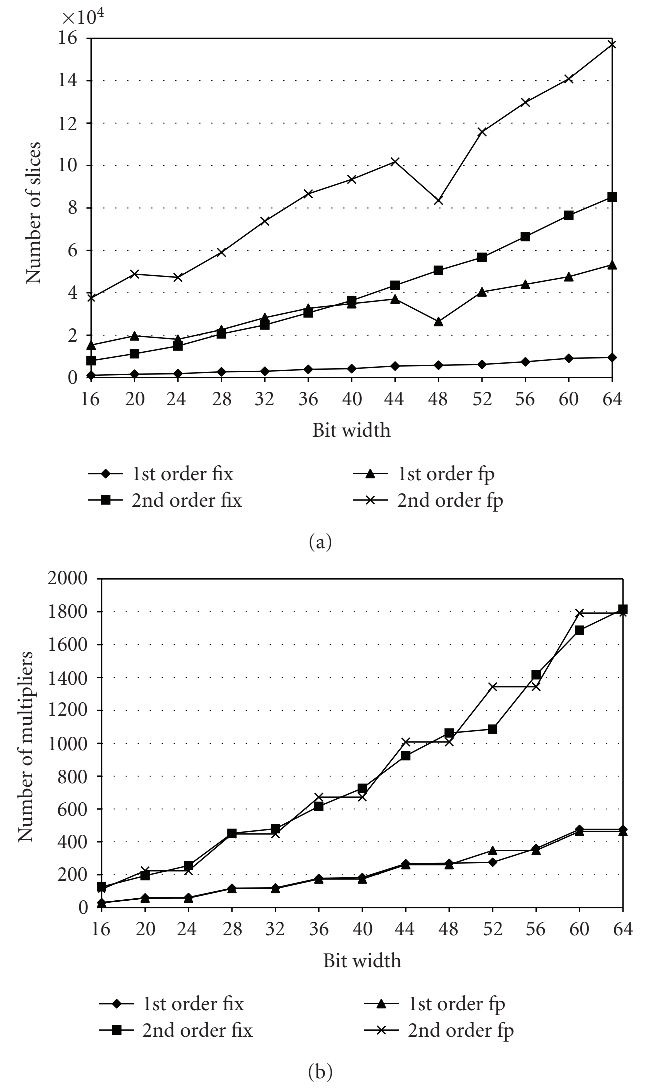

Figure 5: The area requirement of the fixed-point (fix) and floating-point (fp) arithmetic units using different precisions.

develop the optimized arithmetic unit much faster than using conventional VHDL-based approach.

Area requirement of the proposed fixed-point parallel arithmetic units along with the area requirements of the floating-point implementations [5] is shown in Figure 5 (in the following figures, bit width means the sum of the integer and fractional bits of the fixed-point numbers and the width of the mantissa bits in case of the floating-point numbers). Due to the large area requirements of the floating-point arithmetic units, especially the size of the floating-point adders, only the low precision configurations of the fully parallel first-order arithmetic unit can be realized even on the currently available largest FPGAs (Virtex-5 SX240T and LX330T). The fully parallel second-order arithmetic unit cannot be implemented on these devices when floating-point numbers are used. A possible solution could be for this problem if the two steps of the Runge-Kutta method are computed in two steps on the same arithmetic unit. In this case, area requirements can be halved but the computing performance is also halved.

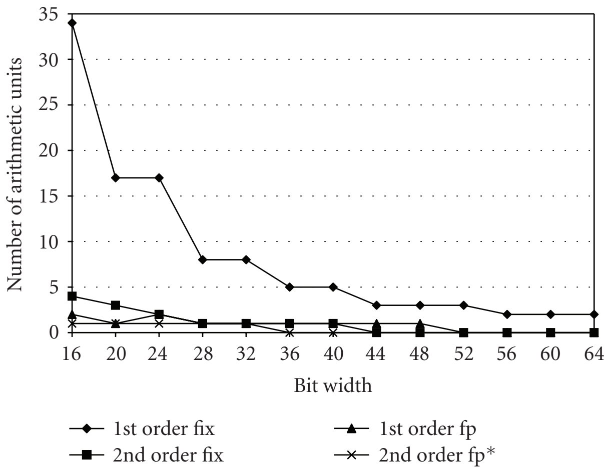

Figure 6: Number of implementable arithmetic units on Virtex-5 XC5VSX240T FPGA (*half arithmetic unit—two clock cycles per cell).

Area requirements of the arithmetic unit can be significantly reduced, compared to the floating-point solution, by using fixed-point numbers and using the optimization method described in the previous section. The required number of dedicated multipliers is about to be equal in the case of fixed- and floating-point arithmetic. However, using fixed-point arithmetic 2–5 times fewer logic elements (slices) are required for the implementation of the first-order arithmetic unit. In the second-order case, the area is decreased more significantly by a factor of 5–15. The number of implementable arithmetic units on the DSP optimized Virtex-5 SX240T FPGA is summarized in Figure 6.

6.2. Test Setup. To show the efficiency of our solution, a complex test case was used, in which a Mach 3 flow over a forward facing step was computed. The simulated region is a two-dimensional cut of a pipe which has closed at the upper and lower boundaries, while the left and right boundaries are open. The direction of the flow is from left to right and the speed of the flow at the left boundary is 3-time the speed of sound constantly. The solution contains shock waves reflected from the closed boundaries. This problem was solved by using the Handel-C simulation of the previously described first- and second-order arithmetic units. In Figures 7 and 8, results of the computation using the derived methods after 0.4 second, 1.2 seconds, and 4 seconds of simulation time with 3.125 milliseconds (1/320 second) time step are shown. In these figures, the dissipative property of the first-order solution can be clearly recognized, while using the second-order method the boundary of the shock waves is sharp on the density distribution map. Because of the applied rectangular, regular grid system a mask was necessary to define the computational domain for the solution. The grid points under the step are masked out and do not take part in the solution resulting in dummy computing cycles. This problem can be eliminated from the system

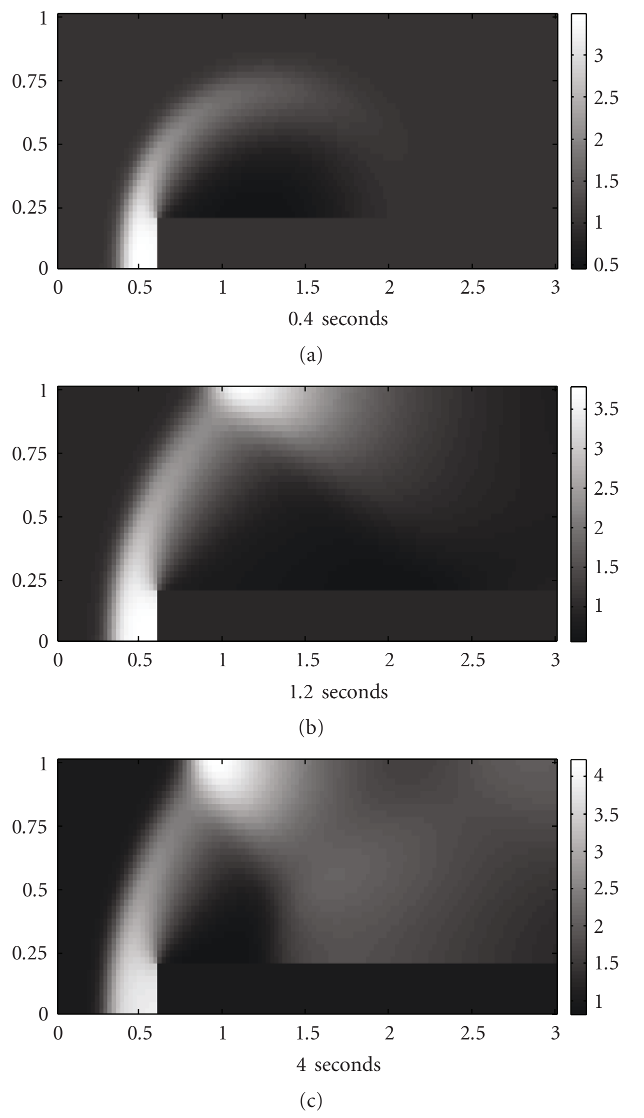

Figure 7: First-order solution of the Mach 3 flow on an 80×240 array after 0.4, 1.2, and 4 seconds of simulation time.

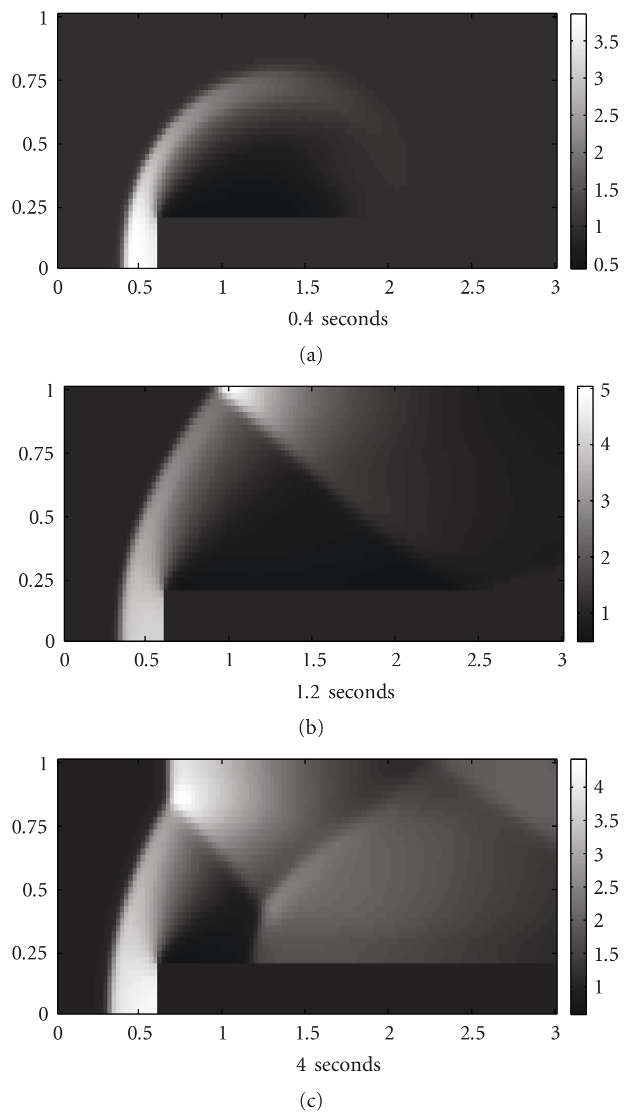

Figure 8: Second-order solution of the Mach 3 flow on an 80×240 array after 0.4, 1.2, and 4 seconds of simulation time.

with the implementation of the multiblock technique when the computational domain is divided into two parts at the forward face of the step.

Reference solution for the previous problem computed by the more accurate residual distribution upwind scheme can be found in [14].

6.3. Performance. Performance of the architecture is determined by the maximum clock frequency and the number of arithmetic units. The huge amount of possible configurations of the arithmetic unit does not enable to carry out postlayout simulations in each case. Therefore, performance data is provided by measuring the maximum performance of the individual functional units. According to the Xilinx data sheets, the floating-point arithmetic cores can run on 350 MHz clock frequency in the case of Virtex-5 FPGAs. Performance of the fixed-point arithmetic cores depends more on the width of the operands, and about 400–550 MHz clock frequency can be achieved. Actual clock frequency of a given configuration can be 0% to 20% smaller according to the utilization of the device and due to changes in placement and routing. Expected performance of the different arithmetic units compared to an Intel Core2Duo microprocessor running on 2 GHz clock frequency is summarized in Figure 9.

The computation of the Mach 3 problem lasts about 2419 seconds on the Core2Duo T7200 microprocessor using first-order approximation while 10591 seconds are required to compute the second-order result. This is equivalent to approximately 1.3 million cell update per second for the first-order method and 0.297 million cell update per second for the second-order approach.

Using 32-bit fixed- and floating-point numbers, all arithmetic units can be implemented on a Virtex-5 SX240T FPGA. On this device, the first-order computation lasts

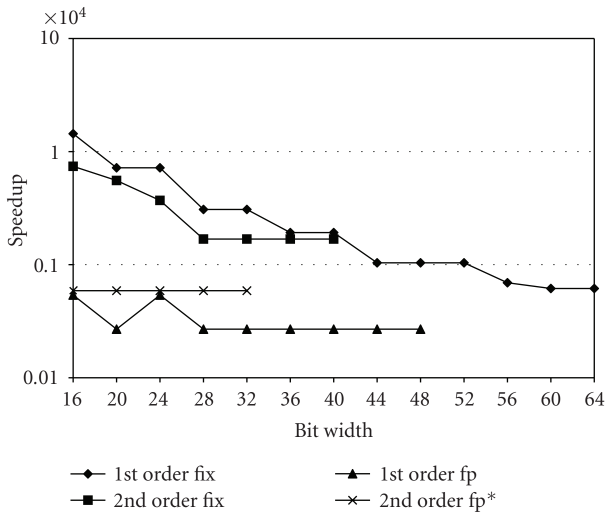

Figure 9: Speedup of the arithmetic unit implemented on Virtex-5 XC5VSX240T FPGA compared to a Core2Duo 2 GHz microprocessor (*half arithmetic unit—two clock cycles per cell).

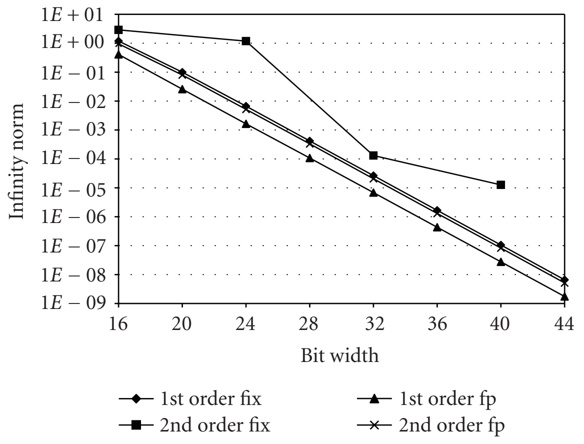

Figure 10: The infinity norm of the solutions.

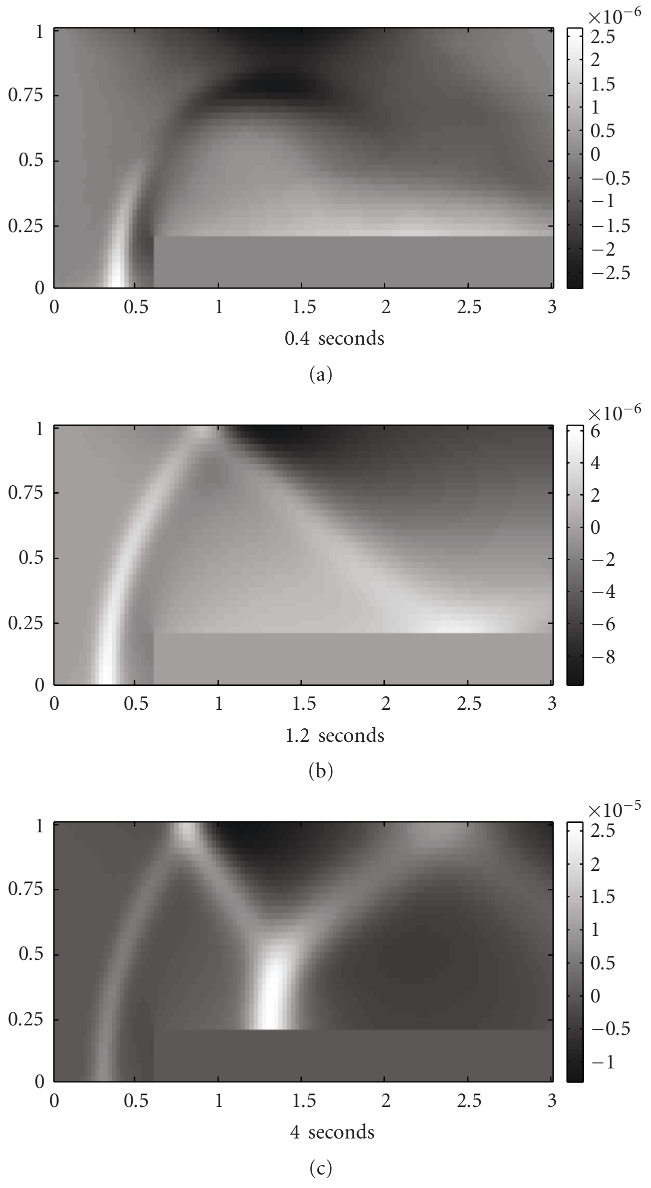

Figure 11: Error distribution of the first-order 32 bit fixed-point solution of the Mach 3 problem after 0.4, 1.2, and 4 seconds of simulation time.

approximately 0.78 second and 8.98 seconds in the fixed- and floating-point cases , respectively, while in the second-order case runtime is increased to 6.29 seconds and 17.97 seconds. The first-order fixed-point arithmetic unit is 11-time faster than its floating-point counterpart and more than 3000-time faster than the Core2Duo microprocessor. In the second-order case, the results are more balanced and the fixed-point arithmetic unit is about 3-time faster than the floating-point arithmetic but its performance is still superior compared to the Core2Duo microprocessor.

Additionally, we tried to use performance data reported in previous works, but fair comparison is hard because different CFD models and discretization schemes are used. Additionally different FPGA architectures are used during the implementations. Smith and Schnore [15] published an FPGA-based CFD solver, but they used 3D model and smaller neighborhood during the computation. Additionally, their architecture was implemented on several FPGAs. In the solution of the Euler equations, they reported 24.6 GFlops sustained performance on four Virtex-II 6000 FPGAs. Sano et al. [16] used 2D systolic array to solve 2D flow problems and reported 11.5 GFlops peak performance on an ALTERA Stratix II FPGA. Sustained performance of our solution using 32-bit fixed-point numbers is 416 and 141 billion fixed-point operations per second in the first- and second-order case, respectively.

6.4. Accuracy of the Solutions. As described in Section 6.1, area requirements of the arithmetic unit can be significantly reduced by decreasing the precision of the state values.

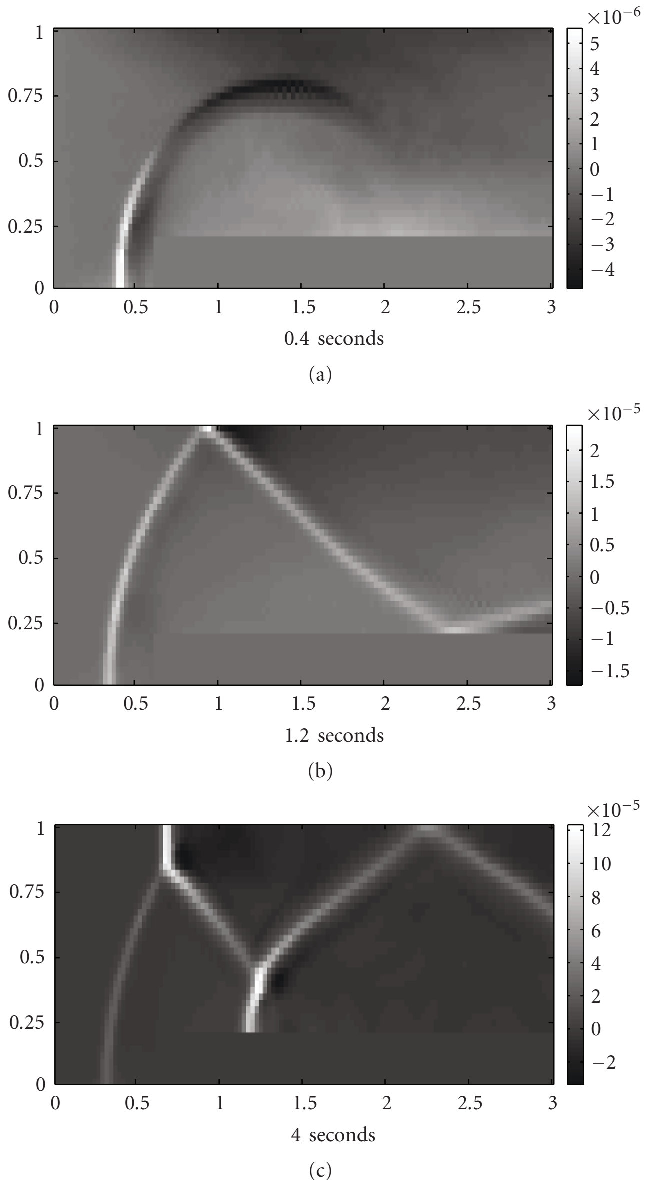

FIGURE 12: Error distribution of the second-order 32 bit fixed-point solution of the Mach 3 problem after 0.4, 1.2, and 4 seconds of simulation time.

However, smaller precision results in less accurate solution. Unfortunately, the exact solution of the Mach 3 problem does not exist, therefore, the fixed- and customized-precision floating-point results were compared to the 64-bit floating-point result. The accuracy of the solutions was measured by computing the infinity norm which is defined as

$$\|e\|_\infty = \max_i \left| u_i^A - u_i^E \right|, \tag{16}$$

where u_i^A is the exact (or in our case the 64-bit) solution, while u_i^E is the numerical approximation using the update scheme with different fixed- and floating-point numbers. The results of the comparison in the case of the Mach 3 problem are shown in Figure 10. Comparing the infinity norm of the solutions to the largest density value ($\rho_{\max}$) in the system, which was in this case about 10, a relative error can be defined as

$$r_{\text{err}} = \frac{\|e\|_\infty}{\rho_{\max}}. \tag{17}$$

The error of the first-order fixed-point solution follows the same trend as the error of the custom width floating-point solution, but the error value in this case is about 4 times higher. The larger error of the solution is balanced by the smaller size and faster operation of the fixed-point arithmetic unit, therefore, it is possible to slightly increase the bit width and compute the results more accurately without loss of the high computing performance.

In the second-order case, the error of the 32-bit fixed-point solution is one-order higher compared to the error of the 32-bit floating-point solution. Increasing the computing precision to 40 bits just slightly increases the accuracy of the solution, and the error compared to the 40-bit floating-point solution is two orders higher. Further investigation is required to find the roots of the different behaviors.

The results, which were calculated applying very low precision (less than 24 bits), are unusable in engineering applications, because the relative error is larger than 10^{-2} in each case. Increasing the precision to 26–36 bits, the relative error of our solution is in the range of 10^{-4}–10^{-6}. These results are accurate enough to use in common engineering applications. Accuracy of the solution can be further increased by using higher precision to represent the state values.

The distribution of the error of the 32-bit fixed-point solutions in the first- and second-order case is presented in Figures 11 and 12, respectively. As it can be seen in these figures in the first-order case the distribution of the error is quite smooth and has a maximum value near the shock waves. In the second-order case, the maximum value of the error is one-order larger and concentrated near the shock waves.

7. Conclusion

The governing equations of the two-dimensional compressible Newtonian flows were solved by using modified emulated digital CNN architecture. The second-order Lax-Friedrichs scheme was used during the solutions. The main advantage of this method over the forward Euler method which is used extensively in the computation of the CNN dynamics is that this approximation is more robust in the case of complex computational geometries and in the presence of shock waves in the solutions.

The arithmetic unit was designed by using both fixed- and floating-point number representations. Interval arithmetic is used to optimally set the precision of the partial results and to reduce the size of the fixed-point arithmetic unit while preserving the accuracy of the solution. The fixed- and floating-point solutions are compared in terms of implementation area, accuracy of the solution, and computing performance.

Implementation area of the arithmetic unit is significantly decreased by the application of fixed-point numbers. The proposed first-order fixed-point arithmetic unit can be implemented on midsized gate arrays. Area requirements of the second-order arithmetic unit are much higher and the currently available largest FPGAs are required for the implementation. The first-order solution using 32 bit fixed-point numbers can be computed 3000 times faster compared to a high-performance microprocessor, while its accuracy is acceptable in engineering applications. The second-order approximation, which models the physical phenomenon more accurately, can be solved 1600 times faster.

In the future, the designed arithmetic unit will be extended to three-dimensional flow problems and nonuniform computational grids could be possible.

References

[1] T. Roska and L.O. Chua, "The CNN universal machine: an analogic array computer," *IEEE Transactions on Circuits and Systems II*, vol. 40, no. 3, pp. 163–173, 1993.

[2] P. Szolgay, G. Vörös, and G. Erőss, "On the applications of the cellular neural network paradigm in mechanical vibrating systems," *IEEE Transactions on Circuits and Systems I*, vol. 40, no. 3, pp. 222–227, 1993.

[3] 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 Transactions on Circuits and Systems I*, vol. 42, no. 10, pp. 807–815, 1995.

[4] Z. Nagy and P. Szolgay, "Numerical solution of a class of PDEs by using emulated digital CNN-UM on FPGAs," in *Proceedings of the 16th European Conference on Circuit Theory and Design (ECCTD '03)*, vol. 2, pp. 181–184, Cracow, Poland, September 2003.

[5] S. Kocsárdi, Z. Nagy, Á. Csík, and P. Szolgay, "Simulation of two-dimensional inviscid, adiabatic, compressible flows on emulated digital CNN-UM," *International Journal of Circuit Theory and Applications*, accepted.

[6] J. D. Anderson Jr., *Computational Fluid Dynamics: The Basics with Applications*, McGraw-Hill, New York, NY, USA, 1995.

[7] T. J. Chung, *Computational Fluid Dynamics*, Cambridge University Press, Cambridge, UK, 2002.

[8] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes: The Art of Scientific Computing*, Cambridge University Press, Cambridge, UK, 2007.

[9] Z. Nagy and P. Szolgay, "Configurable multilayer CNN-UM emulator on FPGA," *IEEE Transactions on Circuits and Systems I*, vol. 50, no. 6, pp. 774–778, 2003.

[10] 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.

[11] O. Aberth, *Introduction to Precise Numerical Methods*, Elsevier, Amsterdam, The Netherlands, 2007.

[12] Xilinx products, 2008, http://www.xilinx.com.

[13] Agility design solutions, 2008, http://www.agilityds.com.

[14] Á. Csík and H. Deconinck, "Space-time residual distribution schemes for hyperbolic conservation laws on unstructured linear finite elements," *International Journal for Numerical Methods in Fluids*, vol. 40, no. 3-4, pp. 573–581, 2002.

[15] W. D. Smith and A. R. Schnore, "Towards an RCC-based accelerator for computational dluid dynamics applications," *Journal of Supercomputing*, vol. 30, no. 3, pp. 239–261, 2004.

[16] K. Sano, T. Iizuka, and S. Yamamoto, "Systolic architecture for computational fluid dynamics on FPGAs," in *Proceedings of the 15th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM '07)*, pp. 107–116, IEEE Computer Society, Los Alamitos, Calif, USA, April 2007.