PAPER

Partial Reconfiguration of Flux Limiter Functions in MUSCL Scheme Using FPGA

Mohamad Sofian ABU TALIP[†a)], **Takayuki AKAMINE**[†], ***Nonmembers***, **Yasunori OSANA**[††], ***Member***, **Naoyuki FUJITA**[†††], ***Nonmember***, ***and*** **Hideharu AMANO**[†], ***Member***

SUMMARY Computational Fluid Dynamics (CFD) is used as a common design tool in the aerospace industry. UPACS, a package for CFD, is convenient for users, since a customized simulator can be built just by selecting desired functions. The problem is its computation speed, which is difficult to enhance by using the clusters due to its complex memory access patterns. As an economical solution, accelerators using FPGAs are hopeful candidate. However, the total scale of UPACS is too large to be implemented on small numbers of FPGAs. For cost efficient implementation, partial reconfiguration which dynamically loads only required functions is proposed in this paper. Here, the MUSCL scheme, which is used frequently in UPACS, is selected as a target. Partial reconfiguration is applied to the flux limiter functions (FLF) in MUSCL. Four FLFs are implemented for Turbulence MUSCL (TMUSCL) and eight FLFs are for Convection MUSCL (CMUSCL). All FLFs are developed independently and separated from the top MUSCL module. At start-up, only required FLFs are selected and deployed in the system without interfering the other modules. This implementation has successfully reduced the resource utilization by 44% to 63%. Total power consumption also reduced by 33%. Configuration speed is improved by 34-times faster as compared to full reconfiguration method. All implemented functions achieved at least 17 times speed-up performance compared with the software implementation.

key words: *computational fluid dynamics (CFD), field programmable gate array (FPGA), scientific computations, reconfigurable hardware, partial reconfiguration*

1. Introduction

CFD (*Computational Fluid Dynamics*) has been widely utilized extensively in the design and optimization of fluid flow applications since many years ago. In aerospace industry, CFD is a cost-effective design tool for aircraft components such as jet engines and wings. It presents methods to solve and analyze problems of the physical phenomena of fluids involving fluid flow on discrete space and time. Therefore, software packages for CFD with high accuracy are needed for aeronautical engineers and researchers. However, the long computation time required to simulate complete aircraft configurations remains as a bottleneck in the design flow of new structures for the aeronautics industry. Thus, reducing the time for aerodynamics analysis is one of the most important challenges of current research in this field.

A typical simulation platform in the aeronautics industry consists of a CFD specific software application, normally written in a high-level language. UPACS (*Unified Platform for Aerospace Computational Simulation*) [1], [2] developed by JAXA (*Japan Aerospace Exploration Agency*) is one of such CFD packages. UPACS adopts structured mesh as grid data in their simulation. Although it is a convenient tool for aerodynamics analysis, it sometimes takes several days or weeks when an analytical area grows large [3]. This is mainly caused by low parallel processing efficiency accompanied with pointer links and a complicated memory access pattern. In UPACS, the increasing demands for accuracy and simulation capabilities produce an exponential growth of the required computational resources. Cluster computing or GPU which makes use of a high degree of parallelism is not an efficient solution [4].

Recently, reconfigurable systems using FPGAs have been utilized for acceleration of specific applications including bio-informatics, digital image processing, finance and others [5]–[7]. 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 [8], [9], targets were simple programs rather than practical software packages.

The goal of our research is to improve the performance of UPACS by using FPGAs as a reconfigurable platform. However, the whole UPACS package is too large to implement in an FPGA, and often larger than the size of the target platform. In our previous research efforts [10], we tried to implement core subroutines in UPACS which has complicated memory access into FPGAs. Although the performance can be improved, it is found that the target requires too vast resources to implement on a small number of FPGAs. Use of multiple FPGAs is one option to resolve this problem. However, we want to save the number of FPGAs to be used, and also reduce power by introducing partial reconfiguration provided in recent FPGAs. Since UPACS consists of various solvers, not all of them are needed to solve a target application. As the first step in this study, we introduce this mechanism into flux limiter functions in MUSCL

Manuscript received March 13, 2012.
Manuscript revised June 16, 2012.

[†]The authors are with the Graduate School of Science and Technology, Keio University, Yokohama-shi, 223–8522 Japan.
[††]The author is with the Department of Electrical and Electronics Engineering, University of the Ryukyus, Okinawa-ken, 903–0213 Japan.
[†††]The author is with Japan Aerospace Exploration Agency (JAXA), Chofu-shi, 182–8522 Japan.
a) E-mail: cfd@am.ics.keio.ac.jp

DOI: 10.1587/transinf.E95.D.2369

scheme available in UPACS package.

The rest of this paper is organized as follows. Section 2 discusses related work to this study. Section 3 is an explanation regards to UPACS package whereas the target subroutines and MUSCL scheme are explained. Section 4 is about partial reconfiguration technology. Section 5 describes about the implementation of this work. Then following by Section 6 for evaluation and Sect. 7 will summarize this work with conclusion.

2. Related Work

In fluid dynamics simulation, execution time or computations time is the largest concern. There are various studies using FPGAs reported so far to examine the issue. Andres et al. [9] and Sano et al. [11] reported the result for FPGA accelerations. However, their implementations are not for practical software packages. Another result is reported for implementation of FPGA-based flow solver based on the systolic architecture for CFD [12]. This work proposed a systolic algorithm for the fractional-step method employing the central difference schemes. Although good results are obtained, their implementation is based on 32-bit single precision which is not sufficient for practical fluid dynamics analysis [13].

A reconfigurable hardware platform has flexibility to fill the gap between hardware and software for algorithm implementation. Therefore, partial reconfiguration technology has attracted many researchers in this field. Recently, this technology was applied to information and communication system security [14]. In computer security applications field, partial reconfiguration is applied in AES algorithm implementation [15]. It is also had been used to accelerate video processing in driver assistance system [16]. It is noted that research has been done on an aerospace application using partial reconfiguration method. LaMeres et al. [17] designed and prototyped the computing architecture which dynamically reconfigures the system depending on the environment. Another work has been done on the framework for a highly reliable fault tolerant system using partial reconfiguration in spaces applications [18]. However, none of them are in the CFD applications. Our trial is the first implementation example of CFD on FPGAs with partial reconfiguration.

3. UPACS

UPACS is a CFD package to simulate compressible flow using multi-block grids. It provides researchers an easy way to run large scale simulations. It has been developed as a common aerospace CFD software equipping with flexibility, scalability and portability since 1998. The application is written in FORTRAN 90 and it supports the MPI interface.

UPACS supports Euler, Navier-Stokes and Reynolds Averaged Navier-Stokes equations as governing equations. By choosing solvers, users can execute simulations on their parallel systems without any code tuning. Users also can select desired solutions and determine the number of process by setting parameters. In order to run a simulation, users just prepare a parameter file and grid data files. Figure 1 shows the simulation flow and profiling result of UPACS and its correspondent percentage of execution time. Note that, the percentage of the execution time is only shown from "boundary condition pressure" to "calculate residual", and thus the sum does not 100%. This profile had taken on SPARC64V processors at 1.3 GHz with Solaris8 operating system with 40^3 grid data.

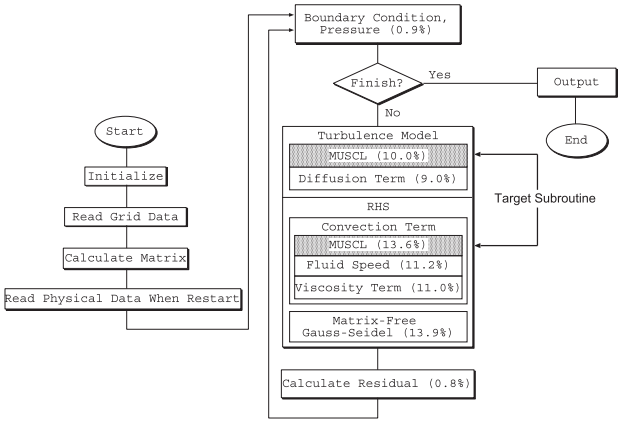

Fig. 1 UPACS profiling result.

In this study, we focus on MUSCL scheme subroutine, since this subroutine is used twice in core routine of UPACS from turbulence model to calculate residual. The core routine occupies about 70% of total execution time and its ratio grow up more than 90% as grid size increases [19].

3.1 MUSCL Scheme

MUSCL (*Monotone Upstream-centered Schemes for Conservation Laws*), which is a method to improve the accuracy, was introduced in a paper by Bram van Leer in 1979 [20]. It provides highly accurate numerical solutions for a given system. In UPACS, MUSCL is used in the turbulence model (TMUSCL) and the convection term (CMUSCL) calculation. It extrapolates cell surface values from cell center values shown in formula (1) to (4) and Fig. 2.

$$q'_{i+1/2} = (q_{i+1} - q_i)/(\Delta_{i+1} + \Delta_i) \tag{1}$$

$$q'_{i-1/2} = (q_i - q_{i-1})/(\Delta_i + \Delta_{i-1}) \tag{2}$$

$$q_{i\pm1/2} \cong q_i \pm \phi(r)\Delta_i q'_{i-1/2} \tag{3}$$

$$r = (q'_{i+1/2})/(q'_{i-1/2}) \tag{4}$$

In the Eqs. (1) to (4), q_i is the cell center value, Δ_i represents the distance between cell center and cell surface, $q_{1/2}$ is the cell surface value, and $\phi(r)$ is the flux limiter function. i in the formulas indicates the direction which can be extended to three dimensions. In addition, q_i consists of five physical values in UPACS, and there are data dependency between them. These physical values are density, velocity, pressure, viscosity and energy. FLFs are used to

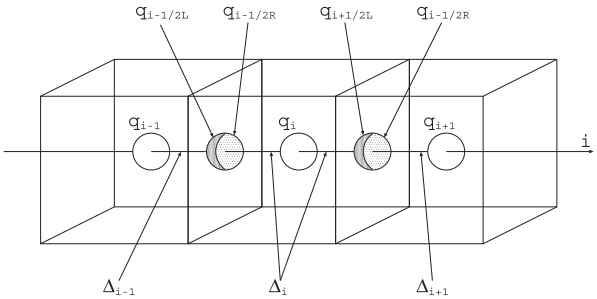

Fig. 2 MUSCL scheme.

suppress oscillation of values, which often arises in the field where values change rapidly with a high order difference scheme. Six FLFs of MUSCL are shown in Eqs. (5) to (10). In TMUSCL, four symmetry FLFs *no limiter, van Albada, van Leer* and *minmod* limiter functions are included. On the other hand, 2nd order CMUSCL uses *no limiter, van Albada, van Leer, minmod* and *superbee* limiter functions. Finally, 3rd order CMUSCL consists of *no limiter, minmod* and *Hemker-Koren* limiter functions.

$$\textit{no limiter}, \quad \phi(r) = 0.5 * (r + 1) \tag{5}$$

$$\textit{van Leer}, \quad \phi(r) = \frac{(r + |r|)}{(1 + r + EPS)} \tag{6}$$

$$\textit{van Albada}, \quad \phi(r) = \frac{(r^2 + r)}{(1 + r^2 + EPS)} \tag{7}$$

$$\textit{minmod}, \quad \phi(r) = max[0, min(1, r)] \tag{8}$$

$$\textit{superbee}, \quad \phi(r) = [0, min(2r, 1), min(r, 2)] \tag{9}$$

$$\textit{Hemker-Koren}, \quad \phi(r) = \frac{(r + 2r^2)}{(2 - r + 2r^2)} \tag{10}$$

Here, r comes from (4), and EPS is a machine epsilon (1×10^{-16}).

4. Partial Reconfiguration

The flexibility of FPGA raises the possibility for hardware configurations with software as needed to improve efficiency, robustness, security and capability to be programmable on the fly. A partially reconfigurable design of an FPGA consists of three major modules: the top module, static module and reconfigurable modules (RMs). The top module includes the static module and the RMs. The static module is a set of non-reconfigurable modules, while RMs are the dynamically reconfigurable part of the design. The area of the device in which RMs is implemented is called Reconfigurable Partition (RP).

MUSCL scheme is used twice in UPACS execution flow. Therefore, at the beginning, users must specify which limiter function they want to use at both parts. First, MUSCL is used in turbulence model with four FLFs got involved. Then, MUSCL is used again in convection term calculations part. 2nd order calculation for convection term involves 5 FLFs, and 3 FLFs are available for 3rd order

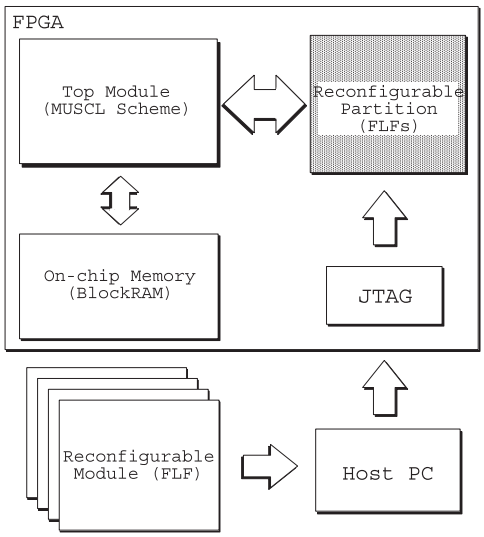

Fig. 3 High level system overview.

calculation. However, in CMUSCL calculation part, 2nd order CMUSCL and 3rd order CMUSCL are alternatively used. Here, partial reconfigurability of the FPGA and intractability of the bitstream is effective to meet the requirements. Figure 3 shows the block diagram of the system. In FPGA, the system consists of top MUSCL module, Reconfigurable Partition module for FLFs and on-chip memory using BlockRAM. The system is connected with the host PC which contains all FLFs bitstreams. The connection is through UART via a JTAG port. JTAG is chosen for configuration port because of quick testing and debugging. Although ICAP port is a good alternative, it requires user-designed PR controller such as custom state machine or embedded processor such as MicroBlaze™. Moreover, when the partial bit files are stored in host PC, JTAG is convenient compared to self configurations using ICAP.

Since each MUSCL function has similar structure except FLFs, we can design a single MUSCL module with all FLFs for TMUSCL, 2nd order CMUSCL and 3rd order CMUSCL. However, it becomes a large hardware which is difficult to be implemented on a single FPGA. The straight forward way is designing three MUSCLs each of which has their own FLFs. Although this approach reduces the hardware, we must provide three independent designs. Our approach is to provide a single design whose FLF can be replaced by making the best use of partial reconfiguration. The total required hardware and power usage can be minimized, since it provides only a single FLF required in the target application. When the execution of UPACS starts and the functions required for MUSCL is decided, an appropriate FLF module is loaded by using the partial reconfiguration while the other part of FPGA is remaining unaffected.

Each FLF module has the same inputs and outputs, thus, it can be specified in the HDL description as the functional modules with the reconfigurable partition attribute in the description of the MUSCL top module. Multiple instances corresponding to each FLF can be defined for such

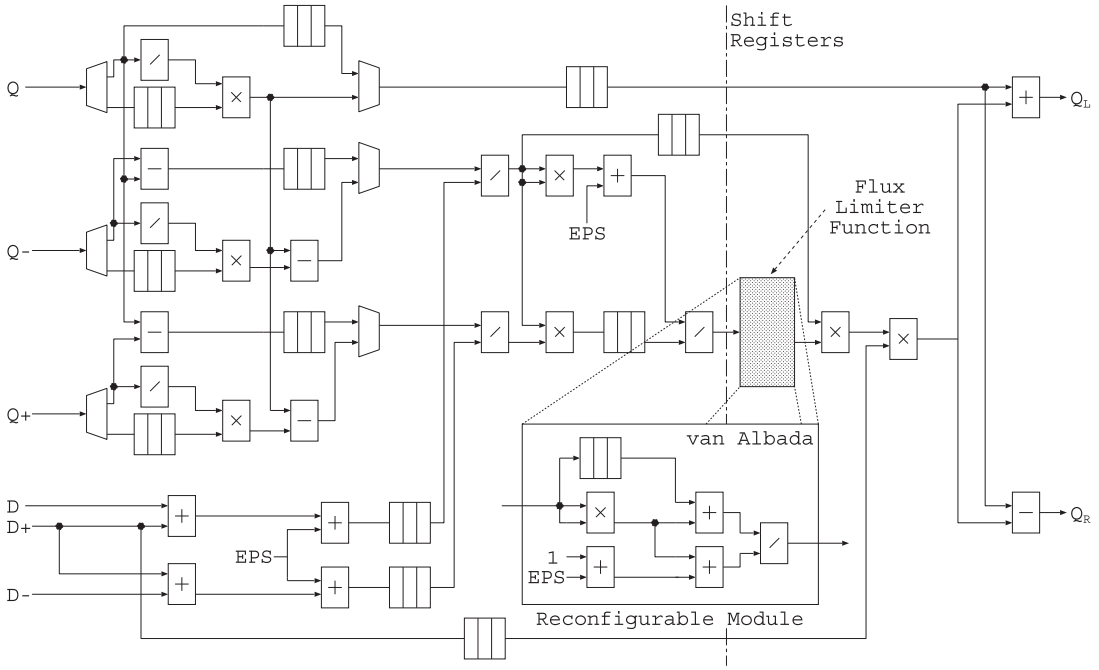

Fig. 4 MUSCL pipeline with *van Albada* limiter function.

a single functional module. Software tools as NGDBuild, MAP and PAR detect the reconfigurable partition attribute on the instance and process it correctly [21].

5. Implementation

Here, Xilinx Virtex-6 FPGA (XC6VLX240T-1FF1156) which supports a partial reconfiguration is chosen as a target device. MUSCL scheme is implemented as a top, static module with a reconfigurable partition. Using its reconfigurable partition, FLFs are implemented as partial reconfigurable modules. The datapath can be obtained from partially simplified data-flow representation of the algorithm shown in Fig. 4. By inserting shift registers in the datapath, the fundamental structure of the pipeline is designed. Xilinx CORE Generator is used to provide the core for floating-point adder, subtractor, multiplier, divider and shift register. Efficient memory system is a concern overhere. However, it is beyond the scope of this paper and had been reported in our previous study [22]. To solve 3D model of fluid dynamics problem, pipeline datapath is implemented three times inside an FPGA.

All modules are described using Verilog HDL and simulated with Xilinx ISim Simulator. The modules are synthesized and used resources are measured using Xilinx ISE 12.4. Floor-planning, constraint entry and design rule checks (DRCs) are all accessed through the PlanAhead 12.4 software environment which supports a partial reconfiguration flow. In order to demonstrate that our system works on the real FPGA, Xilinx ML605 board is used with 200 MHz operating frequency. All modules also 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. In order to generate high performance computation unit, the level of DSP48E usage is set to the maximum to get the desired output.

Table 1 Data of used computing units.

Units	Latency	Registers	LUTs	DSP48E
Adder	14	947	797	3
Subtractor	14	947	798	3
Multiplier	16	483	362	11
Divider	57	5973	3261	0
Comparator	1	0	128	0

Inputs given to the pipeline are vectors each of which is consisting of five physical values mentioned before. At one time, only one FLF is used and employed in the FPGA. All FLFs are synthesized separately from the top module. The top MUSCL and reconfigurable FLF modules are consisting of many arithmetic functions. The parameters used for each computing unit are shown in Table 1. Adder and Subtractor are set to 14 clock cycles per operation using high speed mode. In addition, Multiplier takes 16 clock cycles with 11 DSP48E modules. The latency of Divider is set to be 57. Although it is possible to decrease the divider pipeline latency, it will severely degrade the clock frequency.

The designs for all FLFs are shown in Fig. 5. The one with the smallest clock cycles is *minmod* limiter function which only requires 2 clock cycles. The largest latency is by *Hemker-Koren* limiter function which requires 117 clock cycles to get result. *Van Albada* and *van Leer* limiter functions require 87 and 85 clock cycles, respectively. In these FLF

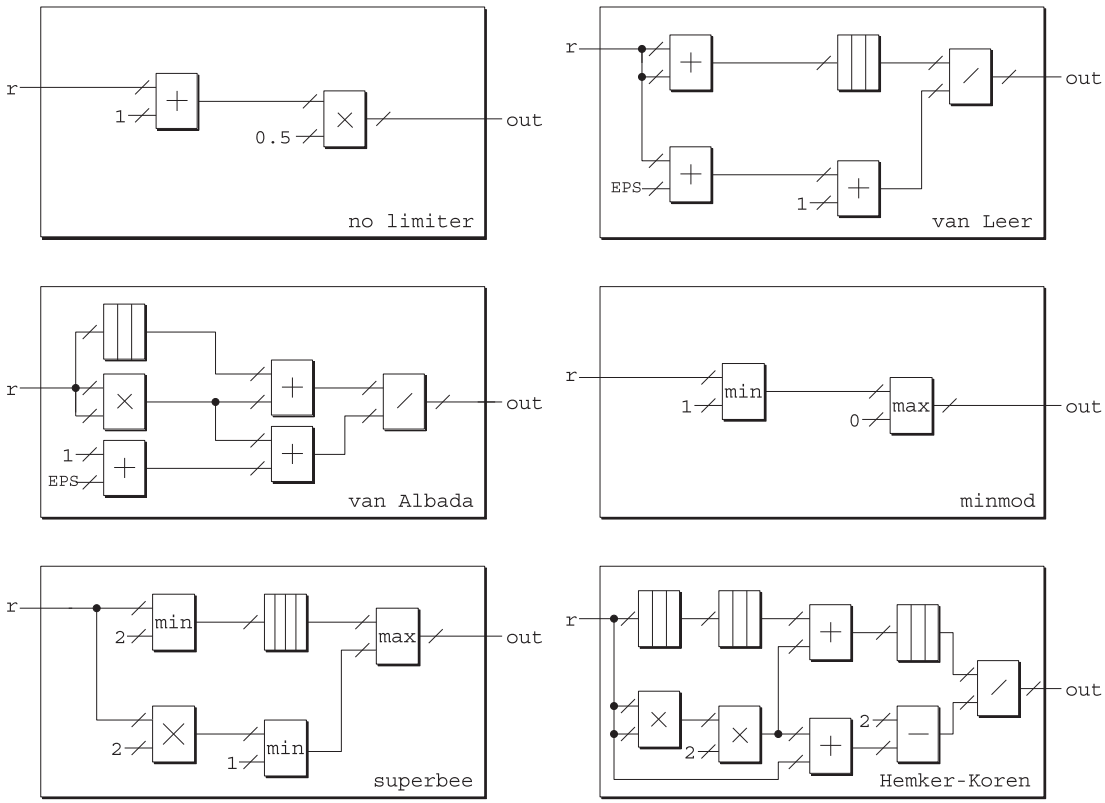

Fig. 5 Implemented flux limiter functions.

modules, shift registers are used to synchronize the input value. Shift registers are 64 bit width and can take various clock cycle depth depending on the situation. Machine epsilon (1×10^{-16}) and constant value 1.0 are also used in these FLFs modules. The remaining *minmod* and *superbee* limiter functions require comparator modules which are used for minimum and maximum value comparison.

6. Evaluation

In this section, evaluation results are shown in order to demonstrate the concept of the system. The following three designs are evaluated and compared.

- *design-1:* One static module of MUSCL scheme with all FLFs.
- *design-2:* Three static modules only with the associate FLFs for TMUSCL, 2nd order CMUSCL and 3rd order CMUSCL.
- *design-3:* One top module of MUSCL scheme with partial reconfiguration FLFs. This is the proposed system.

6.1 Resources Utilization

The amount of required slice registers, slice LUTs and DSP48E are evaluated when the design is synthesized. The results for all three designs are shown in Fig. 6 and Fig. 7.

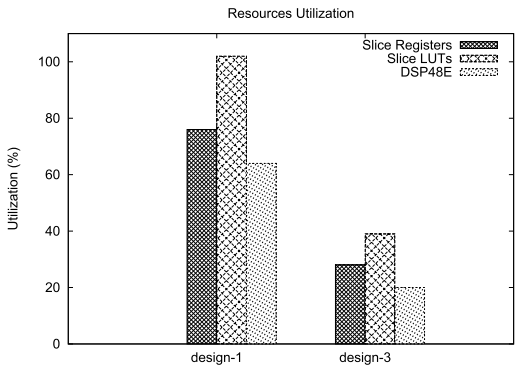

Fig. 6 Resource usage in *design-1* and *design-3*.

The results show that *design-3* has the lowest resource utilization compared to *design-1* and *design-2*. The largest resource is occupied by *design-1*. The slice LUT usage had exceeded 100%, so it cannot be implemented on a single chip. *Design-2* has less resource required for implementation compared to *design-1*. However, *design-2* will require three different FPGAs or three times full reconfiguration on an FPGA.

In *design-2* TMUSCL has four FLFs: *no limiter, van Leer, van Albada* and *minmod*. CMUSCL 2nd has five FLFs: *no limiter, van Leer, van Albada, minmod* and *superbee*. CMUSCL 3rd has three FLFs: *no limiter, minmod* and *Hemker-Koren*. In *design-3*, all FLFs share the same reconfigurable partition in MUSCL. In contrast, the over-

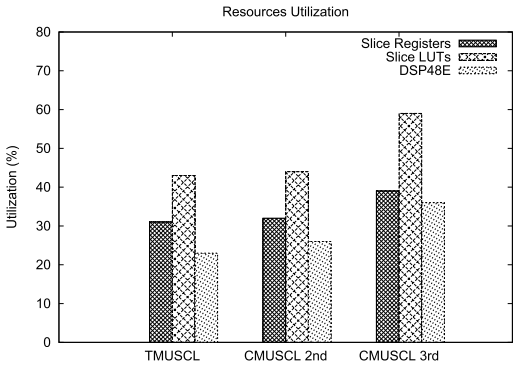

Fig. 7 Resource usage in *design-2*.

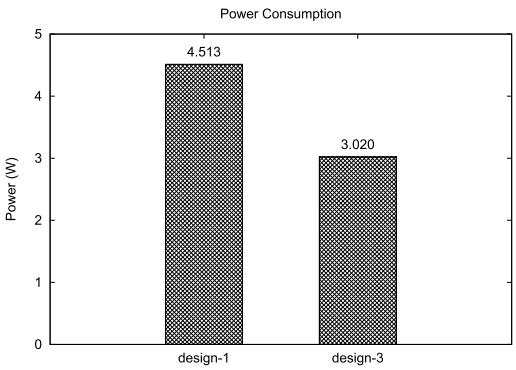

Fig. 8 Total on-chip power for *design-1* and *design-3*.

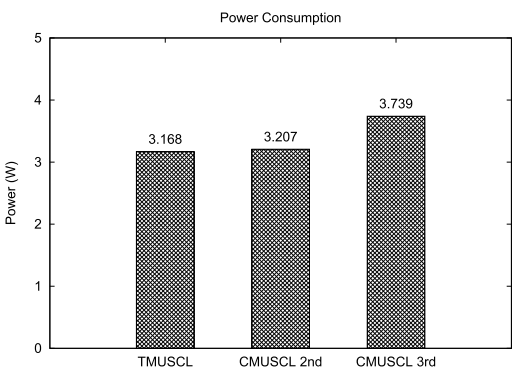

Fig. 9 Total on-chip power for *design-2*.

head in resource utilization is small. It comes from unused resource in reconfigurable partition. Since the partition size is fixed, each FLF is not fully used the resources available. However, in this case, the overhead is negligible.

6.2 Power Consumption

Power consumption becomes one of the biggest concerns in FPGA design as capacity and performance of FPGAs have been increased. The results of total power consumption for all three designs are shown in Fig. 8 and Fig. 9. In *design-1*, Xilinx XPower Estimator (XPE) 13.3 is used since the design cannot be implemented in an FPGA. Resources usage from synthesis result of *design-1* is used as an input to XPE to estimate total power usage. It shows that *design-3* has the lowest power consumption compared to the other two designs. On the other hand, the highest power is consumed by *design-1*.

In *design-1*, the total power is high because of static power by unused limiter functions module. In *design-3*, the total power is only consumed by top MUSCL and the required limiter function. Moreover, the power for *design-2* is less than that for *design-1*, but *design-3* is advantageous even when it is compared with *design-2*. This is because *design-2* also has unused limiter functions in every TMUSCL, CMUSCL 2nd and CMUSCL 3rd during operation. This increases the static power.

6.3 Configuration Time

The configuration time for full reconfiguration and partial reconfiguration are compared. In this case, only *design-2* and *design-3* are evaluated, since *design-1* cannot be implemented on a single chip.

In the case of JTAG configuration, for Virtex-6 device, configuration time is given by:

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

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. In this case, for Virtex-6 device with -1 speed grade, TCK frequency is 66 MHz. 2044 is the total number of clock cycles needed for pre-processing and post-processing while programming the bitstream to FPGA.

In full reconfiguration, each MUSCL module bitstream size is 9,017 KB. Based on the above formula, the configuration time is equal to 1.119 sec. On the other hand, bitstream size for each partial reconfiguration bit file for the 2nd order FLFs is 255 KB. This means that the configuration time is equal to 0.031 sec. In the case of the 3rd order FLFs, partial bitstream size is 266 KB and corresponding configuration time is 0.033 sec. In short, the partial reconfiguration method accelerated the configuration speed by 34 times. In other words, execution time is not so degraded compared with *design-2* when the FLFs are switched dynamically.

In our implementation, the MUSCL will stop working when the FLF is loaded to reconfigurable partition in the FPGA. However, since the time taken to load the FLF is between 0.031 sec to 0.033 sec, this small overhead is acceptable. Even if low speed JTAG is used through iMPACT tool to reconfigure the FPGA, the time to change from one FLF to the other is less than one second, and hard to been recognized by human eyes. Usually, the partial reconfiguration is done once when TMUSCL is changed into 2nd-order CMUSCL or 3rd-order CMUSCL in a job which requires large execution time. Thus, the overhead for partial reconfiguration will be acceptable.

Table 2 Clock-cycle in TMUSCL.

TMUSCL

Flux Limiter Function	# Clock-cycle
no limiter	233
van Albada	290
van Leer	288
minmod	204

Table 3 Clock-cycle in CMUSCL.

CMUSCL

Flux Limiter Function	# Clock-cycle
no limiter 2nd	233
van Albada 2nd	290
van Leer 2nd	288
minmod 2nd	204
superbee 2nd	222
no limiter 3rd	233
minmod 3rd	204
Hemker-Koren 3rd	320

6.4 Performance

MUSCL is implemented with pipelined structure and the clock cycles are measured. Total clock cycles for MUSCL with each FLF is shown in Table 2 and Table 3. The number of clock cycles is corresponding to the time for solving an iteration. In TMUSCL, *van Albada* is the largest, and it takes 290 clock cycles. In CMUSCL, *Hemker-Koren* requires 320 clock cycles to get the result.

The execution time in MUSCL with partial reconfigurable FLFs is compared with the execution time by software. In software, MUSCL is executed by Core 2 Duo 2.4 GHz with Linux Kernel 2.6.18 operating system. The compiler used is GNU Fortran 4.1.2. The execution time to solve $100 \times 100 \times 100$ iterative calculation is measured by using *call cpu_time* in Fortran 90 language and the 3rd order *Hemker-Koren* is selected for comparison, since it has the largest clock cycles. In software, the execution time took 0.08399 sec, while it takes 320 clock cycles to finish one iterative calculations in the FPGA. Adding the time for I/O sending the data sequentially, it took 1,000,320 clock cycles to finish the whole simulation. 1,000,000 comes from the grid size corresponding to the total mesh points. Since the operating frequency in the FPGA is 200 MHz, the total execution time is 5.0016×10^{-3} sec. That is, by execution of CMUSCL in FPGA, about 17 times acceleration is expected. Since the grid size will grow large and take a lot of iterations, configuration time will not caused a bottleneck to the system. Furthermore, overhead also did not influence the operation frequency.

7. Conclusion

UPACS is a convenient CFD package that allows users to select various set of solutions. UPACS solver together with support utilities has proven its effectiveness in a simulating flows around complex configurations using multi-block structured grid scheme. However, it is hard to be implemented even on a large FPGAs because of its complicated structure. Therefore, exploitation of partial reconfigurability in recent FPGAs is considered.

MUSCL scheme using partial reconfiguration platform has been implemented to reducing required hardware resource, power consumption, configuration time and improve its performance. This implementation successfully reduced the resource utilization by 44% to 63%. Also, power consumption was reduced by 33%. Configuration speed is accelerated 34 times faster and overall speed-up at least 17 times in performance is achieved. In future work, other UPACS module is possible for exploration to implement using partial reconfiguration techniques.

Acknowledgements

This work is supported in part by a Grant-in-Aid for the Global Center of Excellence for High-Level Global Cooperation for Leading-Edge Platform on Access Spaces from the Ministry of Education, Culture, Sport, Science and Technology in Japan.

References

[1] H. Yamazaki, S. Enomoto, and K. Yamamoto, "A common CFD platform UPACS," Proc. 3rd International Symposium on High Performance Computing, ISHPC '00, pp.182–190, London, UK, Springer-Verlag, 2000.
[2] R. Takaki, K. Yamamoto, T. Yamane, S. Enomoto, and J. Mukai, "The development of the UPACS CFD environment," ISHPC'03, pp.307–319, 2003.
[3] Y. Matsuo, M. Tsuchiya, M. Aoki, N. Sueyasu, T. Inari, and K. Yazawa, "Early experience with aerospace CFD at JAXA on the fujitsu PRIMEPOWER HPC2500," Supercomputing, 2004. Proc. ACM/IEEE SC2004 Conference, p.11, Nov. 2004.
[4] B. Cope, P.Y.K. Cheung, W. Luk, and L. Howes, "Performance comparison of graphics processors to reconfigurable logic: A case study," IEEE Trans. Comput., vol.59, no.4, pp.433–448, April 2010.
[5] N. Alachiotis and A. Stamatakis, "FPGA acceleration of the phylogenetic parsimony kernel?," Field Programmable Logic and Applications (FPL), 2011 International Conference on, pp.417–422, Sept. 2011.
[6] K. Nakano and E. Takamichi, "An image retrieval system using FPGAs," Design Automation Conference, 2003. Proc. ASP-DAC 2003. Asia and South Pacific, pp.370–373, Jan. 2003.
[7] A. Kaganov, P. Chow, and A. Lakhany, "FPGA acceleration of monte-carlo based credit derivative pricing," Field Programmable Logic and Applications, 2008. FPL 2008. International Conference on, pp.329–334, Sept. 2008.
[8] W.D. Smith and A.R. Schnore, "Towards an RCC-based accelerator for computational fluid dynamics applications," J. Supercomput., vol.30, pp.239–261, Dec. 2004.
[9] E. Andres, M. Molina, G. Botella, A. del Barrio, and J. Mendias, "Aerodynamics analysis acceleration through reconfigurable hardware," Programmable Logic, 2008 4th Southern Conference on, pp.105–110, March 2008.
[10] H. Morisita, K. Inakagata, Y. Osana, N. Fujita, and H. Amano, "Implementation and evaluation of an arithmetic pipeline on FLOPS-2D: Multi-FPGA system," SIGARCH Comput. Archit. News, vol.38, pp.8–13, Jan. 2011.
[11] K. Sano, O. Pell, W. Luk, and S. Yamamoto, "FPGA-based

streaming computation for lattice boltzmann method," Field-Programmable Technology, 2007. ICFPT 2007. International Conference on, pp.233–236, Dec. 2007.
[12] K. Sano, T. Iizuka, and S. Yamamoto, "Systolic architecture for computational fluid dynamics on FPGAs," Field-Programmable Custom Computing Machines, 2007. FCCM 2007. 15th Annual IEEE Symposium on, pp.107–116, April 2007.
[13] F. de Dinechin and G. Villard, "High precision numerical accuracy in physics research," Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment, vol.559, no.1, pp.207–210, 2006.
[14] Y. Hori, H. Yokoyama, H. Sakane, and K. Toda, "A secure content delivery system based on a partially reconfigurable FPGA," IEICE Trans. Inf. & Syst., vol.E91-D, no.5, pp.1398–1407, May 2008.
[15] Y. Hori, A. Satoh, H. Sakane, and K. Toda, "Bitstream encryption and authentication with AES-GCM in dynamically reconfigurable systems," Field Programmable Logic and Applications, 2008. FPL 2008. International Conference on, pp.23–28, Sept. 2008.
[16] C. Claus, J. Zeppenfeld, F. Müller, and W. Stechele, "Using partial-run-time reconfigurable hardware to accelerate video processing in driver assistance system," Proc. Conference on Design, Automation and Test in Europe, DATE '07, pp.498–503, San Jose, CA, USA, EDA Consortium, 2007.
[17] B. LaMeres and C. Gauer, "Dynamic reconfigurable computing architecture for aerospace applications," Aerospace Conference, 2009 IEEE, pp.1–6, March 2009.
[18] B. Osterloh, H. Michalik, S. Habinc, and B. Fiethe, "Dynamic partial reconfiguration in space applications," Adaptive Hardware and Systems, 2009. AHS 2009. NASA/ESA Conference on, pp.336–343, Aug. 2009.
[19] K. Inakagata, H. Morishita, Y. Osana, N. Fujita, and H. Amano, "Modularizing flux limiter functions for a computational fluid dynamics accelerator on FPGAs," Field Programmable Logic and Applications, 2009. FPL 2009. International Conference on, pp.654–657, Sept. 2009.
[20] B. van Leer, "Towards the ultimate conservative difference scheme. V. A second-order sequel to Godunov's method," J. Computational Physics, vol.32, no.1, pp.101–136, 1979.
[21] Xilinx, "Partial reconfiguration user guide UG702," vol.13.1, March 2011.
[22] H. Morishita, Y. Osana, N. Fujita, and H. Amano, "Exploiting memory hierarchy for a computational fluid dynamics accelerator on FPGAs," ICECE Technology, 2008. FPT 2008. International Conference on, pp.193–200, Dec. 2008.

Mohamad Sofian Abu Talip received B.E. degree from University of Science, Malaysia in 2005 and M.S. degree from International Islamic University Malaysia in 2010. He is currently a Ph.D. candidate at Keio University. His research interests include reconfigurable systems and parallel processing

Takayuki Akamine received the B.S. degree from Keio University in 2011. He is a master student in Keio University in the presence. He is interested in computer architecture, reconfigurable computing, and high performance computing.

Yasunori Osana received Ph.D. degree from the Department of Computer Science, Keio University, Japan in 2006. He is currently an Assistant Professor in the Department of Electrical and Electronics Engineering, University of the Ryukyus. His research interests include reconfigurable systems, computer architecture and its application in computational science.

Naoyuki Fujita received the M.E. degree from WASEDA university in 1992. In 1992, he joined National Aerospace Laboratory of Japan where he has been engaged in research on large-scale storage system, FPGA computer, network security. In 2000–2001, he joined U.S. IBM as a visiting scientist to work in HPSS project. He is currently working in Japan Aerospace Exploration Agency.

Hideharu Amano received the Ph.D. degree from Keio University, Japan in 1986. He is currently a Professor in the Department of Information and Computer Science, Keio University. His research interests include the area of parallel architectures and reconfigurable systems.