FPGA-based Stream Computing for High-Performance N-Body Simulation using Floating-Point DSP Blocks

Kentaro Sano*, Shin Abiko, Tomohiro Ueno

Tohoku University
6-6-01 Aramaki Aza Aoba, Sendai 980-8579, JAPAN

ABSTRACT

Recent advancement of FPGAs allows high-performance and low-power computing by constructing deeply-pipelined custom hardware using floating-point DSP blocks. In this paper, we present a stream-computing architecture and design for FPGA-based high-performance N-body simulation, which is different from the parallel-computing-and-reduction approach of the GRAPE systems, which are predecessors of custom N-body machines. The proposed architecture is composed of a force-pipeline module (FPM) and an integral-pipeline module (IPM). FPM has a scalable structure based on n cascade-connected pairs of computing elements (CEs) and streamed register files (SRFs) so that we can scale the performance by increasing n. We also present the performance model. The measure performance of the system prototyped with a single Arria10 FPGA has good agreement with the model, and scales well with n at a higher efficiency when the problem size is large. We demonstrate that the system with $n = 64$ CEs operating at 180 MHz achieves 10944 MFCPS (million force calculation per second) for $N = 262144$ particles.

Keywords

N-body simulation, stream computing, high-performance computation, custom computing, Arria10 FPGA, floating-point DSP blocks

1. INTRODUCTION

The N-body simulation is one of the important techniques in scientific fields including astrophysics and molecular dynamics. In particular, the astrophysical N-body simulation is used as a powerful tool to study astronomical objects such as the solar system, star clusters, galaxies, clusters of galaxies, and large-scale structures of the universe [1, 2]. In the simulation, we numerically solve the equation of motions for N particles interacting each other by gravitational force. During the simulation, most of computing time is spent to evaluate the gravitational force between particles. Since the complexity is $O(N^2)$, the N-body simulation is known as a computationally intensive problem. For the same computing time, parallel computation with m times more processors allows only a $\sqrt{m}$ times larger problem size.

The computational intensity has motivated development of more efficient and faster machines relying on hardware structures dedicated to the N-body problem. The most famous project targeting special-purpose N-body machines is the GRAPE project [2], which has developed GRAPE-4 [3], GRAPE-5 [4], GRAPE-6 [5], and GRAPE-DR [6]. Here, GRAPE means "Gravity Pipe." The customized architectures and circuits of these GRAPE machines allowed the systems to overcome supercomputers.

However, the project faced the problem of cost and time for development [2]. As the semiconductor technology scales, the NRE (non-recursive engineering) cost increases drastically for making masks. Therefore, in terms of cost/performance, it gets unreasonable to fabricate custom VLSI chips for special-purpose systems because the small production makes price per chip very expensive. In addition, huge circuits with state-of-the-art technologies including high-speed memory systems and transceiver links become very difficult to design, implement, and verify, requiring very high skills and a long development period for engineering. Thus, when a new custom chip is launched, it can lose its best time if faster general-purpose processors have already appeared.

To the contrary, FPGAs (Field-Programmable Gate Arrays) have been getting more attractive and feasible to implement high-performance special-purpose hardware instead of custom VLSIs. Nowadays, FPGAs are fabricated using the cutting-edge semiconductor technology such as 14nm, resulting in huge, high-speed, and power-efficient devices regardless of their reconfigurable structures. Furthermore, advancement in FPGA architecture gives FPGAs capability of high-performance computing with floating-point (FP) numbers. Recently, Intel released Generation10 FPGAs such as Arria10 and Stratix10 FPGAs, which have hardened FP-DSP blocks [7, 8]. The peak FP performance of Stratix10 FPGA reaches 10 TFlops, which is comparable to that of the latest GPUs. There have also been reports saying that custom machines implemented with Arria10 FPGA achieved both better performance and higher performance per power than GPUs [9].

However, it is still challenging how to map target algorithms onto the fabrics on FPGAs. In our previous research, we have discovered that stream computation is suitable to bring out potential performance of FPGAs because high-throughput computation is available with a deeply-pipelined hardware structure even if an external memory bandwidth is limited [10]. Although this stream-computing approach is different from that of the GRAPE system especially in the parallel-computing-and-reduction approach with multiple chips, it is promising and feasible due to its simplicity and potential scalability in deepening a pipeline.

Our final goal is to develop a scalable N-body simulation system with multiple FPGAs in a tightly-coupled FPGA cluster [11, 12]. Toward the goal, this paper presents an stream-computing architecture and design for high-performance N-body simulation on a single FPGA. With the architecture, we fully stream the N-body computation including the force calculation and the time integral with particles accelerations. It consists of a force pipeline module (FPM) and an integral pipeline module (IPM). FPM is composed of n computing elements (CEs) to calculate the gravitational force and streamed register-files (SRFs) which store the data of counterparts to be used in force computation. By increasing n to deepen the pipeline, we can increase the performance of force calculation.

*Corresponding author. Email:kentah@caero.mech.tohoku.ac.jp

This work was presented in part at the international symposium on Highly-Efficient Accelerators and Reconfigurable Technologies (HEART2017) Bochum, DE, June 7-9, 2017.

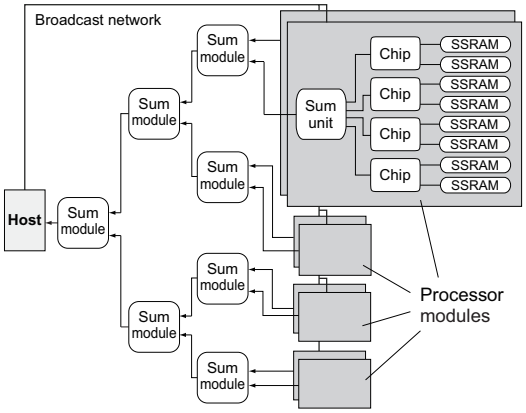

Figure 1: Reduction-tree network and broadcast network for multiple chips of GRAPE-6 [5].

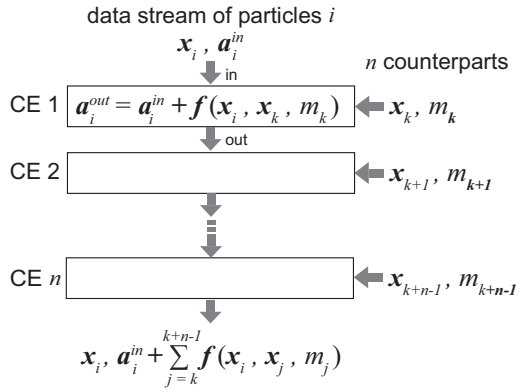

Figure 2: Stream-computing of forces with n counterparts.

We present a model to estimate the performance in computing N particles with n CEs. We implement the proposed hardware by using FP-DSPs on a single Intel Arria10 FPGA to demonstrate the N-body simulation capability, validate the performance model, and evaluate the scalability. Contributions of this work are:

1. fully-streamed architecture with SRFs for N-body simulation targeting multiple FPGAs,
2. hardware design of the system with FPM and IPM,
3. performance model, and
4. implementation and evaluation with a single Arria10 FPGA.

This paper is organized as follows. Section 2 describes related work. Section 3 gives the basics of the N-body problem, its streamed computation, and the hardware design. Section 4 presents prototype implementation, evaluation, and discussion of the proposed hardware for performance and scalability. Finally, Section 5 gives conclusions and future work.

2. RELATED WORK

So far, various ASIC (application-specific integrated circuit) chips and systems for astrophysical N-body simulation have been developed in the GRAPE project [2], which include GRAPE-4 [3], GRAPE-5 [4], GRAPE-6 [5], and GRAPE-DR [6]. GRAPE-4 to 6 are based on special-purpose VLSIs which can perform fixed computation with dedicated data-paths while GRAPE-DR is of programmable chips to be used for various N-body simulations.

Fig.1 shows the multi-chip system of the GRAPE-6 [5]. Each processor module contains four GRAPE-6 chips, which are connected to the sum unit. The sum unit sums up the partial forces computed by the chips as described in Eq.(8). The outputs of the processor modules are sent to the reduction tree network with the sum modules, which are summed up to obtain the final forces and send them to the host computer. The host computer updates the positions and velocities of particles with the received forces.

Thus, the architectural feature of the GRAPE system is 1) the parallel computation with multiple chips and 2) the reduction of the partial results by the tree network. Since the entire set of particle data is distributed among the chips, the updated data need to be re-distributed to all the SSRAMs of the chips. To efficiently execute the distribution of data, the GRAPE system also has a broadcast network as shown in Fig.1. The computation of GRAPE-6 system has the following stages; parallel computation with multiple chips, reduction of the results, updates by the host, and re-distribution of updated data. The transition from each stage to the next has significant overhead resulting in low utilization of chips while the stages can be somehow pipelined at coarse granularity.

On the other hand, our fully-streamed approach makes the scalable computation easier. As shown in Figs.4 and 5, for force calculations and position updates we just stream the particle data through the computing modules in the cascaded FPGAs. Since the updated data always exist only in the DRAMs of the Master FPGA, we don't have to re-distribute it to other DRAMs. Moreover, since all the computation is performed in the pipeline, we can achieve high efficiency making a sustained performance close to the peak performance. Furthermore, the host computer is just required to control the FPGAs. There are no data movement between the host and the Master FPGA except the initial upload of the data.

Thus, we expect that, to a certain extent of a system size, our fully-streamed approach is superior to the parallel-computing-and-reduction approach of the GRAPE system. In this paper, we investigate how scalable the stream-computing can be when we deepen the pipeline.

3. STREAM COMPUTING OF N-BODY SIMULATION

3.1 N-body problem and gravitational force

N-body problem is a problem to solve the equation of motions for N particles interacting each other with some forces. Let $\boldsymbol{x}_i$ and m_i denote the position vector and the mass of particle i, respectively. The force $\boldsymbol{F}_i$ acting on particle i is obtained by:

$$\boldsymbol{F}_i = \sum_{j \neq i} \boldsymbol{F}_{ij}, \tag{1}$$

where $\boldsymbol{F}_{ij}$ is the force given by particle j to particle i. In the case of astrophysical problems, particles interact each other gravitationally with the underlying dynamics of Newton's law. Here

$$\boldsymbol{F}_{ij} = \boldsymbol{F}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \frac{G m_i m_j \boldsymbol{r}_{ij}}{r_{ij}^3}, \tag{2}$$

where G is the gravitational constant, and

$$\boldsymbol{r}_{ij} = \boldsymbol{x}_j - \boldsymbol{x}_i, \tag{3}$$

$$r_{ij} = |\boldsymbol{r}_{ij}|. \tag{4}$$

To solve the equation of motion, we calculate the time integral with gravitational acceleration $\boldsymbol{a}_i$ of particle i.

$$\boldsymbol{a}_i = \frac{\boldsymbol{F}_i}{m_i} = \sum_{j \neq i} \frac{\boldsymbol{F}_{ij}}{m_i} = \sum_{j \neq i} \frac{G m_j \boldsymbol{r}_{ij}}{r_{ij}^3}. \tag{5}$$

If we adopt the Euler method for the time integral, we can simply update the velocity $\boldsymbol{v}_i$ and position $\boldsymbol{x}_i$ of particle i for time-step difference Δt as:

$$\boldsymbol{v}_i^{n+1} = \boldsymbol{v}_i^n + \Delta t \, \boldsymbol{a}_i, \tag{6}$$

$$\boldsymbol{x}_i^{n+1} = \boldsymbol{x}_i^n + \Delta t \, \boldsymbol{v}_i. \tag{7}$$

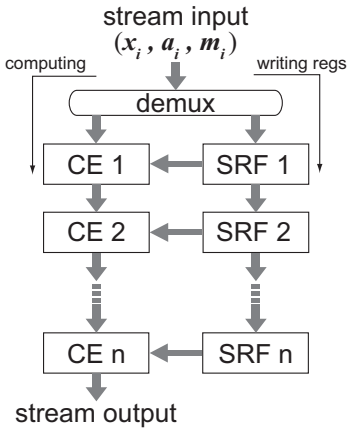

Figure 3: Structure of a force pipeline module (FPM).

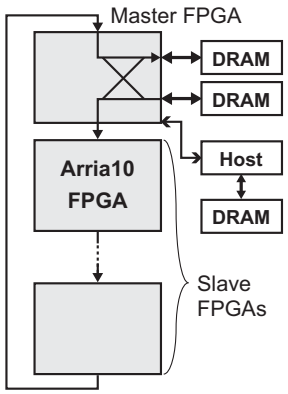

Figure 4: Stream computing with multiple chips (future work).

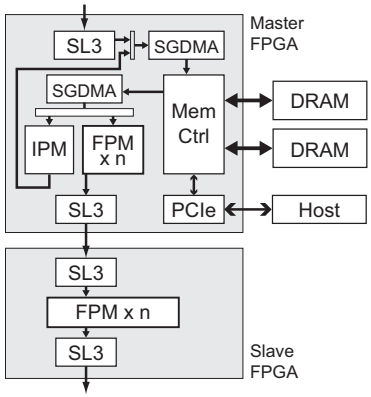

Figure 5: Future designs for master and slave FPGAs. IPM and FPM denote an integral pipeline module and a force pipeline module, respectively.

Although better methods for the force calculation and the time integral than the above have been proposed [1], we use Eqs.(5) to (7) for simplicity in architectural exploitation.

3.2 Stream computing of N-body simulation

Stream computing is an approach to sequentially read and compute data elements in a data stream from an external memory. The regularity and continuity in reading data allow us to fully exploit the peak memory bandwidth. Moreover, we can overcome high latency and limited bandwidth by deeply pipelining the computation. When we can additionally cascade more hardware modules to compute the elements in a data stream, we can perform more operations without increasing memory-bandwidth requirement.

In this paper, we design a stream-computing algorithm for computing partial sum of the summation in Eq.(5).

$$\boldsymbol{a}_i^{k,k+n-1} = \sum_{j=k}^{k+n-1} \frac{Gm_j \boldsymbol{r}_{ij}}{r_{ij}^3}, \tag{8}$$

where $\boldsymbol{a}_i^{k,k+n-1}$ means the partial acceleration for particle i given from n particles; k to $(k+n-1)$. For stream computing of Eq.(8), we cascade n computing elements (CEs), each of which computes

$$\boldsymbol{a}_i^{\text{out}} = \boldsymbol{a}_i^{\text{in}} + \frac{Gm_j \boldsymbol{r}_{ij}}{r_{ij}^3} = \boldsymbol{a}_i^{\text{in}} + \boldsymbol{f}(\boldsymbol{x}_i, \boldsymbol{x}_j, m_j). \tag{9}$$

Fig.2 shows the stream-computing of the partial sum with the n counterparts. By repeating Eq.(9) with n CEs, we obtain the results of Eq.(8). With the cascaded CEs as shown in Fig.2, we can perform stream-computing to obtain partial sum for all the particles.

To completely calculate the summation in Eq.(5), we need to further repeat Eq.(8) for N/n times, where N is the total number of particles. In each repetition, we have to replace the counterparts for particles k to $(k+n-1)$. For example, in the first stream-computing, we set counterparts of 0 to $(n-1)$ to obtain $\boldsymbol{a}_i^{0,n-1}$ for all the particles. Then, we replace the counterparts with ones of n to $(2n-1)$ to obtain $\boldsymbol{a}_i^{0,2n-1}$.

3.3 Hardware design for stream computing

To achieve the stream computing with counterpart replacement, we design the force pipeline module (FPM) shown in Fig.3. FPM is composed of n CEs and n streamed register-files (SRFs). Each CE computes Eq.(9) with a pipelined data-path. Each SRF has the data of each counterpart which are provided to the corresponding CE. In order to increase the utilization of CEs for computing, the time to replace the counterpart data should be made as short as possible. In this research, we make the registers chained so that they can be written at a time with a data stream. We use the demultiplexer (demux) in Fig.3 to switch computing with CEs and data replacement for SRFs. When we supply a data stream for the chained SRFs, the data are written into the SRFs as they are shifted through the registers. Once necessary data are written into the SRFs, we input the data stream into the cascaded CEs.

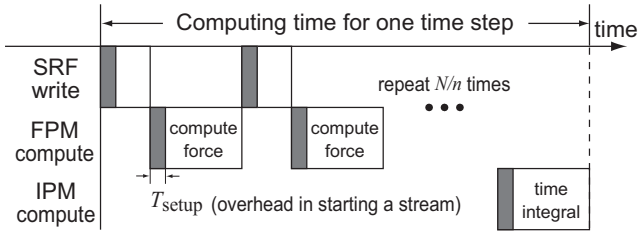

Figure 6: Time-space diagram of computing behavior.

This stream-computing approach is also advantageous in scaling the system with multiple FPGAs as shown in Fig.4, which is our target in the future work. The system consists of one "Master" FPGA and "Slave" FPGAs. Only the Master FPGA has the external memories to store a data stream, and connection to the host machine for system control. The Slave FPGAs are implemented only with stream-computing pipelines. Note that particle data can easily be written into the SRFs in Slave FPGAs. We can just stream data for the SRFs without using an extra broadcast network.

Fig.5 shows the block diagrams of the Master FPGA and the Slave FPGA. In the Master FPGA, we have the PCI-Express interface (PCIe), the memory controller for DRAMs, the scatter-gather DMAs (SGDMAs), serialite-III IP core (SL3) for high-speed transceiver link between FPGAs, and the computing modules of FPM and IPM. Here, IPM denotes an integral pipeline module, which computes the time integration of Eqs.(6) and (7). After the accelerations of Eq.(5) are completely obtained for all the particles, we perform stream computing of the time integration with IPM to update the positions of the particles.

In the Slave FPGA, we have SL3 and FPM. By cascading more Slave FPGAs to increase n, we can reduce the number of repetitions, N/n, to speedup the entire computation.

3.4 Performance model

We model the performance of stream-computing for the force computation and the time integral. The variables used in the

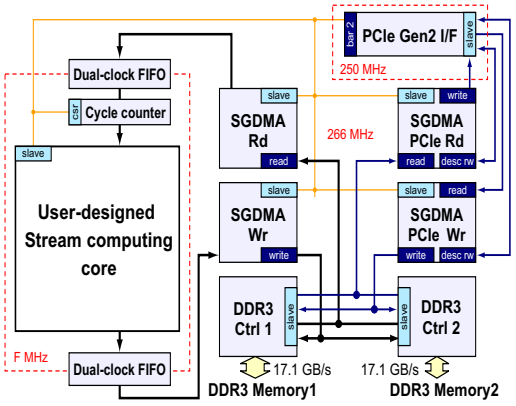

Figure 7: Acceleration system platform on FPGA.

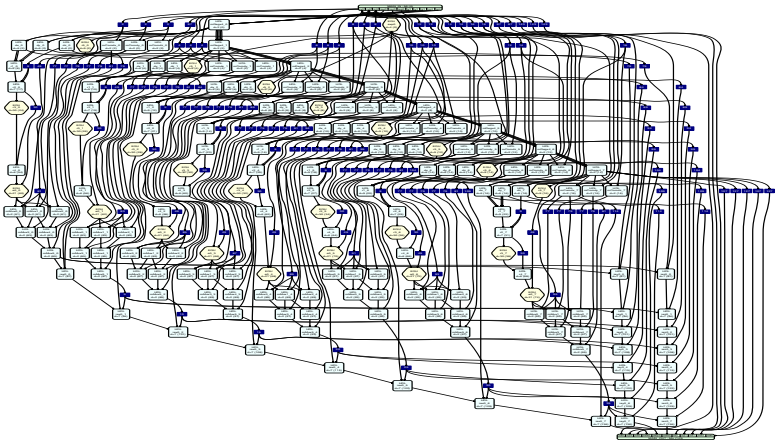

Figure 8: DFG of FPM implemented with eight CEs.

Table 1: Variables for the performance model.

Variable	description	used value
N	the number of particles	-
n	the total number of CEs	-
F	operating frequency	180 MHz
T_{setup}	overhead time for streaming data	3.64×10^{-5} s
D_{8CEs}	delay cycles of 8 CEs	143
D_{IPM}	delay cycles of IPM	14

model are summarized in Table 1. Here, we use the number of delay cycles for the FPM with 8 CEs, D_{8CEs}, which is a unit of implementation. Please note that the number of delay cycles for SRFs is equal to that for CEs because of our implementation where SRFs are connected to CEs at the same pipeline depth.

Fig.6 shows the behavior of the modules in the time-space diagram. We repeat streaming for the SRF write and the FPM computation for N/n times. After this, we perform streaming for the time integral once.

The time for a single stream to write n SRFs is obtained as:

$$T_{\text{SRF}} = T_{\text{setup}} + \left(n + \frac{n}{8} D_{\text{8CEs}}\right) \frac{1}{F}, \tag{10}$$

where n data elements are written into the SRFs through the pipeline with a depth of $\frac{n}{8} D_{\text{8CEs}}$ cycles. T_{setup} is an overhead time in starting each stream. Similarly, the time for a single stream to compute with n CEs is:

$$T_{\text{FPM}} = T_{\text{setup}} + \left(N + \frac{n}{8} D_{\text{8CEs}}\right) \frac{1}{F}, \tag{11}$$

where N data elements are streamed through the FPM. The time for a single stream to compute with IPM is given:

$$T_{\text{IPM}} = T_{\text{setup}} + (N + D_{\text{IPM}}) \frac{1}{F}. \tag{12}$$

Considering N/n repetitions of streaming for SRF and FPM, the total time for a single time-step is obtained:

$$\begin{aligned} T(N,n) &= \frac{N}{n}\left(T_{\text{SRF}} + T_{\text{FPM}}\right) + T_{\text{IPM}} && (13)\\ &\simeq \frac{N}{n}\left(T_{\text{SRF}} + T_{\text{FPM}}\right) && (14)\\ &= \frac{2N}{n} T_{\text{setup}} + \frac{N}{F} + \frac{N^2}{nF} + \frac{2N D_{\text{8CEs}}}{8F}. && (15) \end{aligned}$$

The first term with T_{setup} is an overhead for data streaming. The second term of $\frac{N}{F}$ is of streaming cycles to write the SRFs. The third term of $\frac{N^2}{nF}$ is of streaming cycles for force computation. The last term of $\frac{2N D_{\text{8CEs}}}{8F}$ is of the overhead due to the prologue and epilogue in pipelining. When N is sufficiently large compared to the pipeline depth, D_{8CEs}, the third term occupies the total time, which can be reduced with $\frac{1}{n}$.

We evaluate the performance with the number of force calculations per second. Since $N(N-1)$ force calculations of Eq.(2) are performed for a single time-step, the performance in force calculations per second [FCPS] is obtained:

$$P(N,n) = \frac{N(N-1)}{T(N,n)} \simeq \frac{N^2}{T(N,n)} \quad \text{for large } N. \tag{16}$$

As described in the next session, we implement n CEs by connecting a unit FPM with 8 CEs. Accordingly, we evaluate the speedup in comparison with the performance for 8 CEs as follows:

$$S_8(N,n) = \frac{P(N,n)}{P(N,8)}. \tag{17}$$

The efficiency is calculated as:

$$E_8(N,n) = \frac{S_8(N,n)}{n}. \tag{18}$$

4. EVALUATION

4.1 System overview

For verification and evaluation we implement the streamed N-body simulation system with a single DE5A-NET board [13]. The board has Intel Arria10 10AX115N3F45I2SG FPGA [14], two DDR3-2133 SDRAMs, and PCI-Express (PCIe) Gen2 interface. Each SDRAM provides the peak bandwidth of 17.1 GB/s.

In addition to the stream-computing core, we also require peripheral circuits including the host interface with PCIe, DDR3 memory controller, and scatter-gather DMAs (SGDMAs). We implement the acceleration platform of Fig.7 by using Intel's system-on-programmable-chip development tool, Qsys. We also develop a PCIe driver for Linux, so that we can transfer data between the host and the FPGA, and control the SGDMAs by software running on a host PC. The platform has the three clock domains of 250 MHz, 266 MHz for DDR3 memories, and up to 225 MHz for user defined region. In the present implementation, the computing core operates at 180 MHz.

4.2 Implementation of computing core

We design and implement FPM and IPM by using the second version of our developed SPGen compiler for stream computation [15] to describe data-flow graphs (DFGs) with formulae. We

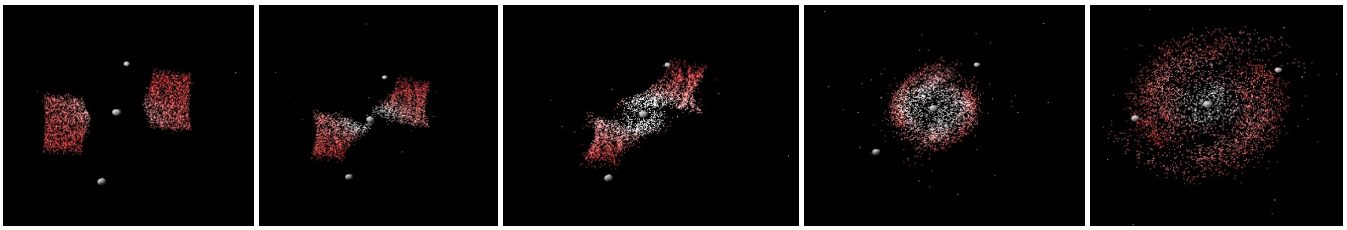

Figure 9: Simulation results by using FPGA with 4096 particles.

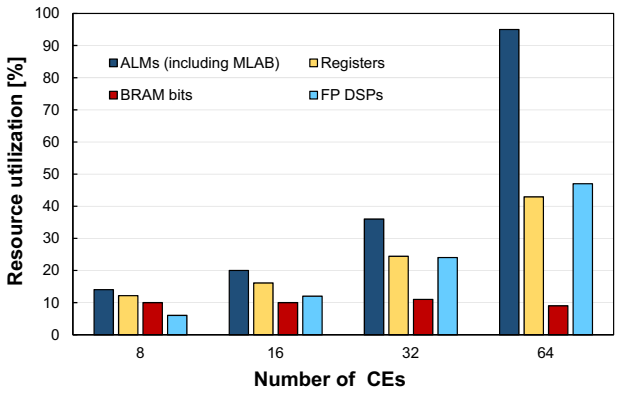

Figure 10: Resource consumption.

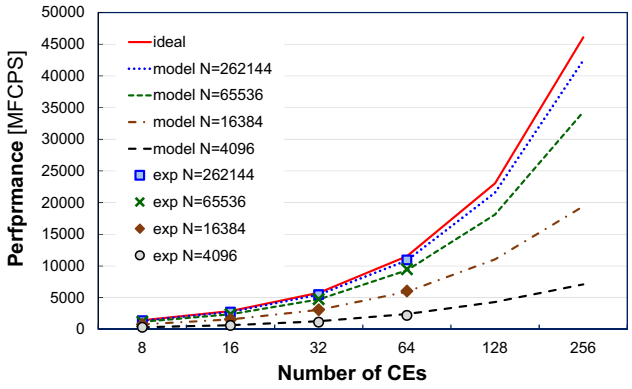

Figure 13: Performance in MFCPS (million force calculation per second).

implement an SRF in Verilog-HDL which can be used for SPGen compiler as its library module. Fig.8 shows the DFG of FPM containing eight CEs to compute Eq.(9). In this figure, the rounded hexagons denote pipeline modules for floating-point computations. The rounded rectangles are modules for other processing, such as using submodules or multiplexing data flows.

The small filled rectangles are delay modules which are automatically inserted by the SPGen compiler to adjust delay cycles along all the paths from the input to the output of the entire module. This delay insertion makes data flow synchronous and smooth, so that a pipeline throughput does not decrease. The integer with "d=" in each module is the number of delay cycles of the module.

We implement an FPM with n CEs and an IPM for $n = 8, 16, 32, 64$ using a unit FPM containing 8 CEs. As shown in Table 1, the pipeline delay of an FPM with 8 CEs is $D_{8\text{CEs}} = 143$. All the computing cores operate at $F = 180$ MHz. For floating-point addition and multiplication, we use Intel's IP core to use the floating-point DSP blocks in Arria10 FPGA. For other operations such as division and square root operation, we use an open-source tool FloPoCo [16] to generate their pipelined cores. All the floating-point operations are in single precision.

4.3 Verification and resource consumption

By comparing the computational results by software and FPGA, we made sure that the implemented hardware with FPGA works correctly and gives an identical result with that of the software-based simulation. Fig.9 shows the visualized results of the FPGA-based N-body simulation with 4096 particles, where three big spheres give larger gravity force than the others.

Fig.10 shows resource consumption of the platforms with the computing core compiled by using Intel Quartus II ver 15.1. The resources are of ALMs (Adaptive logic modules), registers, bits used with BRAM (block-RAM), and FP-DSPs (floating-point DSP blocks). Note that ALMs include MLABs (memory logic array blocks), which are ALMs with LUTs used as a memory.

In the case of $n = 64$ CEs, the utilization of ALMs reaches 95% limiting the implementation while the other resources are less than 50%. Especially, only 47% of the total FP-DSPs are utilized. This high utilization of ALMs is due to small memories such as input buffer FIFOs of DFG nodes and small delay nodes inserted into DFGs. These small memories are implemented using MLABs rather than BRAMs, which are M20K block RAMs in Arria10 FPGA. However, the total amount of used MLABs is not so big, and should fit the unused M20K BRAMs if the number of small buffers is reduced by combining some of them into less larger buffers. In our future work, we will optimize the implementation to reduce the ALM utilization, and fully utilize FP-DSPs to put more CEs for higher performance.

4.4 Computational performance

Here we evaluate the performance of the implemented hardware with the performance model of Eq.(16) and the computing time measured by using RDTSC instructions to read the time stamp counter in a x86 CPU. Fig.13 shows the performance in MFCPS (million force calculation per second) for a different number of particles, $N = 4096$, 16384, 65536, and 262144, as the number of CEs is increased with $n = 8$,16,32, 64, 128, and 256. The lines are the performance estimated with the model. The marks in Fig.13 show the results derived with the measured computing time, which have good agreement with the model. This means that the model is appropriately made for performance estimation.

The red line shows the ideal performance for infinite length of streamed data. With the small data set of $N = 4096$ particles, the performance scales very slowly as n increases. However, when N gets larger such like 262144, the performance gets closer to the ideal one, and can scale well with up to 256 CEs. Figs.11 and 12 show the parallel processing efficiency of Eq.(18) for the model and the measurement, respectively, with a baseline of the performance for 8 CEs. With $N = 4096$ particles, the efficiency decreases into less than 90% when $n = 64$ while it is kept close to 100% with $N = 262144$. Thus, even if we rely on the deeply-pipelined approach, we can make the performance scale well with a large data set.

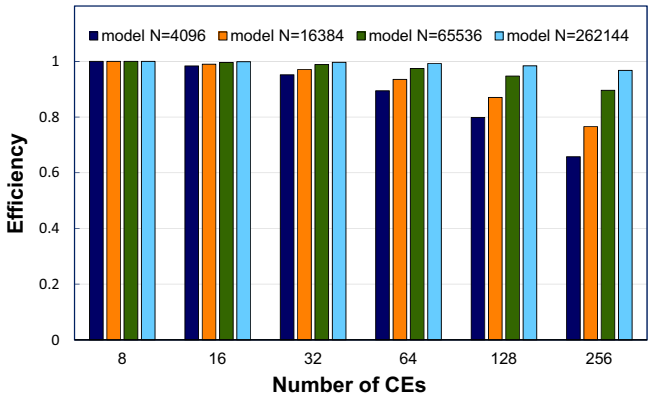

Figure 11: Parallel processing efficiency by model.

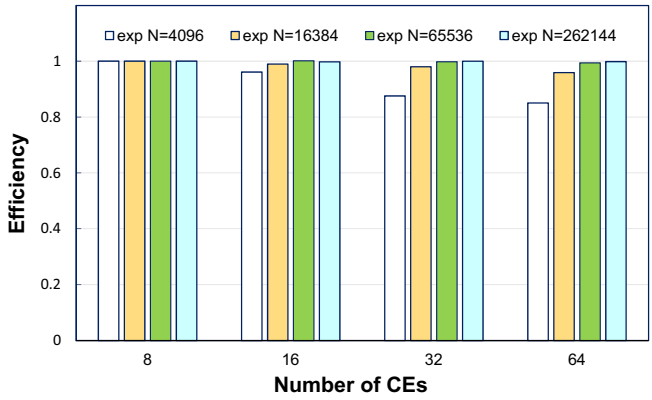

Figure 12: Parallel processing efficiency by experiments.

5. CONCLUSIONS

This paper presents the fully-streamed architecture and design for FPGA-based high-performance N-body simulation. Relying on the deeply-pipelined approach, we can avoid reduction of partial results to the host and broadcast of updated data from the host, resulting in simplicity and higher efficiency. We design the stream-computing core with the force pipeline module (FPM) and the integral pipeline module (IPM), so that all the computation can be performed without the host. FPM has a scalable structure with n computing elements (CEs) and streamed register-files (SRFs). SRFs allow us to quickly write a partial set of particle data to them.

Using the floating-point DSPs of Intel Arria10 FPGA, we implement the prototype system where the computing core operates at 180 MHz. We make sure that the stream-computing hardware scales the performance well from $n = 8$ to 64 CEs for N =262144 particles, finally achieving the performance of 10944 MFCPS (million force calculation per second).

In future work, we will optimize the design so that more CEs can be implemented with a single FPGA. We will build a 1D ring of FPGAs to further scale the performance. We will evaluate and compare the performance, scalability, and performance per power of the multi-FPGA N-body simulation with those by other architectures such as many-core CPUs and GPUs.

Acknowledgments

This research was partially supported by Grant-in-Aid for Scientific Research (B) No.17H01706 and from MEXT, Japan. We are grateful to the university program of ALTERA corporation (now part of Intel) for their long-term support.

6. REFERENCES

[1] Keigo Nitadori, Junichiro Makino, and George Abe. High-performance small-scale simulation of star clusters evolution on cray xd1. *arxiv*, June 2006.

[2] Junichiro Makino. The GRAPE project. *Computing in Science & Engineering*, 8:30–40, January 2006.

[3] Junichiro Makino, Makoto Taiji, Toshikazu Ebisuzaki, and Daiichiro Sugimoto. GRAPE-4: A massively parallel special-purpose computer for collisional n-body simulations. *The Astrophysical Journal*, 480(1):432–446, May 1997.

[4] Atsushi Kawai, Toshiyuki Fukushige, Junichiro Makino, and Makoto Taiji. GRAPE-5: A special-purpose computer for n-body simulations. *Publications of the Astronomical Society of Japan*, 52:659–676, August 2000.

[5] Junichiro Makino, Toshiyuki Fukushige, Masaki Koga, and Ken Namura. GRAPE-6: Massively-parallel special-purpose computer for astrophysical particle simulations. *Publications of the Astronomical Society of Japan*, 55:1163–1187, December 2003.

[6] *GRAPE-DR: 2-Pflops massively-parallel computer with 512-core, 512-Gflops processor chips for scientific computing*, number 18, November 2007.

[7] Martin Langhammer and Bogdan Pasca. Floating-point DSP block architecture for FPGAs. *Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 117–125, February 2015.

[8] David Lewis, Gordon Chiu, Jeffrey Chromczak, David Galloway, Ben Gamsa, Valavan Manohararajah, Ian Milton, Tim Vanderhoek, and John Van Dyken. The stratix 10 highly pipelined FPGA architecture. *Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 159–168, February 2016.

[9] Kohei Nagasu, Kentaro Sano, Fumiya Kono, and Naohito Nakasato. Performance and power evaluation of FPGA-based tsunami simulator using floating-point DSPs. *Proceeding of IEEE Symposium on Low-Power and High-Speed Chips (COOLChips XIX)*, April 2016. poster paper #17.

[10] Kentaro Sano, Yoshiaki Hatsuda, and Satoru Yamamoto. Multi-FPGA accelerator for scalable stencil computation with constant memory-bandwidth. *IEEE Transaction on Parallel and Distributed Systems*, 25(3):695–705, March 2014.

[11] Kentaro Sano, Yoshiaki Kono, Hayato Suzuki, Ryotaro Chiba, Ryo Ito, Kyo Koizumi, and Satoru Yamamoto. Efficient custom computing of fully-streamed lattice boltzmann method on tightly-coupled FPGA cluster. *ACM SIGARCH Computer Architecture News*, 41(5):47–52, 2013.

[12] FPGA Cluster with Stratix V FPGAs. `http://mail.terasic.com.tw/epaper/2014/00_file/DE5-Net_Kentaro%20SANO.p%df`.

[13] TERASIC Corp. WEB. `http://www.terasic.com.tw`.

[14] Altera Corp. WEB. `http://www.altera.com/`.

[15] Kentaro Sano, Hayato Suzuki, Ryo Ito, Tomohiro Ueno, and Satoru Yamamoto. Stream processor generator for HPC to embedded applications on FPGA-based system platform. *Proceedings of the International Workshop on FPGAs for Software Programmers*, pages 43–48, September 2014.

[16] FloPoCo Project WEB. `http://flopoco.gforge.inria.fr`.