International Journal of Networking and Computing – www.ijnc.org
ISSN 2185-2839 (print) ISSN 2185-2847 (online)
Volume 3, Number 2, pages 289–306, July 2013

Adaptive Flux Calculation Scheme in Advection Term Computation Using Partial Reconfiguration

Mohamad Sofian Abu Talip, Takayuki Akamine, Mao Hatto, Hideharu Amano
Graduate School of Science and Technology
Keio University
Yokohama-shi 223-8522 Japan

Yasunori Osana
Department of Electrical and Electronics Engineering
University of the Ryukyus
Nishihara, Okinawa-ken 903-0213 Japan

Naoyuki Fujita
Aerospace Research and Development Directorate
Japan Aerospace Exploration Agency
Chofu-shi, Tokyo 182-8522 Japan

Received: February 20, 2013
Revised: May 18, 2013
Revised: June 24, 2013
Accepted: June 26, 2013
Communicated by Yasuaki Ito

Abstract

In aerospace industry, computational fluid dynamics (CFD) is used as a common design tool. Fast Aerodynamics Routines (FaSTAR) is one of the most recent CFD software package, convenient for users with various solvers and automatic generation of grid data. The problem of FaSTAR is hard to be executed in parallel machines because of its irregular and unpredictable data structure. Exploiting reconfigurable hardware with their advantages to make up for the inadequacy of the existing high performance computers had gradually become the solutions and trends. However, a single FPGA is not enough for the FaSTAR package because the whole module is very large. Instead of using a large number of chips, partially reconfigurable hardware available in recent FPGAs is explored for this application. Advection term computation module in FaSTAR is chosen as a target subroutine. We proposed a reconfigurable flux calculation scheme using partial reconfiguration technique to save hardware resources to fit in a single FPGA. We developed flux computational module and three flux calculation schemes are implemented as reconfigurable modules. This implementation has advantages of up to 42% resource saving and enhancing the configuration speed by 6.28 times. Performance evaluation also shows that 2.65 times acceleration is achieved compared to Intel Core 2 Duo at 2.4GHz.

Keywords: Computational Fluid Dynamics (CFD), Field-Programmable Gate Array (FPGA), Scientific Computation, Reconfigurable Hardware, Partial Reconfiguration

1 Introduction

CFD (*Computational Fluid Dynamics*) has been widely utilized in the design and optimization of fluid flow applications. In aerospace industry, CFD is a cost-effective design tool for aircraft components. It presents methods to solve and analyze problems of the physical phenomena of fluids involving fluid flow on discrete space and time. Ground test facilities do not exist in all flight regimes covered by such hypersonic flight. No wind tunnels that can simultaneously simulate the higher Mach numbers and high flow-field temperatures to be encountered by trans-atmospheric vehicles. Hence, CFD has become the major player in the design of such vehicles. In addition, compressible flow simulations are of vital importance to the aerospace engineering community, which will always seek more accuracy and higher resolution creating a demand for faster codes and making the use of high performance computing strategies invaluable.

A typical simulation platform in the aeronautics industry consists of a CFD specific software application, normally written in a high-level language. FaSTAR (*Fast Aerodynamics Routines*), a compressible flow analysis code developed by JAXA (*Japan Aerospace Exploration Agency*) is one of such CFD software packages [6]. FaSTAR adopts unstructured mesh as grid data in the simulation. Although it is a convenient tool for aerodynamics analysis, it sometimes takes several days or weeks when an analytical area grows large. This is mainly caused by low parallel processing efficiency accompanied with pointer links and a complicated memory access pattern. In FaSTAR, the increasing demands for accuracy and simulation capabilities produce an exponential growth of the required computational resources.

Recently, reconfigurable systems using FPGAs have been utilized for acceleration of specific applications including bio-informatics and financial problems [8, 9]. Even though the early reconfigurable systems did not focus on large-scale numerical scientific application, the use of FPGAs for such areas has been growing remarkably because of the rapid performance improvement of modern FPGAs with a large number of configurable logic blocks, memory blocks and embedded multipliers. However, although some research works using FPGAs achieved significant speed-up ratio to the software, targets were simple programs rather than practical software packages.

The goal of our research is to improve the performance of FaSTAR by using FPGAs as a reconfigurable platform. Previously, we have proposed out-of-order mechanisms to cope with unstructured mesh for efficient execution of FaSTAR [1]. Another problem of FaSTAR is its versatile functions using a lot of solvers. For implementation of the whole package, a lot of expensive FPGAs each of which has a certain level of size are required. Using partial reconfiguration is a hopeful strategy for reducing the total cost. This work is the first trial of applying the partial reconfiguration technique to a practical scientific computation package. The total required hardware becomes large if all functions of the package are implemented in a single design. We can avoid it by providing a set of subset designs with only required functions in advance. However, the number of designs to be prepared in advance becomes large for a complicated application package, since there are several parts each of which has selectable functions and required numbers of subset designs are their products. By introducing the partial reconfiguration, we can quickly prepare the appropriate design for the user. This trial is the first step of using partial reconfiguration technique for a large practical scientific package, and it has a possibility to extend the application field of the partial reconfiguration for this field. Here, as a target function, we select the advection term computation, which is a time consuming large function in FaSTAR.

The rest of this paper is organized as follows. Section 2 discusses related work to this study. Section 3 is an explanation regards to FaSTAR and target subroutines. Section 4 introduces the theory of implemented flux calculation schemes. Section 5 describes about the implementation of this work. Then, following to Section 6 for performance evaluation, Section 7 discusses about future work, and Section 8 summarizes this work with conclusion.

2 Related Work

In fluid dynamics simulation, execution time is one of the largest concerns. There are various studies using FPGAs reported so far to examine the issue. Andres *et al.* and Sano *et al.* reported

Table 1: Example of available solutions used in FaSTAR.

Solutions	**Available Solvers**
Governing Equation	Euler Equation Navier-Stokes Equation Reynolds Averaged Navier-Stokes Equation
Turbulent Flow Model	Spalart-Allmaras Model (SA) Menter k-ω Shear Stress Transport Model (SST) Automatic Wall Processing Capability
Inviscid Flux Calculation Scheme	Roe Scheme HLLE Scheme HLLEW Scheme
Flux Evaluation	Least-Square Law Green-Gauss Law
Flux Limiter Function	Hishida Limiter Venkatakrishnan Limiter Bart-Jespersen Limiter minmod Limiter
Time Integration	LU-SGS(regular/unsteady, local/global time stepping)
Convergence Acceleration	Multigrid (FAS:Full Approximation Storage) Krylov Law (GMRES)

the result for FPGA accelerations [2, 15]. However, their implementations are not for practical software packages. Another result is reported for implementation of FPGA flow solver based on the systolic architecture for CFD [14]. This work proposed a systolic algorithm for the fractional-step method employing the central difference schemes. Although satisfactory performance is achieved, their implementation focuses on systolic algorithms.

Partial reconfiguration technology has attracted many researchers for filling the gap between hardware and software for algorithm implementation. Recently, in computer security applications, partial reconfiguration is applied in AES *(Advanced Encryption Standard)* algorithm implementation [7]. It also had been used to accelerate video processing in driver assistance system [3]. Also, a few researches had been done on an aerospace applications using partial reconfiguration method. LaMeres *et al.* [10] designed and prototyped the computing architecture which dynamically reconfigures the system depending on the environment. Another work has proposed the SoCWire architecture verification, test and results on network on chip for safe dynamic partial reconfiguration in spaces applications [12].

Our previous trial [16], was the first example of using partial reconfiguration in CFD implementation. However, it was applied to small limiter functions in a subroutine called MUSCL used in UPACS [18]. Although the total hardware is reduced, the effect is limited since it only occupies a small portion of the target FPGA.

3 FaSTAR

FaSTAR is a CFD software package developed by JAXA to simulate compressible flow using unstructured grids [6]. FaSTAR consists of many solvers with multiple solutions. Its source code is written in Fortran 90 with message passing interface (MPI). By choosing certain solvers, users can select various solutions supported by the application and run simulation in parallel with their systems without specific software tunings. Users just are requested to prepare parameter file and grid data file before the simulation. Table 1 shows example solvers available in FaSTAR. By combining these solvers, a user can simulate with desired solutions. When the partial reconfiguration is applied to selectable solutions, first, the profiling is required. Then, the part in which multiple schemes are available is picked up from the subroutines, which take a long computation time.

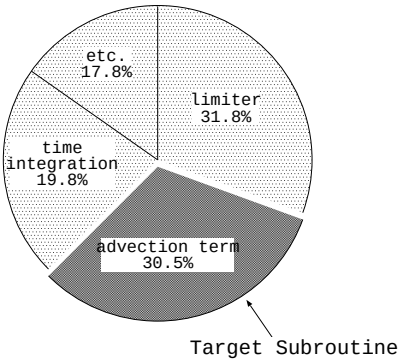

Figure 1: FaSTAR profiling result.

3.1 Target Subroutine

As a first step, we profiled the execution time of FaSTAR to find out which routines consume the highest percentage. Compiler used is Intel Fortran Compiler 10.1 on Intel Core 2 Duo processor at 2.66 GHz with Linux Kernel 2.6. The profiling results are shown in Figure 1.

Code for a single core without MPI was compiled. The result indicates that more than 60% of the total execution time is occupied by two calculations part: `limiter` and `advection term`. FaSTAR limiter part is difficult for implementation in reconfigurable hardware because of its complicated iteration. Here, we selected the advection term, since it occupies a large part of the total computation time, and has a selectable function whose hardware requirement is relatively large.

Advection term is consisting of three subroutines:

- `pre-processing`,
- `flux calculation`, and
- `surface integral of flux`.

In `pre-processing`, data of a specific cell number are prepared. Using these data, flux is obtained by applying to scheme equations in `flux calculation` part. Here, we try to apply partial reconfiguration to make reconfigurable scheme selection as a focus of this paper. The finite volume method for discretization of the space is processed in `surface integral of flux`. We have reported our study for this part in [1].

4 Flux Calculation Scheme

In flux calculation subroutines, there are three schemes available for selection: Roe's scheme, HLLE scheme and HLLEW scheme [13, 4, 11]. We describe how to compute the inviscid flux using Riemann solver approximation. As shown in Figure 2, conserved quantities, Q_{na} and Q_{nb} in cell A and cell B were used to determine flux, F_n. In this case, with the respective to cell A, the normal vector, d_s has same right direction with flux. On the other hand, with respect to cell B, the opposite direction is positive.

4.1 Roe's Scheme

The method proposed by Roe [13] is based on Euler's equation into windward of the linearization. The numerical flux function can be written as follows:

$$F_n = \frac{1}{2}[f(Q_{na}) + f(Q_{nb}) - |A|_{ave}(Q_{nb} - Q_{na})] \quad (1)$$

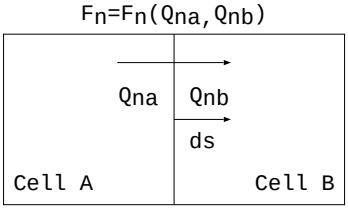

Figure 2: Flux in definition of cell surface boundary.

In this example, F_n is evaluated when it is viewed from the cell A. If it is viewed from the cell B, negative sign is added to become -F_n. In detail we may write:

$$f(Q_{na}) = \begin{pmatrix} \rho_a u_{na} \\ \rho_a u_{na}^2 + p_a \\ \rho_a u_{na} u_{t1a} \\ \rho_a u_{na} u_{t2a} \\ \rho_a u_{na} H_a \end{pmatrix} = \rho_a u_{na} \begin{pmatrix} 1 \\ u_{na} \\ u_{t1a} \\ u_{t2a} \\ H_a \end{pmatrix} + \begin{pmatrix} 0 \\ p_a \\ 0 \\ 0 \\ 0 \end{pmatrix} \tag{2}$$

$$Q_{na} = \begin{pmatrix} \rho_a \\ \rho_a u_{na} \\ \rho_a u_{t1a} \\ \rho_a u_{t2a} \\ e_a \end{pmatrix} \tag{3}$$

That is, the way to obtained $f(Q_{nb})$ and Q_{nb} is the same. In addition, A is the Jacobian matrix for flux,

$$|A|_{ave} = R_{ave} |\Lambda|_{ave} R_{ave}^{-1} \tag{4}$$

R is the matrix with right eigenvector, Λ is matrix with eigenvalue and R^{-1} is matrix with left eigenvector. The matrix with subscript ave, is composed by variables which obtained by Roe average. The Roe average can be expressed as follows:

$$\begin{aligned} \rho_{ave} &= \sqrt{\rho_a \rho_b} \\ u_{ave} &= \frac{\sqrt{\rho_a} u_a + \sqrt{\rho_b} u_b}{\sqrt{\rho_a} + \sqrt{\rho_b}} \\ H_{ave} &= \frac{\sqrt{\rho_a} H_a + \sqrt{\rho_b} H_b}{\sqrt{\rho_a} + \sqrt{\rho_b}} \\ c_{ave} &= \sqrt{(\gamma - 1)(H_{ave} - \frac{1}{2} u_{ave}^2)} \end{aligned} \tag{5}$$

Here, c is the speed of sound, H is the total enthalpy per unit mass, and γ is specific weight representing the force exerted by gravity on a unit volume. Therefore, $\gamma = \rho * g$. Then, we can write the matrix in details as follows:

$$R = \begin{bmatrix} 1 & 0 & 0 & \frac{\rho}{2c} & \frac{\rho}{2c} \\ u & 0 & 0 & \frac{\rho(u+c)}{2c} & \frac{\rho(u-c)}{2c} \\ v & 0 & -\rho & \frac{\rho v}{2c} & \frac{\rho v}{2c} \\ w & \rho & 0 & \frac{\rho w}{2c} & \frac{\rho w}{2c} \\ \frac{u^2+v^2+w^2}{2} & \rho w & -\rho v & \frac{\rho}{2c}(H + uc) & \frac{\rho}{2c}(H - uc) \end{bmatrix} \tag{6}$$

$$|\Lambda| = \begin{bmatrix} |u| & 0 & 0 & 0 & 0 \\ 0 & |u| & 0 & 0 & 0 \\ 0 & 0 & |u| & 0 & 0 \\ 0 & 0 & 0 & |u+c| & 0 \\ 0 & 0 & 0 & 0 & |u-c| \end{bmatrix} \tag{7}$$

Here, if *ave* subscript can be omitted, $u = u_n$, $v = u_{t1}$ and $w = u_{t2}$.

4.2 HLLE Scheme

Einfeldt [4] discussed an adapted version of the HLL scheme [5], called HLLE (*Harten-Lax-van Leer-Einfeldt*) scheme, which can be considered as a modification of Roe's scheme. This scheme is a stable procedure that is solved by approximation of two characterized waves, which are valid for the flow with a strong expansion. However, there is a disadvantage that the numerical viscosity is large. Total flux can be calculated using the following equations:

$$F_n = \frac{b^+ f(Q_{na}) - b^- f(Q_{nb})}{b^+ - b^-} + \frac{b^+ b^-}{b^+ - b^-}(Q_{nb} - Q_{na}) \tag{8}$$

where,

$$\begin{aligned} b^+ &= max(u_{ave} + c_{ave}, u_{nb} + c_b, 0) \\ b^- &= min(u_{ave} - c_{ave}, u_{na} - c_a, 0) \end{aligned} \tag{9}$$

In this example, we evaluated F_n when it is viewed from the cell A. Again, if it is viewed from cell B, it will become -F_n. The average value b, is calculated using Roe average in Eq. 5.

4.3 HLLEW Scheme

Obayashi and Wada proposed a new, modified HLLE scheme that satisfies the positively conservative condition called HLLEW (*Harten-Lax-van Leer-Einfeldt-Wada*) scheme [11]. The numerical flux can be calculated using the following equations:

$$F_n = \frac{1}{2}\left[f(Q_{na}) \begin{pmatrix} 1 \\ u_{na} \\ u_{t1a} \\ u_{t2a} \\ H_a \end{pmatrix} + f(Q_{nb}) \begin{pmatrix} 1 \\ u_{nb} \\ u_{t1b} \\ u_{t2b} \\ H_b \end{pmatrix} + \begin{pmatrix} 0 \\ p_a + p_b + \delta_2 \\ \delta_3 \\ 0 \\ 0 \end{pmatrix} \right] \tag{10}$$

where,

$$\begin{aligned} \delta_2 &= -(\lambda^+ \rho_{ave} \Delta u + \lambda^- \frac{\Delta p}{c_{ave}}) \\ \delta_3 &= -(\lambda_1 \Delta p + u_{ave} \delta_2) \\ \lambda^+ &= \frac{\lambda_2 + \lambda_3}{2} - \lambda_1 \\ \lambda^- &= \frac{\lambda_2 - \lambda_3}{2} \end{aligned} \tag{11}$$

Coefficients, δ_2 and δ_3 are obtained using the Roe average in Eq. 5. Same as Roe and HLLE schemes, we evaluated F_n when it is viewed from cell A.

Table 2: Implementation environments.

Name	Tools
HDL	Verilog HDL
FPGA	Virtex-6 XC6VLX240T FF1156
Synthesis	ISE 12.4 XST
Simulation	ISim M.81d Simulator
PR Flow	PlanAhead 12.4
Programming	iMPACT M.81d
Floating-point Unit	CORE Generator

5 Design and Implementation on an FPGA

Xilinx Virtex-6 FPGA (XC6VLX240T-1FF1156) is chosen as a target device, which supports partial reconfiguration. In a large software package, a subroutine itself is not always appropriate as a target of partial reconfiguration. For example, all three schemes treated here include Roe average calculation and so it must be implemented as a static module. Before the design, the target is well re-structured so that the static module and partial reconfiguration modules are appropriately separated.

Overview of the system is shown in Figure 3. At the beginning, the system will initialize and updates mesh size for each grid data and face index. Then, it will calculate Roe average value in Roe average module since this value is needed for all schemes. Inputs for the Roe average module are density, ρ, velocity, u and total enthalpy per unit mass, H. The result of Roe average module is directly inputted to the flux calculation module. Meanwhile, grid data file of the cell number and index are also inputted to the flux calculation module.

After that, in the reconfigurable module, users can choose which scheme they want to use. Partial reconfigurability of the FPGA and intractability of the bitstream meet the requirements of this system. Each scheme module has the same inputs and outputs as shown in Figure 4, thus, it can be specified in the HDL description as the functional modules with the reconfigurable partition attribute in the description of the top module. Multiple instances corresponding to the schemes are defined for such a single functional module. Software tools as NGDBuild, MAP and PAR detect the reconfigurable partition attribute on the instance, and process it correctly. RAM is allocated in each scheme to store variables values during calculation. It is built with Block RAMs, in which data are stored temporarily.

Programmable input/output (I/O) module is designed to control the access to memory. A result of flux calculation module is stored in memory. We implement simple dual-port RAM for each adjacent cell, cell A and cell B, shows as memory A and memory B. Here, Block RAM used is 36 Kb block, RAMB36E, which is configured in a simple dual-port RAM mode. Read/Write data width is set to 64 bit. Read/Write process is performed in parallel with the flux calculation module. Summation of all cell flux values gives the total flux.

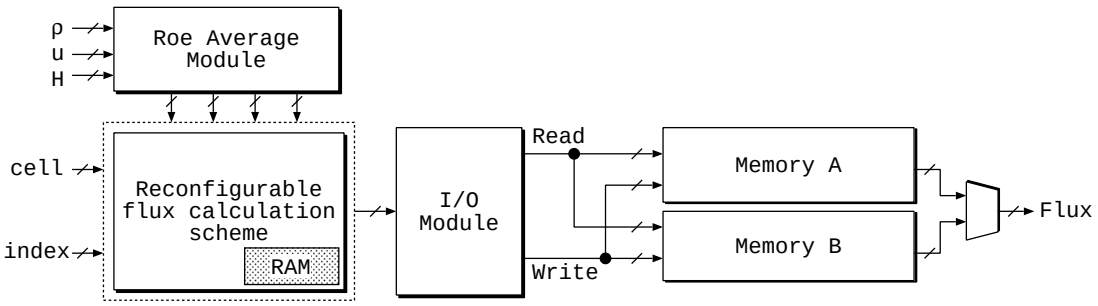

Figure 3: System overview.

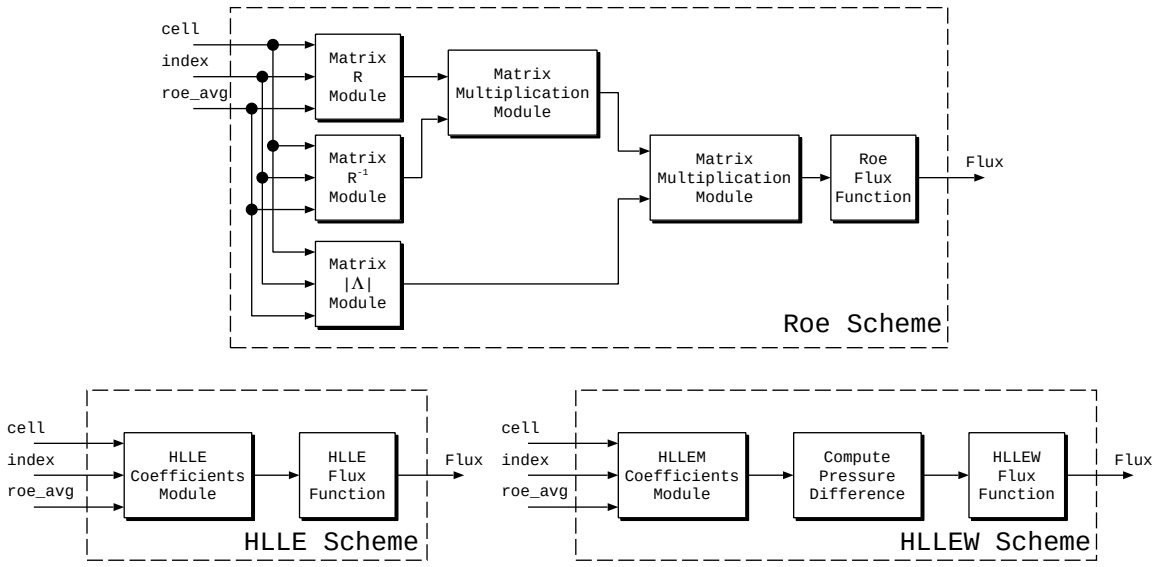

Figure 4: Implemented flux calculation schemes.

All modules are described using Verilog HDL and simulated with Xilinx ISim simulator. The modules are synthesized, and used resources are measured with Xilinx ISE 12.4. Floorplanning, constraint entry and design rule checks (DRCs) are all accessed through the PlanAhead 12.4 software environment, which supports partial reconfiguration flow. All modules are implemented using IEEE754 standard 64-bit double precision floating-point arithmetic. Here, the floating-point computational module is based on the Xilinx Floating-Point Operator v5.0 incorporated into Xilinx ISE 12.4 software. The Floating-Point Operator v5.0 is an IP core for handling floating-point operations, and it is configurable by the user specifications. Xilinx CORE Generator is used to provide the core for floating-point arithmetic units. So as to generate high performance computation unit, the level of DSP48E usage is set to the maximum to get the desired output. In order to demonstrate that our system works on a real FPGA, Xilinx ML605 board is used with 200 MHz operating frequency. Finally, for programming the FPGA, Xilinx iMPACT software is used. Summary for the implementation environments is shown in Table 2.

At one time, only one scheme is used and employed in the FPGA. The top, static and reconfigurable modules are consisting of many arithmetic functions. The parameters used for each computing unit are shown in Table 3. For high performance computation, adder and subtractor are set to 14 clock cycles per operation using high-speed mode. In addition, the multiplier takes 16 clock cycles with 11 DSP48E usages. The latency of square root and divider is set to be 57 clock cycles. Although it is possible to decrease the square root and divider pipeline latency, it will severely degrade the clock frequency. Latency for the comparator unit is the fastest; it only requires one clock cycle.

We used bottom-up synthesis technique to synthesize the design by modules. This synthesis technique requires that a separate netlist is written for reconfigurable partition ensuring that each

Table 3: Data of used computing units.

Units	Latency	Registers	LUTs	DSP48E	bit width	data type
Adder	14	947	797	3	64	floating-point
Subtractor	14	947	798	3	64	floating-point
Multiplier	16	483	362	11	64	floating-point
Divider	57	5973	3261	0	64	floating-point
Square Root	57	3283	1902	0	64	floating-point
Comparator	1	0	128	0	64	floating-point

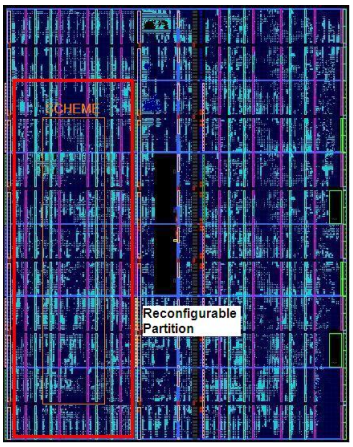

Figure 5: Floorplan of reconfigurable partition with HLLE scheme.

portion of the design is synthesized independently. Top and static module are synthesized with black box for the reconfigurable partition. In this case, flux calculation scheme module is defined as a black box in a top module synthesis. Roe, HLLE and HLLEW scheme modules are synthesized beforehand to provide the required netlist.

Next crucial step is to perform manual floorplanning for reconfigurable partition, which requires knowledge of the physical architecture of FPGA and understanding of how to floorplan for optimal performance and area. The challenge is how to create and pack large flux calculation schemes into a single partition. The partition boundary is defined so that the inserted proxy logic and the extra wiring cost may not degrade the total performance. Although irregular shaped partition such as T or L shapes is allowed, placement and routing in such regions sometimes degrade the performance because of the shortage of the routing resources and long wires. Therefore, we chose a certain size rectangular shape for flux calculation scheme as shown in Figure 5, which is adequate for all 3 schemes. Figure 5 shows a floorplan of the system with HLLE scheme is deployed. This is important for all reconfigurable modules to have enough resources to fit in the partition when the bitstream is loaded. After that, timing constraint entry and design rule checks (DRC) are performed.

5.1 Roe Average Module

As shown in Section 4, all schemes are needed of Roe average values before the flux computation is processed. Therefore, Roe average module is decided as a static module since it will be used in all cases. In this module, managing parallelism is an important issue. The FaSTAR source code for this calculation in Fortran 90 is written as follows:

```
RAT = SQRT(RRHT/RLFT)
RATI = 1.0/(RAT + 1)
RAV = RAT*RLFT
UAV = (RAT*URHT + ULFT) * RATI
VAV = (RAT*VRHT + VLFT) * RATI
WAV = (RAT*WRHT + WLFT) * RATI
HAV = (RAT*HRHT + HLFT) * RATI
QA2 = UAV*UAV + VAV*VAV + WAV*WAV
CA2 = GM1*(HAV - 0.5d0*QA2)
CAV = SQRT(CA2)
```

The code is executed sequentially from top to the bottom. The advantage of sequential operations is that they efficiently use the resources, whereas parallelism can be used to reduce the time to completion and get the Roe average values at the expense of additional hardware resources.

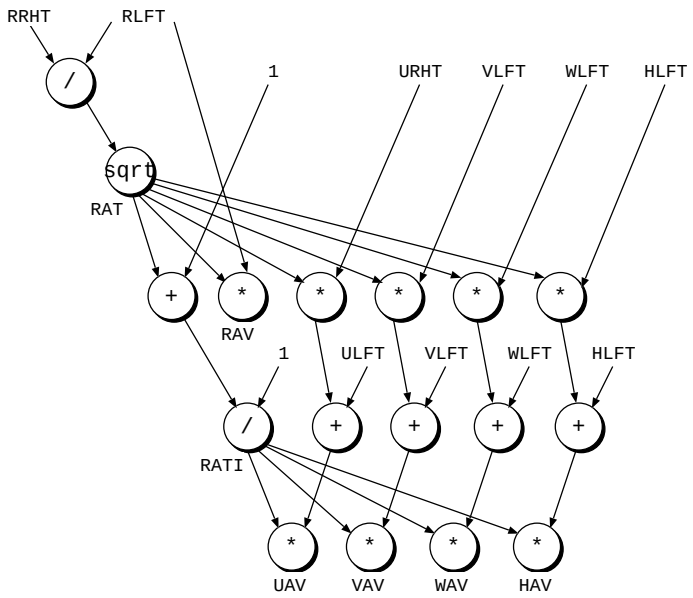

Figure 6: Scheduled data flow graph for Roe average module.

The idea behind control parallelism is that the statements used to compute u_{ave} (`UAV`), v_{ave} (`VAV`), w_{ave} (`WAV`) and H_{ave} (`HAV`) can be performed simultaneously while still producing the correct answer. A scheduled data flow graph as shown in Figure 6 represents a data dependencies between operations. After `SQRT` output to get the value of `RAT`, all multiplications operations are executed in parallel. At this stage, ρ_{ave} and `RAV` are obtained. Then, after `RATI` value is obtained, all `UAV`, `VAV`, `WAV` and `HAV` can be also computed in parallel. However, `CAV` cannot begin until `QA2` and `CA2` finish. At the beginning to compute `RAT` will take a large number of clock cycles, since square root and divider computing units require a large number of clock cycles each. Therefore, we implement pipeline datapath to address this issue. Implemented pipeline datapath for Roe average module is shown in Figure 7. Registers are inserted in the dot line to create a single stage pipeline.

5.2 Roe Scheme Module

Roe scheme module is implemented in reconfigurable partition as a reconfigurable module. This module is designed and synthesized separately from the top module. Roe calculation scheme involves 5 steps to get the results:

1. Compute matrix R.
2. Compute matrix R^{-1}.
3. Compute matrix $|\Lambda|$.
4. Compute Jacobian matrix $|A|$.
5. Compute Roe's numerical flux.

The main arithmetic operation of this module is a 5×5 matrix multiplication as shown in Eq. 4. In general, the standard matrix multiplication $C = A \times B$ is defined as follows:

$$C_{i,j} = \sum_{k=0}^{N-1} A_{i,k} \times B_{k,j}, (0 \leq i \leq M, 0 \leq j \leq R)$$

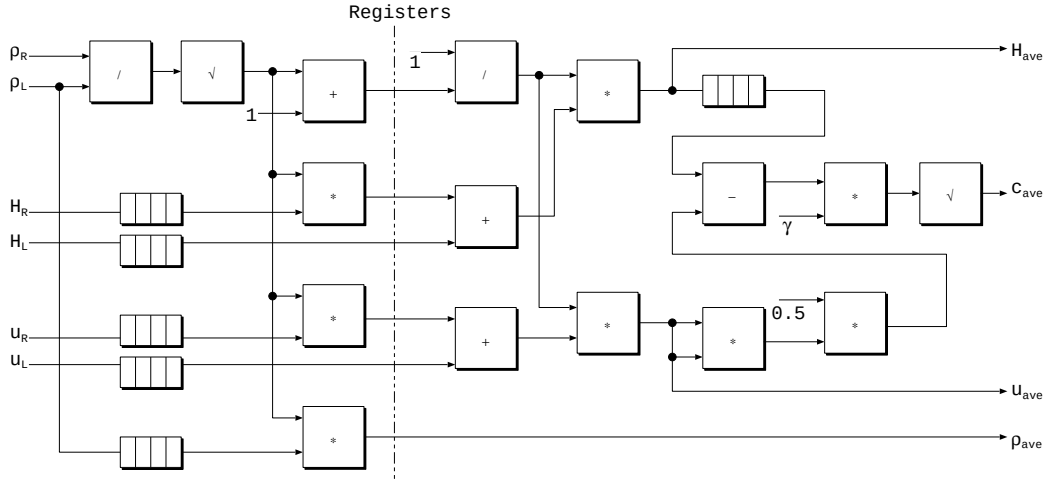

Figure 7: Pipeline datapath for Roe average module.

Where A, B and C are $M \times N, N \times R$, and $M \times R$ matrices, respectively. Since all matrices are 5×5, the pseudo code for matrix multiplication is shown in Algorithm 1. However, it requires two times matrix multiplication to obtain the Jacobian matrix, which utilizes a lot of resources.

However, computation of each matrices R, R^{-1} and $|\Lambda|$ are done in parallel. Then, matrix R is multiplied with matrix R^{-1}. Result of this matrices is multiplied with matrix $|\Lambda|$ to obtain Jacobian matrix. Finally, Jacobian matrix is used to compute the numerical flux.

We implemented a MAC *(Multiplication and Accumulation)* unit structure that couples the multiplication and the accumulation closely as shown in Figure 8. The multiplier receives the elements of A and B in a data driven manner. That means whenever both of the data are available, they will enter the pipeline. After the multiplication, the result is stored in FIFO and loaded address is generated. The next multiplication result will be added with the prefetch data from FIFO, and accumulated results are stored in temporary memory. This operation strategy is repeated continuously until calculation finishes.

5.3 HLLE Scheme Module

HLLE scheme module is rather straight forward compared to Roe scheme module. This module is also defined as a reconfigurable module same as Roe scheme module. Therefore, it is designed and synthesized separately from the top module to produce required netlist. HLLE calculation scheme requires 2 steps to get the final result:

1. Compute HLLE coefficients and eigenvalues.

2. Compute HLLE numerical flux.

Algorithm 1 Pseudocode for Matrix Multiplication.

```
1: for (i = 0; i < 5; i = i + 1) do
2:     for (j = 0; j < 5; j = j + 1) do
3:         C[i, j] = 0;
4:         for (k = 0; k < 5; k = k + 1) do
5:             C[i, j] = C[i, j] + A[i, k] * B[k, j];
6:         end for
7:     end for
8: end for
```

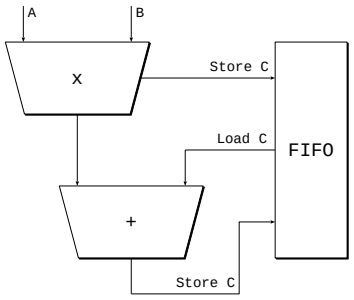

Figure 8: The structure of MAC organization for Roe scheme.

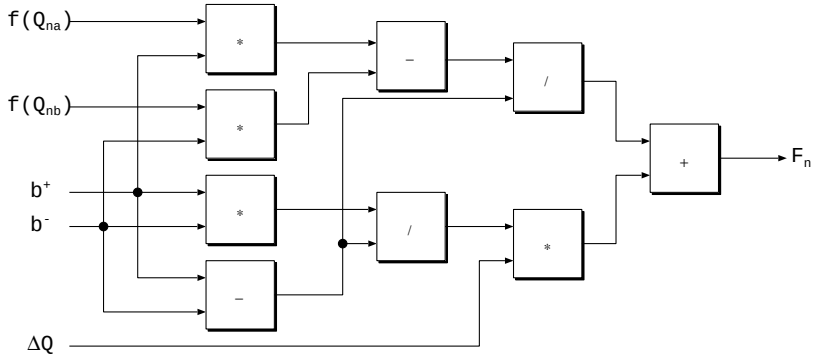

Figure 9: HLLE flux function circuit.

After received the Roe average values, HLLE coefficients, b^+ and b^- are computed using adder, subtractor and comparator computing units. Then, HLLE flux function circuit shown in Figure 9 is used to compute the HLLE numerical flux. This hardware implementation divides the fraction of Eq. 8 into two part. Therefore, calculation of left fraction and right fraction are done in parallel. Summation of these two values makes the total flux.

5.4 HLLEW Scheme Module

Apparently, HLLEW scheme is a modification of HLLE scheme. This scheme is also based on Roe scheme as well as HLLE scheme. However, to compute the HLLEW flux, there is no need to do a matrix computation as suggested for Roe scheme. HLLEW scheme is also implemented as a reconfigurable module in reconfigurable partition. Therefore, it is synthesized beforehand and separately to provide the required netlist to the top module. Computation of HLLEW flux requires the following steps:

1. Compute coefficients and eigenvalues.

2. Compute pressure difference.

3. Compute HLLEW numerical flux.

Again, Roe average values are used here to compute the coefficients and eigenvalues. After that, pressure difference, Δp is calculated. When all values are obtained, inputs are sent to the HLLEW flux function circuit as shown in Figure 10 to compute the numerical flux. This circuit is explicitly designed to calculate the Q_{na} and Q_{nb} data values in parallel. Total flux is obtained after summation of result divided by two.

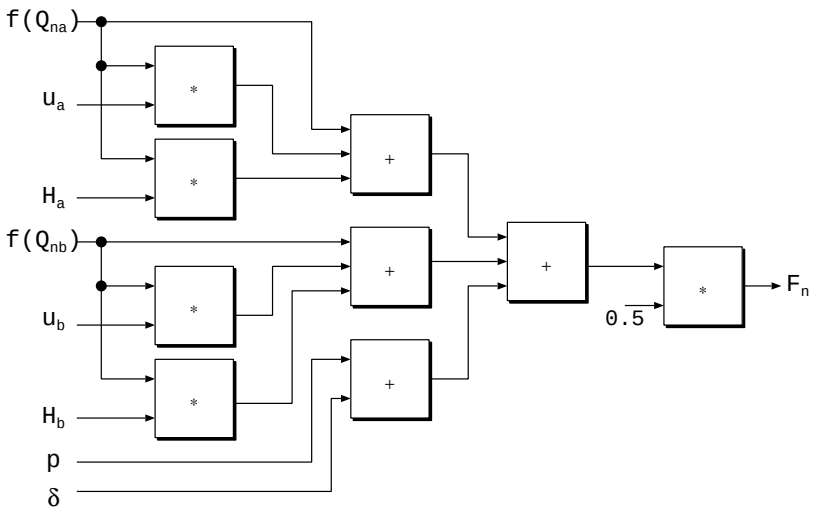

Figure 10: HLLEW flux function circuit.

5.5 Implementation Issues

Our challenge is to implement large scale scientific computation using partial reconfiguration. Careful design requirements and considerations are carried out. At the same time, the design specification must be analyzed thoroughly, and the limitations associated with partial reconfigurable designs are considered. In addition, difficulties must be resolved to improve the quality of the design. The challenges and solutions on implementation are listed as follows:

1. *I/O in each scheme module*
   Flux calculation schemes must include the I/O circuitry, input buffer (IBUF) and output buffer (OBUF) that are required to connect internal logic to package pins. In other words, the I/O features must be completely contained within the scheme module, but the port list for the complete design remains at the top-level design description. Besides, the limitation of the I/O pins of FPGA must be considered, since flux calculation module requires many I/O.

2. *DSP blocks in each scheme module*
   It is also important that the physical region selected has adequate resources especially DSP48E for all schemes. Flux calculation scheme requires a lot of DSP blocks to perform the computation. Therefore, we properly set the last blocks occupied in both end column of reconfigurable partition are DSP blocks, instead of slice or block RAM. Using this strategy, we can maximize DSP blocks in the partition.

3. *Interaction with CORE Generator*
   Since we used CORE Generator to generate all computing units, netlist-based cores were created to be instantiated in the design. To make sure these cores can be instantiated easily, the boundaries of flux calculation scheme partition is not modified. We also made considerations for the definition of the flux calculation scheme region to ensure the proper elements are contained within.

4. *Optimization*
   In order to optimize the design time for bit file generation, the most complicated and highly resource consuming design should be selected first. Here, Roe scheme is done first, then HLLEW follows, and HLLE is done last.

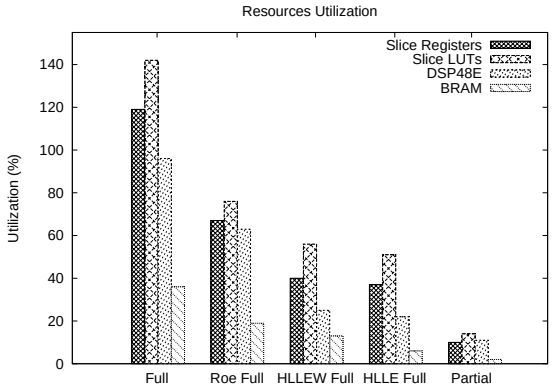

Figure 11: Resources utilization for all possible implementation.

6 Evaluation

In order to demonstrate the effectiveness of our design, we used a sample data of NACA 0012 airfoil. *The National Advisory Committee for Aeronautics* (NACA) develops the NACA airfoils shapes for aircraft wings. The grid data consisting of 11,564 grids with 22,883 faces are used in our study. We evaluated the used resources, configuration speed for full and partial reconfiguration, and system performance.

6.1 Resource Utilization

The amount of required slice registers, slice LUTs, DSP48E and BlockRAM are evaluated when the design is synthesized. There are 3 design options for implementation consideration. Result of the resource utilization is shown in Figure 11. Obviously, the first option is to fit in all modules in a single FPGA, shown in first column noted by "Full". The main advantage of this method is that it only requires one time configuration and no reconfiguration is needed. However, it requires a large amount of hardware that is not enough in a single FPGA. Total resource utilization for registers and LUTs usage exceeds 100%. Registers and LUTs required are 119% and 142% respectively.

Second strategy is implementing only a scheme in one design. Means, for three schemes, there are three difference designs. This is shown in second, third and fourth column noted by "Roe Full", "HLLEW Full" and "HLLE Full". Although there are enough resources to do this, two disadvantages arise. First, it will require two times full reconfiguration if a user wants to change from one scheme to another. Second, resource is overused since the same design is used again except the flux scheme module.

The third option, which is our proposed method, is to utilize partial reconfiguration. This is shown in fifth column noted by "Partial". Top, static and reconfigurable modules are fixed in a single FPGA. Flux calculation schemes bitstreams are stored in the host PC. When users want to use any particular scheme, it is loaded to an FPGA. Consumed resources when no scheme loaded is small. In addition, consumed resources when system is in use and one scheme loaded is the same with the second option. However, another advantage of this technique is the configuration time. If users want to change from one scheme to another, it can be faster compared to the full reconfiguration.

We examined the amount of resources utilization for each design option. We found out that all modules cannot be implemented in single Virtex-6 XC6VLX240T-1FF1156 FPGA since resources available are not enough to accommodate all modules. For the second option design, Roe, HLLEW and HLLE schemes are implemented separately. Although there are enough resources to implement this, all designs require full reconfiguration to load in an FPGA independently. In partially reconfigurable design, bitstreams of Roe, HLLEW and HLLE schemes are stored in host PC. Therefore, maximum resources reduction is measured when Roe scheme is deployed since it requires the highest resources. On average, consumed resources for "Full" design is 98.25% while for partially reconfig-

urable design when Roe scheme is deployed is 56.25%. As a result, resource utilization is successfully reduced by 42% on average.

Even though the resources are not enough to implement all modules in a single FPGA, there are resources overhead in partially reconfigurable design. This is because all schemes modules are implemented in reconfigurable partition. The partition is manually floorplaned and resources allocated are fixed to fit in all modules. Since Roe scheme occupied higher resources than HLLEW and HLLE schemes, the reconfigurable partition has wasted resources when HLLE or HLLEW schemes are loaded. However, unused resources in these schemes are used again when Roe scheme is selected.

6.2 Configuration Time

The configuration time for the full reconfiguration and partial reconfiguration is compared. The speed of configuration is directly related to the size of the partial bit file and the bandwidth of the configuration port. Since we use JTAG configuration port, for Virtex-6 device [17], configuration time is given by:

$$configuration\ time = \frac{(2044 + bits\ in\ bitstream)}{TCK\ frequency}$$

where *bits in bitstream* is size of the configuration bitstream in bits and *TCK frequency* is maximum configuration TCK (*Test Clock*) frequency and used for boundary-scan operations. 2044 is the total number of clocks needed for pre-processing and post-processing for single device configuration sequence while programming the bitstream to FPGA. Although the maximum bandwidth available is 66 Mbps, we found out that while configuring the FPGA using iMPACT, used bandwidth is 16.7 Mbps and data width is 1 bit.

In a full reconfiguration, total bitstream size is 9,017 KB. Based on the given formula, the configuration time is equal to 4.423 sec. On the other hand, bitstream size for each Roe, HLLE and HLLEW scheme is 1422 KB. This means configuration time for partial reconfiguration is 0.704 sec. In short, the partial reconfiguration method accelerated the configuration speed by 6.28 times.

There is no overhead for partial reconfiguration since the users must decide which scheme they want to use before calculation start. Therefore, configuration time will not affect the computation time in FPGA. In addition, configuration time will not cause a bottleneck to the system when the grid size grows large and takes a lot of iterations. This is because all flux schemes bitstream is fixed and not affected by large input size.

6.3 Performance

Flux calculation scheme is implemented and the total clock cycles are measured. Total clock cycles for each Roe, HLLEW and HLLE scheme are 205600×10^3, 197400×10^3 and 191200×10^3 respectively. The execution time for flux calculation scheme in software is compared with the execution time by hardware. The result summary is shown in Figure 12.

In software, all schemes are executed by Core 2 Duo 2.4GHz with Linux Kernel 2.6.18 operating system. All schemes are compiled by using Intel Fortran Compiler 10.1. Execution time is measured by using `call system_clock` prepared in Fortran 90 language. We found out the execution time took 5.400 sec for Roe scheme, 4.533 sec for HLLEW scheme and 4.399 sec for HLLE scheme. This is show in first column of Figure 12, noted by "Software".

In hardware, since we know the total clock cycles required from beginning to the end, and operating frequency for the FPGA is 200 MHz. Therefore, computation time by FPGA for Roe scheme is 1.028 sec, 0.987 sec for HLLEW scheme and 0.956 sec for HLLE scheme. Adding the configuration time and computation time resulting an execution time in hardware. Therefore, the second column of Figure 12 shows an execution time in FPGA if second option design of full reconfiguration (FR) is deployed, noted by "FPGA FR". Adding 4.423 sec configuration time to each scheme computation time of Roe, HLLEW and HLLE scheme give an execution time become 5.451 sec, 5.410 sec and

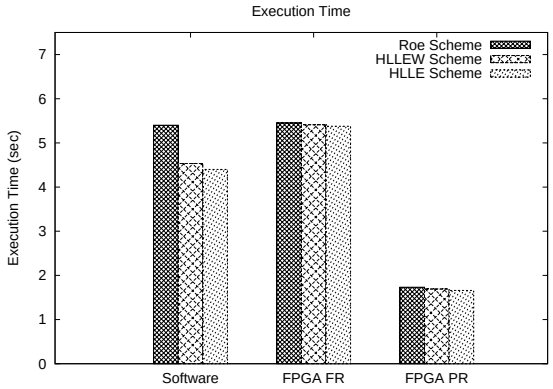

Figure 12: Execution time in software and FPGA.

5.379 sec respectively. Third column of Figure 12 shows an execution time in FPGA if partial reconfiguration (PR) is used, noted by "FPGA PR". Adding 0.704 sec configuration time to each scheme computation time give an execution time of Roe, HLLEW and HLLE scheme are 1.732 sec, 1.691 sec and 1.660 sec respectively. Acceleration speed between hardware and software is compared when HLLE scheme in partially reconfigurable design is deployed since it is executed fastest in software. Therefore, the execution time of FPGA is 2.65 times faster compared to the software execution.

Full reconfiguration design strategy for each scheme gives an almost same performance produced by software. In fact, software execution time for HLLE and HLLEW schemes are faster than FPGA. However, partial reconfigurable design approach gives a 2.65 fold speed-up compared to software execution. Therefore, taking configuration time into account, performance improvement using partial reconfiguration method is justified.

In this implementation, there are three flux calculation schemes for selection. As discussed in Section 6.1, it is possible to design three different schemes separately. However, this design has following three advantages. Firstly, practical software package has various parts, which selectively use several functions. If there are M different parts each of which has N_i functions, the total number of subset designs becomes $\prod_{i=1}^{M} N_i$. By using partial reconfiguration, the users can select their favorites combination easily on demand. Secondly, when we want to add another scheme to the implementation, it is easy to add another reconfigurable module instead of modifying the whole design. Finally, in advection term computation, there is no obvious scheme for every particular flow problem. In other words, flux calculation scheme must be able to yield stable and provide solutions under various flow conditions. Each scheme has their advantages and disadvantages. Therefore, by trying one scheme to another quickly will help users to draw best conclusion. Thus, reducing the switching time by using partial reconfiguration will be helpful.

7 Future Work

In this work, our target is only flux calculation scheme in advection term computation. Therefore, many other routines in FaSTAR are still untapped for further exploration. We must try to extend partial reconfiguration technique more aggressively using other modules that are available in FaSTAR. Instead of single partition, multiple reconfigurable partitions in single FPGA are good design strategy for future research. However, this work shows that single FPGA can fitted only one subroutine in advection term computation.

While the limitations of a single FPGA are noticed, multi-FPGA platform with multiple reconfigurations can be the next target for future work. They offer the potential to mega-boost the capacity of resource in FPGA as well as many more modules in FaSTAR can be reconfigured such as flux evaluation, flux limiter function or convergence acceleration. This approach also hopeful because FaSTAR computational intensive part must be performed iteratively, and that can be parallelized

for high-performance using multiple FPGA. If this target is realized, it is possible to implement the whole FaSTAR package on FPGAs. However, a lot of works need to be done such as the interconnection between FPGA, data transfer and synchronization is another issues to solve before successful implementation could be achieved.

8 Conclusion

In this study, the efficient use of partial reconfigurability in recent FPGAs is explored. Advection term computation is chosen as a target, and flux calculation scheme is deployed as a reconfigurable module. Three flux calculation schemes are implemented: Roe, HLLE and HLLEW schemes.

The implementation using partial reconfiguration platform has successfully reduced required hardware resources, improved configuration time and its performance. Resources utilization is saved up to 42% on average. The proposed design also improves the configuration time by 6.28 times faster and accelerates the system at least 2.65 times in performance.

Acknowledgments

We are grateful to the anonymous reviewers for constructive feedback and insightful suggestions, which greatly improved this article.

References

[1] Takayuki Akamine, Kenta Inakagata, Yasunori Osana, Naoyuki Fujita, and Hideharu Amano. Reconfigurable out-of-order mechanism generator for unstructured grid computation in computational fluid dynamics. In *Field Programmable Logic and Applications (FPL), 2012 22nd International Conference on*, pages 136 –142, Aug. 2012.

[2] E. Andres, M. Molina, G. Botella, A. del Barrio, and J. Mendias. Aerodynamics analysis acceleration through reconfigurable hardware. In *Programmable Logic (SPL), 2008 4th Southern Conference on*, pages 105–110, March 2008.

[3] Christopher Claus, Johannes Zeppenfeld, Florian Müller, and Walter Stechele. Using partial-run-time reconfigurable hardware to accelerate video processing in driver assistance system. In *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '07, pages 498–503, San Jose, CA, USA, 2007. EDA Consortium.

[4] Bernd Einfeldt. On Godunov-type methods for gas dynamics. *SIAM J. Numer. Anal.*, 25(2):294–318, April 1988.

[5] Amiram Harten, Peter D. Lax, and Bram van Leer. On upstream differencing and Godunov-type schemes for hyperbolic conservation laws. *SIAM Review*, 25(1):pp. 35–61, 1983.

[6] Atsushi Hashimoto, Keiichi Murakami, Takeshi Aoyama, Manabu Hishida, Lahur Paulus R., Masahide Sakashita, and Yukio Sato. Development of fast flow solver FaSTAR. In *Proc. 42nd Fluid Dynamics Conference/Aerospace Numerical Simulation Symposium*, 2010.

[7] Yohei Hori, Hiroyuki Yokoyama, Hirofumi Sakane, and Kenji Toda. A secure content delivery system based on a partially reconfigurable FPGA. *IEICE - Trans. Inf. Syst.*, E91-D:1398–1407, May 2008.

[8] H.M. Hussain, K. Benkrid, H. Seker, and A.T. Erdogan. FPGA implementation of K-means algorithm for bioinformatics application: An accelerated approach to clustering microarray data. In *Adaptive Hardware and Systems (AHS), 2011 NASA/ESA Conference on*, pages 248–255, June 2011.

[9] Qiwei Jin, Diwei Dong, Anson H.T. Tse, Gary C.T. Chow, David B. Thomas, Wayne Luk, and Stephen Weston. Multi-level customisation framework for curve based monte carlo financial simulations. In *Reconfigurable Computing: Architectures, Tools and Applications*, volume 7199 of *Lecture Notes in Computer Science*, pages 187–201. Springer, 2012.

[10] B.J. LaMeres and C. Gauer. Dynamic reconfigurable computing architecture for aerospace applications. In *Aerospace conference, 2009 IEEE*, pages 1–6, March 2009.

[11] Shigeru Obayashi and Yasuhiro Wada. Practical formulation of a positively conservative scheme. *AIAA Journal*, 32(5):1093–1095, 1994.

[12] Bjorn Osterloh, Harald Michalik, Bjorn Fiethe, and Frank Bubenhagen. Architecture verification of the SoCwire NoC approach for safe dynamic partial reconfiguration in space applications. In *Adaptive Hardware and Systems (AHS), 2010 NASA/ESA Conference on*, pages 1–8, June 2010.

[13] P.L Roe. Approximate Riemann solvers, parameter vectors, and difference schemes. *Journal of Computational Physics*, 43(2):357 – 372, 1981.

[14] K. Sano, T. Iizuka, and S. Yamamoto. Systolic architecture for computational fluid dynamics on FPGAs. In *Field-Programmable Custom Computing Machines, 2007. FCCM 2007. 15th Annual IEEE Symposium on*, pages 107–116, April 2007.

[15] K. Sano, O. Pell, W. Luk, and S. Yamamoto. FPGA-based streaming computation for Lattice Boltzmann method. In *Field-Programmable Technology, 2007. ICFPT 2007. International Conference on*, pages 233–236, Dec. 2007.

[16] Mohamad Sofian Abu Talip, Takayuki Akamine, Yasunori Osana, Naoyuki Fujita, and Hideharu Amano. Partial reconfiguration of flux limiter functions in MUSCL scheme using FPGA. *IEICE Transactions on Information and Systems*, 95-D(10):2369–2376, Oct. 2012.

[17] Xilinx Inc. *Virtex-6 FPGA Configuration User Guide v3.2 UG360*, Nov 2010.

[18] Hiroyuki Yamazaki, Shunji Enomoto, and Kazuomi Yamamoto. A common CFD platform UPACS. In *Proceedings of the Third International Symposium on High Performance Computing*, ISHPC '00, pages 182–190, London, UK, 2000. Springer-Verlag.