FPGA-Based Scalable Custom Computing Accelerator for Computational Fluid Dynamics Based on Lattice Boltzmann Method

Kentaro Sano

Abstract This paper presents a tightly-coupled FPGA cluster for custom computing of fluid dynamics simulation, and evaluates its performance with prototype implementation. For scalable and efficient computation with a lot of FPGA accelerators, we propose an accelerator-domain network (ADN) that brings low-latency and high-speed data transfer by directly connecting FPGAs. We describe implementation of a prototype cluster node with four FPGAs, and their on-chip framework for high-speed data streaming and computing. In performance evaluation, we demonstrate that our custom computing machine for fluid dynamics computation with the lattice-Boltzmann method (LBM) exploits both temporal and spatial parallelism, and scales the performance well with the number of FPGAs. As a result, we achieved 98.8 % of the peak performance of 73.0 GFlop/s with four FPGAs.

1 Introduction

Numerical simulations such as computational fluid dynamics (CFD) are now indispensable technology in engineering and industry because their lower-cost acquisition capability of various data than experiments, for example, tests with a wind tunnel. In order to meet the growing demand for high-performance computation (HPC) of large-scale and high-resolution simulations, accelerators such as GPUs have been getting more popular due to their computational performance per cost. GPU's throughput-oriented architecture, which is different from latency-oriented microprocessors, allows a lot of computing units to perform massively-parallel computation of thousands of threads, bringing very high peak-performance. Thus a single GPU can provide great acceleration if the computing units are sufficiently utilized, and systems with GPUs have been getting popular [1], so that they appear in Top500 list [2].

K. Sano (✉)
Graduate School of Information Sciences, Tohoku University, 6-6-01 Aramaki Aza Aoba, Aoba-ku, Sendai 980-8579, Japan
e-mail: kentah@caero.mech.tohoku.ac.jp

M.M. Resch et al. (eds.), *Sustained Simulation Performance 2014*,
DOI 10.1007/978-3-319-10626-7_16

However, it is not easy to obtain high sustained performance close to the peak in computing with a lot of GPUs. This is because throughput and latency of inter-GPU communication limit performance improvement with the number of GPUs, resulting in performance saturation. In a common GPU cluster, GPUs are not directly connected to each other. They are connected to PCI-Express interfaces of host nodes, and require indirect communication via the node's network to transfer data to other GPUs of different nodes suffering from a long latency and a narrow bandwidth, while its overhead can be somewhat reduced by such a technique as GPU direct. As a result, the scalability of a multi-GPU system is limited especially for bandwidth-intensive and/or latency-sensitive computations. This is especially conspicuous in the case of *strong-scaling*, where the problem size per GPU shrinks as the number of GPUs increases, because each GPU loses computation required to hide the communication latency.

For efficient acceleration, custom computing is another promising approach with good scalability for large-scale systems [3, 4]. Designs dedicated to an individual application allow computing resources in a system to be efficiently utilized, being well balanced with a bandwidth of a network and memories. Such tailor-made computing is a feasible solution with a suitable platform of field-programmable gate array (FPGA). FPGA is a circuit-reconfigurable semiconductor device, which has been getting advanced with a lot of logic elements, DSP blocks, on-chip memories, and high-speed I/O blocks. Not only for bit-level or integer computing, but also for floating-point computing, FPGA now has computing performance of several hundreds of GFlops with thousands of floating point units. Vendors announce that the next generation of 14 nm FPGAs will bring 10 TFlops per chip. In addition to this high peak floating-point performance, FPGAs are also expected to be advantageous for bandwidth and power consumption. Various hard macros of differential I/Os for DDR3 memory controllers, PCI-Express interfaces and high-speed transceivers are integrated onto state-of-the-art FPGAs, giving very wide bandwidth to external memories and inter-FPGA connections. Relatively low operating frequency and power-aware architectures make FPGAs a power-efficient solution.

These trends have motivated many researches on floating-point operations with FPGAs [5–8] and FPGA-based custom computing for scientific applications including Lattice Boltzmann method (LBM) [9–14]. LBM is one of the methods to compute fluids based on generalized cellular automata with numerical computation [15]. In addition to the regular and coarse parallelism of LBM, it can be implemented as stream computation which is performed with data flowing through a kernel function [9]. Since these features are suitable for custom computing, there have been several reports of FPGA-based LBM computation.

Sano et al. have developed an FPGA-based LBM accelerator [9] where stream computation is performed by dedicated pipelines, and evaluated its performance and power consumption [16]. However, streaming throughput between an FPGA and a CPU limits the performance because all the computational stages of LBM are not fully streamed. In their design, particle translation and boundary computation are

still performed by a CPU. Furthermore, parallel computing with multiple FPGAs was not considered.

Murtaza et al. reported LBM computation on a multi-FPGA system, called Maxwell [12]. Maxwell is a cluster of nodes, each of which has two FPGA boards connected via PCI interface. The FPGAs are directly connected to each other by Rocket I/O, but this is not used in their work. Each FPGA contains several processing elements (PEs) to compute a sub-lattice given by domain decomposition. Boundary data on an FPGA board are read to the host PC, communicated between adjacent PCs by message-passing, updated by each PC, and finally returned to the FPGA board for the next iteration. They showed that speedup does not increase sufficiently for more than 8 FPGAs because of the communication via the host network. In particular, the time for boundary-data transfer between a CPU and an FPGA has a big impact to the scalability in strong scaling where a problem size per FPGA shrinks due to domain decomposition. These results suggest that in order to scale the performance, custom computing with multiple FPGAs should take its own way to directly transfer data between FPGAs.

To this problem, our answer is a tightly-coupled FPGA-cluster where FPGAs are directly connected by an accelerator-domain network (ADN) with a low latency and a wide bandwidth. So far, we have proposed a parallel custom computing architecture for LBM (PCCAL), and showed its linear scalability with a performance model [13]. In this paper, we present:

1. Architecture and design of our tightly-coupled FPGA cluster,
2. Implementation of a prototype node of the cluster,
3. On-chip framework for high-speed data streaming and computing on FPGAs, and
4. Performance evaluation with custom computing of LBM on four FPGAs.

This paper is organized as follows. Section 2 presents our tightly-coupled FPGA cluster and the first prototype implementation. Section 3 describes custom machines for the lattice Boltzmann method to be computed on the cluster. Section 4 evaluates the performance and efficiency. Finally, section "Conclusions" gives conclusions and future work.

2 Tightly-Coupled FPGA Cluster for Scalable Custom Computing

2.1 Architecture of Tightly-Coupled FPGA Cluster

Figure 1 shows the architecture of a tightly-coupled FPGA cluster. The cluster is composed of computing nodes, each of which has multiple FPGA accelerators that are connected to the host node via PCI-Express interface. Each FPGA has its local DRAMs to store data for computation. For streaming data from memory to

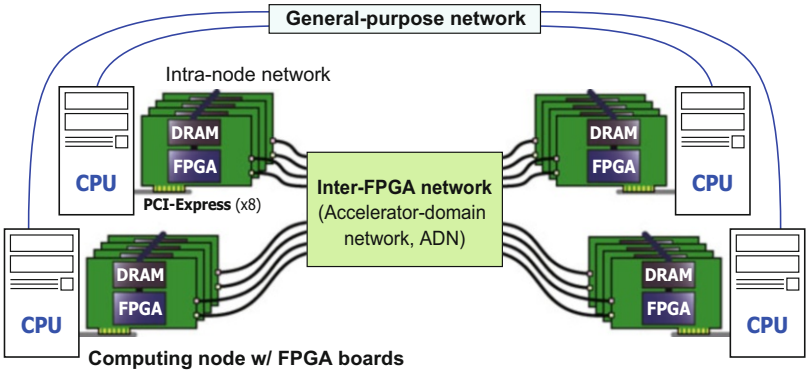

Fig. 1 Architecture of a tightly-coupled FPGA-cluster

memory, we assume two banks of memories so that read and write can be performed separately. The key feature of this architecture is the accelerator-domain network, called ADN, which directly connects FPGAs to each other. Although we can apply any network topology to ADN, here we made a choice of a 1D ring network for low latency communication, where each FPGA can transfer data directly to the adjacent FPGAs. This is because our primary application, LBM, requires local communication with adjacent FPGAs. If we need global communication among FPGAs, we can make a choice of an indirect network with switches, while FPGA-to-FPGA latency becomes relatively larger.

Recent high-end FPGAs have high-speed differential I/Os with a bandwidth higher than 10 Gbps, we can construct a high-throughput ADN for FPGAs which allows low-latency data transfer from a data-path to another data-path among FPGAs. In comparison with a conventional cluster with accelerators, we can increase the number of accelerators per node because we don't use the host-FPGA connection for inter-FPGA communication. Furthermore, the 1D ring network scales with the number of FPGAs because the we can ensure a constant bandwidth to each local connection between FPGAs.

2.2 Design and Implementation of a Cluster Node

In our research project, we have implemented a node of the tightly-coupled FPGA cluster with four DE4-230 FPGA boards designed by TERASIC Co., Ltd. [17]. Figure 2 shows a photograph of the board. The DE4-230 board has ALTERA Stratix IV EP4SGX230 FPGA [18], two DIMM slots of DDR2 memories, PCI-Express (PCIe) Gen 2.0 interface, and two high-speed mezzanine connectors (HSMCs). Stratix IV EP4SGX230 FPGA is a medium-sized device of the ALTERA's 40 nm-generation high-end FPGA series. It contains 182400 ALUTs (adaptive lookup

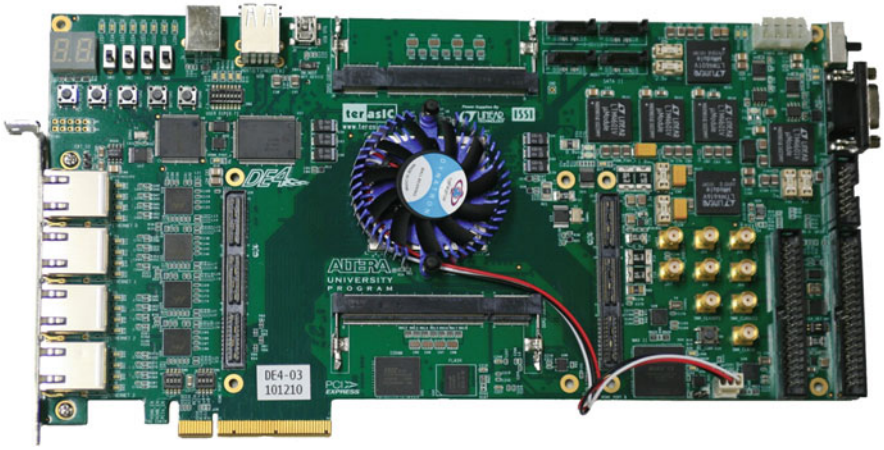

Fig. 2 Photograph of DE4-230 FPGA board [17]

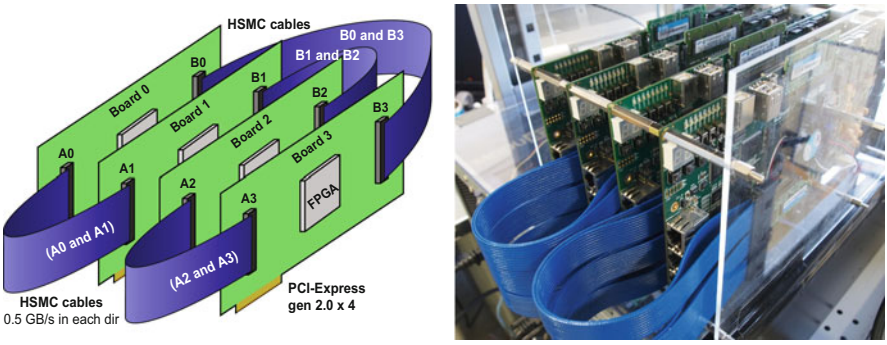

Fig. 3 Prototype cluster node with DE4-230 FPGA boards connected by a ring network

tables), block RAMS of 1828224 bytes in total, and 1288 18-bit DSP blocks, which is a hard macro of integer multiplier.

We connected four DE4-230 boards to a host PC by using PCIe Gen 2.0 with x4 lanes. We have developed a PCIe driver for Linux that provides high-speed data transfer between a CPU and one of the FPGAs, by driving a scatter-gather DMA (direct memory access) module on FPGA. As shown in Fig. 3, we made ADN of a ring network by directly connecting HSMCs of the FPGAs by their dedicated cables. We adopted single-end signaling for this connection while LVDS (low-voltage differential signaling) is also available. Single-end signaling at 125 MHz allows the cable to provide an unidirectional bandwidth of 500 MB/s at a latency of only six cycles.

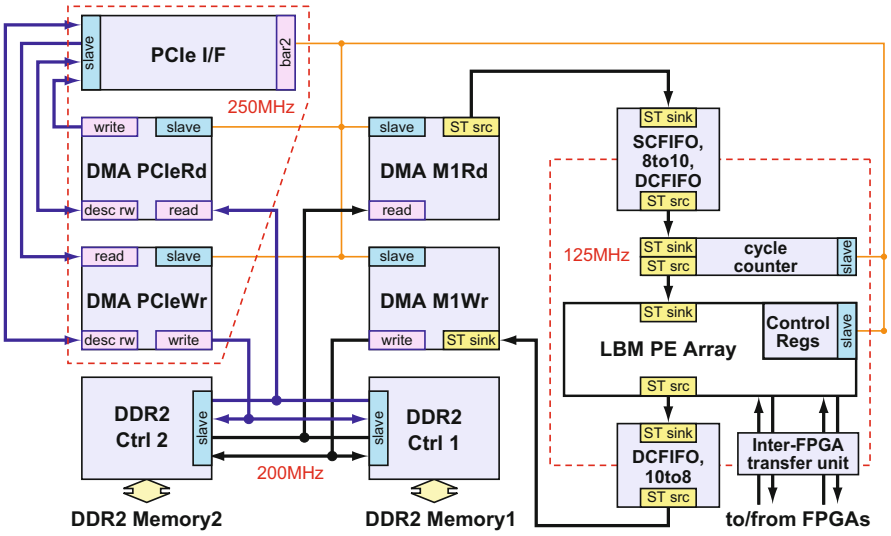

Fig. 4 On-chip system framework for high-speed data streaming

2.3 Acceleration Framework on FPGA

To achieve high-speed data transfer between CPU and FPGA, and a data stream between the DDR2 memories, we have designed and implemented an on-chip system framework with scatter-gather DMA (SGDMA) engines. Figure 4 shows the block diagram of the framework. We used ALTERA's system-on-programmable-chip development tool, Qsys. The modules in the framework are connected by using ALTERA's connection standards: Avalon-MM (memory mapped) and Avalon-ST (stream). The framework consists of the following modules: the PCIe interface, the two DDR2 memory controllers, the four SGDMA engines, the stream width conversion units with single-clock and dual-clock FIFOs, the cycle counter, and the user logic. Our custom computing machine for LBM is implemented with several control registers in this user logic. The PCIe, DDR2 memory controllers, SGDMA engines are available as IP cores in the Qsys module library. We have implemented the other modules by writing Verilog HDL-code so that the module interfaces satisfy the Avalon-MM or Avalon-ST.

To control the SGDMA engines, the cycle counters, and the LBM core, a program running on a host PC writes or reads control-and-status registers (CSRs) of these modules via the PCIe interface. Our Linux driver and API libraries provide these functions. For example, we can call a service function to kick SGDMA by writing its CSRs when we perform DMA transfer. The PCIe interface, the DDR2 memory controllers, and the user logic operate at different clock frequencies: 250, 200 and 125 MHz, respectively. This 125 MHz is a half clock frequency of the PCIe frequency. If we need other frequencies, we can configure PLL (phase-locked loop)

modules to obtain desired clock frequencies. To transfer data between different clock-domains, we need to insert clock-crossing bridges to avoid meta-stability.

3 Case Study: Custom Computing with Lattice Boltzmann Method

3.1 Lattice Boltzmann Method

The lattice Boltzmann method (LBM) [15–21] models fluids with propagation and collision processes of fictive particles over a discrete lattice mesh. In this paper, we explain the 2D orthogonal 9-speed (2D9V) model of the lattice Boltzmann method. In 2D9V model, each cell has a distribution function f_i as shown in Fig. 5 for each of the nine particle speeds $\mathbf{c}_i = (c_{x_i}, c_{y_i})$:

$$c_{x_i} = \cos\frac{2\pi(i-1)}{8}, \quad c_{y_i} = \sin\frac{2\pi(i-1)}{8} \tag{1}$$

where $0 \leq i \leq 8$. Here we assume that $\Delta t = \Delta x = \Delta y = 1$ for simplicity. The propagation and collision processes on the lattice mesh are defined by the following equation:

$$f_i^{\text{new}}(\mathbf{x} + \mathbf{c}_i, t+1) = f_i(\mathbf{x}, t) - \frac{f_i(\mathbf{x}, t) - f_i^{\text{eq}}(\mathbf{x}, t)}{\tau} \tag{2}$$

where $f_i^{\text{eq}}(\mathbf{x}, t)$ is an equilibrium distribution function of particles at $\mathbf{x}$ in a lattice at time step t, τ is a single relaxation time, and Δt is a time step for particles to move to the adjacent cells. The following stages solve Eq. (2):

1. MACRO : macroscopic physical quantity calculation stage
2. EQUI : equilibrium calculation stage

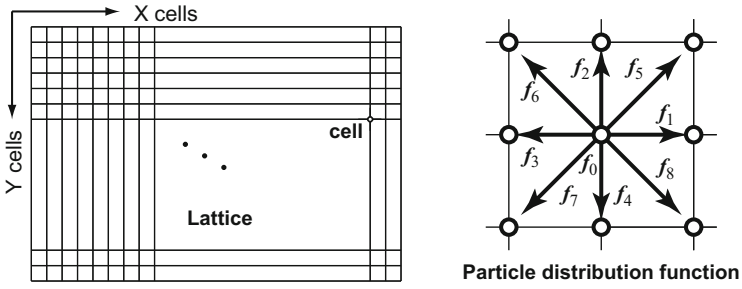

Fig. 5 2D lattice and particle-distribution functions at each cell for 2D9V model

3. COL : collision calculation stage
4. TRANS : translation stage
5. BOUND : boundary calculation stage

In MACRO stage, we compute the fluid density ρ and the fluid velocity $\mathbf{V} = (u, v)$ at $\mathbf{x}$ from the distribution functions $f_i(\mathbf{x}, t)$ by

$$\rho = \sum_i f_i(\mathbf{x}, t), \quad \rho\mathbf{V} = \sum_i \mathbf{c}_i \cdot f_i(\mathbf{x}, t). \tag{3}$$

In EQUI stage, we compute the equilibrium distribution function, $f_i^{\mathrm{eq}}(\mathbf{x}, t)$, by

$$f_i^{\mathrm{eq}} = \rho \left\{ A_i + B_i(\mathbf{c}_i \cdot \mathbf{V}) + C_i(\mathbf{c}_i \cdot \mathbf{V})^2 - D_i \mathbf{V}^2 \right\} \tag{4}$$

where A_i, B_i, C_i and D_i are constants varying according to $i = 0, 1, \ldots, 8$. In COL stage, we compute the right-hand side of Eq. (2). In TRANS stage, we propagate the particles by moving the right-hand side value of Eq. (2) at $\mathbf{x}$ to the adjacent cell at $(\mathbf{x} + \mathbf{c}_i)$ with its particle speed, $\mathbf{c}_i$.

In BOUND stage, we adjust boundary cells that require special treatment based on their conditions, e.g., for solid surfaces. We have three types of cells: fluid cells, solid cells, and outside cells. The solid-surface boundary cells, which are in contact with solid cells, are updated with the bounce-back condition [15]. The bounce-back condition simply reflects incoming particles on non-slip surfaces. For inlet/outlet boundary cells, which are in contact with outside cells, we apply a constant-pressure condition. In this condition, we calculate unknown f_i incoming to the cell with the other known f_i so that the fluid density ρ stays constant. By repeating MACRO to BOUND stages, we obtain time-dependent flow for a series of time steps.

3.2 Architecture for Stream Computation

We define stream computation for efficient custom computing of LBM:

Inputs:	$f_0, f_1, \ldots, f_8$
Kernel:	optimized Eqs. (3), (4), and (2)
Outputs:	$f_0^{\mathrm{new}}, f_1^{\mathrm{new}}, \ldots, f_8^{\mathrm{new}}$

where the input and output correspond to a data stream generated by traversing the lattice in the x-direction first. Please note that we optimize the computation of LBM to reduce the number of floating-point operations as described in [9, 13], So far we have proposed a parallel custom-computing architecture for LBM, PCCAL, to achieve efficient and scalable computing by using multiple FPGAs with ADN [13]. Figure 6 is the overview of PCCAL to be implemented on each FPGA, which is

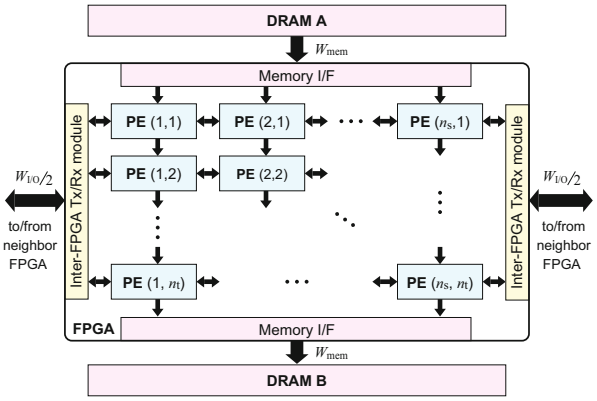

Fig. 6 Parallel computing architecture for LBM [13]

composed of the $n_s \times n_t$ PE array to exploit both spatial and temporal parallelism, the memory interfaces to stream data between external memories, and the inter-FPGA data transceiver/receiver modules to exchange the sub-lattice boundary data between the adjacent FPGAs.

We adopt 1D domain decomposition to use $(n_s N_{FPGA})$ PE columns on N_{FPGA} FPGAs for spatially parallel computation. The entire lattice of $X \times Y$ cells is partitioned into sub-lattices in x-direction. In computation, data of the sub-lattices are streamed from DRAM A and input to the columns of the PE array. In each column, the PE performs the stream computation for the allocated sub-lattice exchanging sub-lattice boundary data with the connected left and right PEs inside or outside the FPGA. Since the outputs of the first row of the PE array are considered as data streams for the next time-step, they are then input to the second row without consuming external memory bandwidth. We can increase operations executed per cycle with a constant stream-bandwidth by pipelining computations of successive time steps [14, 22]. Finally, the last row outputs the updated results to DRAM B. When streaming finishes, the streaming direction changes alternately.

3.3 PE Design for Fully-Streamed Computation

As reported in [23], we designed a processing element (PE) that performs all the stages of LBM with a stream of f_i and cell attribute. The cell attribute is a 32-bit word to give a cell information of fluid, solid, or boundary. Figure 7 shows a structure of a PE, which is composed of the following units: the MACRO, EQUI, and, COL unit (MECU) to compute cells with their floating-point pipelines, the delay unit (DU) to give delays to the cell attribute, the translation unit (TLU), and the boundary computing unit (BDU). The input stream of ten words is splitted into two streams of f_i and the cell attribute. Then, the stream of f_i is input to MECU.

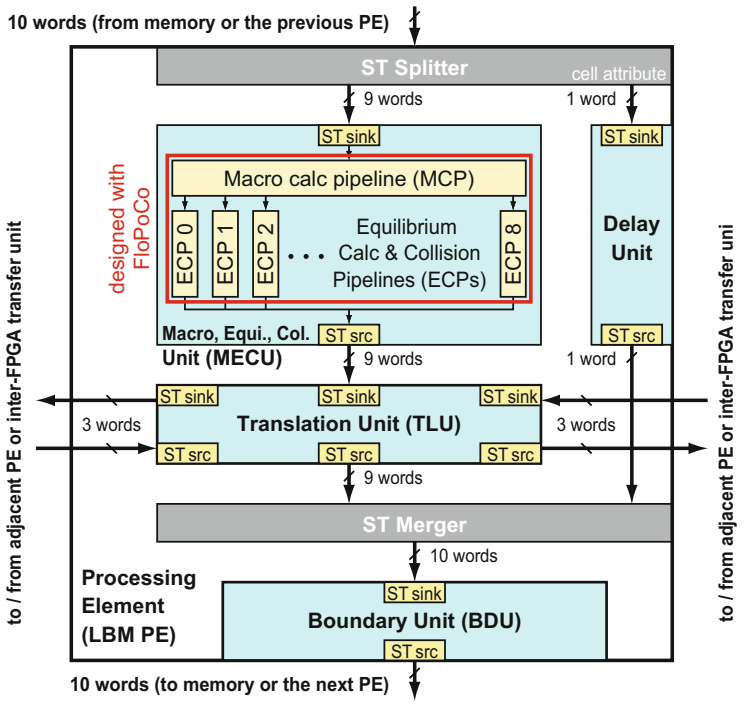

Fig. 7 Processing element [13]

MECU consists of the Macro calculation pipeline (MCP), and nine pipelines for equilibrium calculation and collision (ECPs) of f_i, respectively. MCP and ECPs have deep pipelines with a lot of single-precision operators: floating-point (FP) adders, multipliers, and a divider.

The output of MECU is streamed into TLU, and translation is performed by using shift registers as described in [13]. Moreover, TLU sends and receives data to and from the connected left and right PEs to exchange f_i with them when computing cells at sub-lattice boundaries. We design a synchronization mechanism for TLUs to control stall of the pipelines in a PE. The cell attribute from DU and the output of TLU are merged into a single stream, and then input to BDU. BDU updates cells for the bounce back condition and the constant pressure condition.

4 Performance Evaluation

4.1 Implementation of PEs

For performance evaluation, we have implemented a single PE or two cascaded PEs on each FPGA. The PE operates at 125 MHz. TLU was implemented with multi-tap

shift registers for computing an $X \times Y$ lattice with $X = 360, 720, 1{,}440,$ *or* $2{,}880$. To execute LBM computation on the array, we use the SGDMA engine for PCIe to transfer the initial lattice data to DDR2 memory1 of FPGA. Then, we activate the other two SGDMA engines to stream the data between the DDR2 memories. When the stream computation finishes and the result is written into the DDR2 memory2, then we start stream computation in an opposite direction, i.e., from the DDR2 memory2 to memory1. By repeating this ping-pong streaming, the on-chip system can efficiently perform the LBM computation without accessing a host. If necessary, temporal results can be read out by the host.

4.2 Resource Consumption

The resource consumption of the system implemented with two PEs on one FPGA is shown in Table 1. The entire system consumes 75.4 % ALUTs (Adaptive lookup tables), 65.6 % registers, 38.0 % BRAM (block-RAM) bits, and 46.0 % 18-bit DSP blocks of those on this FPGA. The framework except the PEs consumes 20.8 % ALUTs, 22.7 % registers, 19.2 % BRAM bits, and no DSP block. The remaining resources are consumed by PEs 1 and 2. Each of PEs 1 and 2 consumes about 18 % ALUTs, 20 % registers, 2 % block-RAM bits, and 23 % DSP blocks. Although almost half of the DSP blocks are not utilized yet, a system with more than two PEs did not fit the FPGA due to ALUT and register consumption. We need to optimize resource consumption in order to implement more PEs. With the remaining BRAM bits, we estimate that the two PEs on this FPGA can compute a three-times larger lattice with $8{,}640 \times Y$ cells.

Table 1 Resource consumption of the implemented system

	ALUTs	Registers	BRAM [Kbits]	18-bit DSPs
Stratix IV **E**P4SGX230	182400	182400	14283	1288
Entire system	137446 (75.4 %)	119621 (65.6 %)	5433.8 (38.0 %)	592 (46.0 %)
Framework	37942 (20.8 %)	41316 (22.7 %)	2748.4 (19.2 %)	0 (0.0 %)
LBM PE array (PE x 2)	995046 (54.6 %)	78305 (42.9 %)	2685.4 (18.8 %)	592 (46.0 %)
PE 1	46889 (25.7 %)	38117 (20.9 %)	1272.4 (8.9 %)	296 (23.0 %)
PE 2	47403 (26.0 %)	38463 (21.1 %)	1404.9 (9.8 %)	296 (23.0 %)

4.3 Computational Performance

We calculate the peak Flop/s performance by multiplying the number of operators and an operational frequency. MECU contains 72 FP adders, 63 FP multipliers, and one FP divider, which operate every cycle at 125 MHz when valid data are streamed. BDU contains 7 FP adders and 3 FP multipliers. Therefore, we obtain the peak performance of a single PE, P_{PE}, as

$$P_{\mathrm{PE}} = (72 + 63 + 1 + 7 + 3) \times 0.125 = 18.25 \ \text{GFlop/s}. \tag{5}$$

The peak performance of two PEs is $2 \times P_{\mathrm{PE}} = 36.5$ GFlop/s. However, all the peak performance is not always achieved due to several overheads. In this system, the setup time for starting SGDMA, stall cycles in streaming data, and pipeline epilogue and prologue can degrade the sustained performance. We measured the time between the start and the end of SGDMA transfer by using an inline assembly code with RDTSC instruction to obtain cycle counts of x86 CPUs. By multiplying the difference in a count number with a cycle period of a CPU, we can obtain the elapsed time at high accuracy. The obtained time includes both the net computing time and the overhead.

For evaluation, we computed flow in a 2D sudden expansion channel with a solid circle of Fig. 8, by using one, two, or four FPGAs connected with an HSMC cable. We changed the lattice size from 360×120 to $5{,}760 \times 1{,}920$ in order to evaluate the pipelining overhead of a PE array. We used only 1 PE implementation for the two- and four- FPGA case because the inter-FPGA Tx/Rx module was not available yet for data exchange of two cells.

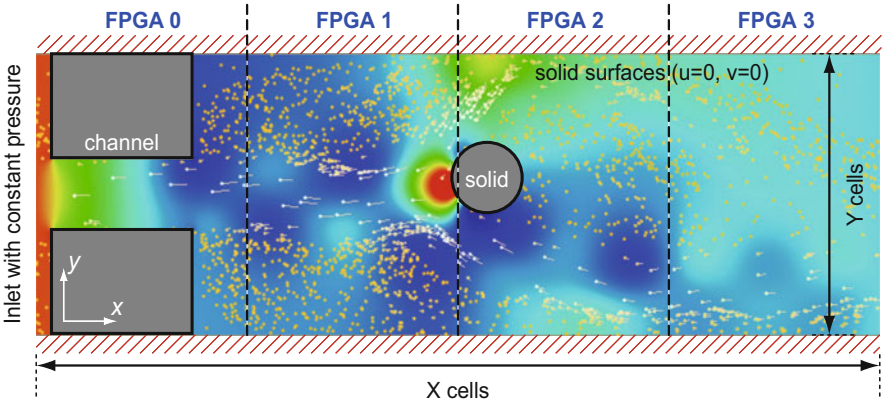

Fig. 8 Computational condition of 2D sudden-expansion channel flow with a *solid circle*. The color shows pressure, and the particles show fluid velocities. *Red* and *blue* are for high and low pressures, respectively. A Karman vortex street is made after the solid. The lattice is partitioned into four parts to be computed in parallel by four FPGAs.

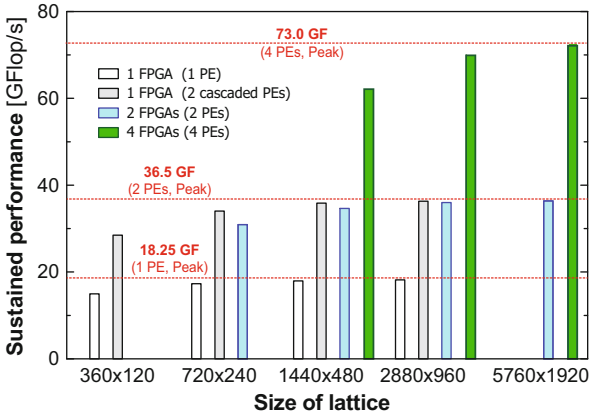

Fig. 9 Computing performance for different lattice sizes for one to four FPGAs

Figure 9 shows the computing performance of a single FPGA with one or two PEs, and two or four FPGAs with one PE each. Each bar shows GFlop/s performances obtained with measured time including the SGDMA kickoff overhead and the pipelining overhead. We observed that stall cycles are less than 1.0 % of all the cycles in almost all cases. The white bar shows the sustained performance for a single PE on one FPGA. The performance for the 360×120 lattice is 15.0 GFlop/s, which 82 % of the peak performance. This performance degradation is because of the SGDMA kickoff overhead and the pipelining overhead. As the lattice gets larger, the sustained performance increases as the overhead relatively shrinks. As a result, 18.2 GFlop/s, which is 99.7 %, was achieved for the $2{,}880 \times 960$ lattice. The similar trend is also seen in the other cases.

The light gray bar shows the sustained performance of two cascaded PEs on one FPGA. Since the depth of the pipeline is doubled in comparison with a single PE, the overhead appears larger for the 360×120 lattice. However, finally 36.3 GFlop/s, 99.5 % of the peak, was achieved with the $2{,}880 \times 960$ lattice. This demonstrates scalability by cascading PEs on an FPGA.

The light blue and green bars show the sustained performance of two FPGAs and four FPGAs with one PE on each FPGA, respectively. Since each FPGA computes a sub-lattice that is smaller than the entire lattice, the overhead is seen larger especially for a small lattice. Similarly to the other cases, increasing lattice size makes efficiency higher. The $2{,}880 \times 960$ and $5{,}760 \times 1{,}920$ lattices allow the systems to achieve 36.38 GFlop/s (99.7 % of the peak) and 72.13 GFlop/s (98.8 % of the peak), respectively. These results show that our proposed architecture has scalability with multiple FPGAs. Thus, by cascading PEs and increasing the number of FPGAs, we can exploit both the temporal and spatial parallelism to scale the performance of LBM computation with our proposed custom computing architecture for LBM.

Conclusions

This paper presents the architecture and design of the tightly-coupled FPGA cluster for custom computing of fluid dynamics simulation, and evaluates its performance with prototype implementation. The key feature of the cluster is the accelerator-domain network (ADN) that brings low-latency and high-speed direct data transfer between FPGAs. We preserve the implementation of the cluster node with four ALTERA Stratix IV FPGAs, and the on-chip framework for high-speed data streaming and computing on them. We describe the custom computing machine for fluid dynamics computation with the lattice-Boltzmann method (LBM). In performance evaluation, we demonstrate that the machine exploits both temporal and spatial parallelism of LBM computation, and scales well with the number of FPGAs. Finally, the four processing elements on four FPGAs achieves 98.8 % of the peak performance of 73.0 GFlop/s.

We are now building a larger FPGA cluster with 28 nm generation FPGAs and 10G Ethernet switch. In future work, we implement a custom computing machine of LBM for the cluster, and evaluate its performance and power consumption. We will also develop a high-level synthesis compiler to easily generate custom computing machines for stream computation on FPGAs.

Acknowledgements This research was partially supported by Grant-in-Aid for Scientific Research (B) No. 23300012 and Grant-in-Aid for Challenging Exploratory Research No. 23650021 from the Ministry of Education, Culture, Sports, Science and Technology, Japan. We thank the support by ALTERA university program.

References

1. Takizawa, H., Yamada, N., Sakai, S., Kobayashi, H.: Radiative heat transfer simulation using programmable graphics hardware. In: Proceedings of the 5th IEEE/ACIS International Conference on Computer and Information Science, pp. 29–36 (2006)
2. TOP500. http://www.top500.org (2014)
3. Makino, J.: The GRAPE project. Comput. Sci. Eng. **8**(1), 30–40 (2006)
4. Shaw David E., et al. Anton, a special-purpose machine for molecular dynamics simulation. Commun. ACM **51**(7), 91–97 (2008)
5. Shirazi, N., Walters, A., Athanas, P.: Quantitative analysis of floating point arithmetic on FPGA based custom computing machines. In: Proceedings of the IEEE Symposium on FPGA's for Custom Computing Machines, pp. 155–162 (1995)
6. deLorimier, M., DeHon, A.: Floating-point sparse matrix-vector multiply for FPGAs. In: Proceedings of the International Symposium on Field-Programmable Gate Arrays, pp. 75–85 (2005)
7. Zhuo, L., Prasanna, V.K.: Sparse matrix-vector multiplication on FPGAs. In: Proceedings of the International Symposium on Field-Programmable Gate Arrays, pp. 63–74 (2005)
8. Dou, Y., Vassiliadis, S., Kuzmanov, G.K., Gaydadjiev, G.N.: 64-bit floating-point FPGA matrix multiplication. In: Proceedings of the International Symposium on Field-Programmable Gate

Arrays, pp. 86–95 (2005)
9. Sano, K., Pell, O., Luk, W., Yamamoto, S.: FPGA-based streaming computation for lattice boltzmann method. In: Proceedings of the International Conference on Field-Programmable Technology, pp. 233–236 (2007)
10. Murtaza, S., Hoekstra, A.G., Sloot P.M.: Compute bound and I/O bound cellular automata simulations on FPGA logic. ACM Trans. Reconfigurable Technol. Syst. **1**(4) (2009) Article 23.
11. Sano, K., Luzhou, W., Hatsuda, Y., Iizuka, T., Yamamoto, S.: FPGA-array with bandwidth-reduction mechanism for scalable and power-efficient numerical simulations based on finite difference methods. ACM Trans. Reconfigurable Technol. Syst. **3**(4) (2010)
12. Murtaza, S., Hoekstra, A.G., Sloot, P.M.: Cellular automata simulation on a FPGA cluster. Int. J. High Perform. Comput. Appl. **25**(2), 193–204 (2011)
13. Kono, Y., Sano, K., Yamamoto, S.: Scalability analysis of tightly-coupled FPGA-cluster for lattice Boltzmann computation. In: Proceedings of the 22nd International Conference on Field-Programmable Logic and Applications, August 2012. Paper W3B1.
14. Sano, K., Hatsuda, Y., Yamamoto, S.: Multi-FPGA accelerator for scalable stencil computation with constant memory-bandwidth. IEEE Trans. Parallel Distrib. Syst. **25**(3), 695–705 (2014)
15. Skordos, P.A.: Initial and boundary conditions for the lattice Boltzmann method. Phys. Rev. E **48**(6), 4823–4842 (1993)
16. Sano, K., Nishikawa, T., Aoki, T., Yamamoto, S.: Evaluating power and energy consumption of FPGA-based custom computing machines for scientific floating-point computation. In: Proceedings of the International Conference on Field-Programmable Technology, pp. 301–304 (2008)
17. Terasic Technologies. http://www.terasic.com (2013)
18. Altera Corporation. http://www.altera.com/literature/ (2013)
19. Inamuro, T., Yoshino, M., Ogino, F.: A non-slip boundary condition for lattice Boltzmann simulations. Phys. Fluids **7**(12), 2928–2930 (1995)
20. Noble, D.R., Chen, S., Georgiadis, J.G., Buckius, R.O.: A consistent hydrodynamic boundary condition for the lattice Boltzmann method. Phys. Fluids **7**(1), 203–209 (1995)
21. Chen, S., Wang, Z., Shan, X., Doolen, G.D.: Lattice Boltzmann computational fluid dynamics in three dimensions. J. Stat. Phys. **68**(3–4), 379–400 (1992)
22. Kobori, T., Maruyama, T., Hoshino, T.: A celluer automata system with FPGA. In: Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines, pp. 120–129 (2001)
23. Sano, K., Kono, Y., Suzuki, H., Chiba, R., Ito, R., Koizumi, K., Yamamoto, S.: Efficient custom computing of fully-streamed lattice Boltzmann method on tightly-coupled FPGA cluster. ACM SIGARCH Computer Architecture News, **41**(5), 47–52 (2013)