Acceleration of the aggregation process in a Hall-thruster simulation using Altera SDK for OpenCL

Hiroyuki Noda
Keio University
3-14-1 Hiyoshi, Yokohama,
223-8522, Japan
noda@am.ics.keio.ac.jp

Ryotaro Sakai
Keio University
3-14-1 Hiyoshi, Yokohama,
223-8522, Japan
ryotaro@am.ics.keio.ac.jp

Takaaki Miyajima
Japan Aerospace Exploration
Agency (JAXA)
7-44-1 Jindaiji-higashi, Chofu,
182-8522, Japan
miyajima.takaaki@jaxa.jp

Naoyuki Fujita
Japan Aerospace Exploration
Agency (JAXA)
7-44-1 Jindaiji-higashi, Chofu,
182-8522, Japan
fujita@chofu.jaxa.jp

Hideharu Amano
Keio University
3-14-1 Hiyoshi, Yokohama,
223-8522, Japan
asap@am.ics.keio.ac.jp

ABSTRACT

The Full Particle-In-Cell (Full-PIC) method is a numerical simulation technique used in the research and development of Hall-thrusters which are a type of electric propulsion engines. It treats ions, neutrons, and electrons as particles and is highly accurate compared with other methods which treat them as a fluid. However, it requires a large computationcost. The Japan Aerospace Exploration Agency (JAXA) is developing a software package called NSRU-Full-PIC that implements such a method. One of the important computing tasks in NSRU-Full-PIC is the aggregation process, which can cause Read-After-write (RAW) hazards, and hence makes parallel computation difficult.

In this paper, we tackle this problem by introducing a reduction operation with an FPGA accelerator. We use Altera's mid-range SoC, Arria 10 which embeds floating-point DSPs for high performance numerical computation. Altera SDK for OpenCL is available for this platfprm and allows easy offloading of complex tasks. We implemented two-types reduction kernel, a Full-Unroll and a Read16. As a result, the aggregation process becomes 72.4 times faster than the single-thread version on an ARM Cortex-A9 1.5 GHz, and 13.3 times faster than that on a Xeon E5-2660 2.9 GHz in Full-Unroll implementation.

1. INTRODUCTION

1.1 Hall-thrusters and Full-PIC method

Figure 1 shows an operating principle of Hall-thrusters. In an annular plasma acceleration part called channel, electrons emitted from the cathode are trapped, and drift to a circumferential direction by applying an radial magnetic field (green line) and an axial electric field (yellow line). Electrons that move circumferentially generate the Hall current in a circumferential direction of the channel. Propellant that flow from a anode into the channel collides with electrons performing circumferential movement and turns into plasma. The Lorentz force generated by the Hall current and the radial magnetic fields hinders the movement of electrons

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

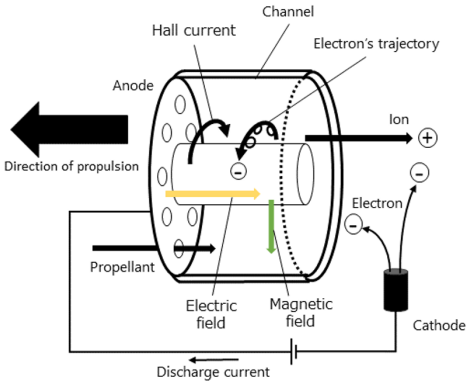


Figure 1: Operating principle of Hall-thrusters

in the plasma to cancel the electric fields in a axial direction. As a result, the electric fields in the axial direction is maintained. By this, only ions in the plasma are accelerated and emitted as a beam outside the thruster. The thruster itself obtains thrust by a reaction force of the generated Lorentz force. The channel is kept a quasi-neutral state because the electrons flowing from the cathode are scattered toward the anode in the channel. In addition, a part of the electrons supplied from the cathode neutralizes the ions emitted outside the thruster, so negative charge in the channel can be avoided.

For development of Hall-thrusters, the numerical simulation is essential, since it is much more cost effective than real experiments. The Full-PIC method, which is classified as the particle method that discretizes the motion of a continuum as the motion of a finite number of particles is used to analyse the state of Hall-thrusters. It doesn't adopt any modelization, so takes a long time to compute [1][2][3][4].

The JAXA has been developing in-house Full-PIC program called NSRU-Full-PIC. An important computing tasks in the code is called the aggregation process, which can cause the RAW hazards. Miyajima et al. introduced an atomic operation to implement the aggregation process on a GPGPU [5]. They could only achieve 10% of performance improvement. Also, a few subroutines of NSRU-Full-PIC were of-

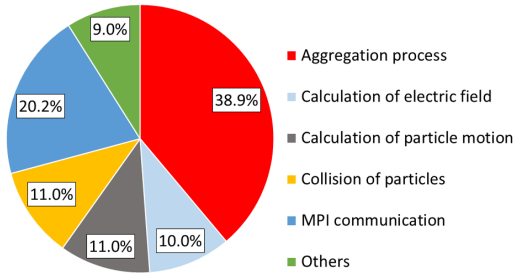


Figure 2: Profile of NSRU-Full-PIC on CRAY XE 6 with 128 processes

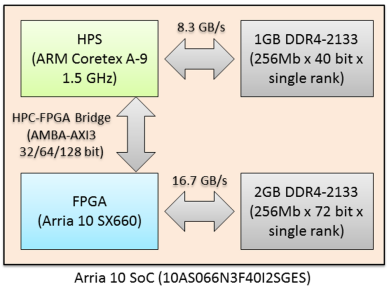


Figure 3: Arria10 consists of HPS and FPGA.

floaded to a Zynq, which is an ARM-based SoC with FPGA has been proposed [6]. However, their focus was not on the aggregation process due to the limited resources of the used platform. This paper addresses the RAW harzards of the aggregation process introducing reduction operations. This is implemented on Altera's mid-range SoC, Arria10, using Altera SDK for OpenCL. The contribution of the paper is as follows.

- The RAW hazards of the aggregation process is avoided modifiying loops to reduction operations.
- The Reduction is implemented on Altera's Arria 10 SoC using Altera SDK for OpenCL.

1.2 Aggregation process in NSRU-Full-PIC

NSRU-Full-PIC is a numerical simulation program for Hall-thrusters under development by JAXA. The code is written in Fortran90 of about 7000 lines. In this research, we adopted this code as a target of acceleration. NSRU-Full-PIC analyses the plasma behavior and state of the electrostatic field in the channel in each time step by updating particle and field physical quantities alternately, which has a large loop structure. One time step corresponds to real time $1 \times 10^{-12}\,sec.$ A computational field called a cell devides the channel inside the thruster. There are 270×310 cells, and the distance between them is $0.2\,mm$. In addition, the number of particles necessary for thruster analysis is tens - hundreds of millions.

We conducted a preliminary experiment of NSRU-Full-PIC. We used a CRAY XE 6[7] supercomputer in Kyoto University with 128 processes. The evaluation environment is as follows, CPU: an AMD Opteron 6200 2.5 GHz, Memory: 64 GB/node, OS: a Cray Compute Node Linux, MPI Process: 128. Figure 2 shows the profiling result of NSRU-Full-PIC by the CRAY XE 6 with 128 MPI processes. The aggregation process accounted for about 40% of the total processing time. Thus, the aggregation processing is a largest part of NSRU-Full-PIC.

The aggregation process adds the physical quantities held by each particle to the four corners of the cell containing them. Figure 4(a) shows how it is performed on multiple particles. Here, the values held by two particles p1 and p2 are added to GP[0-3], and the values are then updated. If we execute the computation of two particles in parallel, RAW hazards can occur, which means that it has to be done sequentially. This is a vital problem to be addressed to enable efficient parallel processing.

For the practical simulation 270×310 cells are used and each cells has 256 particles at most. Thus, $270 \times 310 \times 256 = 21,427,200$ particles are in the simulation.

2. ALTERA SDK FOR OPENCL AND ARRIA 10 SOC

2.1 Altera's Arria 10 SoC

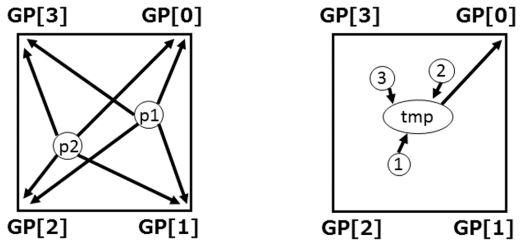
Arria 10 SoC is a mid-range SoC FPGA developed by Altera [8]. Arria 10 SoC consists of Hard Processor System (HPS) with dual core ARM Cortex-A9 MPCore and the FPGA logic as shown in Figure 3. The HPS unit is consisting of a processor unit including a CPU, cache, on-chip memorys, external memory interface, a communication interface controller, and AXI interconnect. The FPGA logic part of Arria 10 SoC embeds hardened floating-point DSPs for high performance numerical computation. Arria 10 SoC can compute floating point operations using this much faster than common FPGAs such as Zynq.

2.2 Altera SDK for OpenCL

Altera SDK for OpenCL is an OpenCL-based High-Level Synthesis environment for FPGAs developed by Altera[9]. It is designed for describing high-performance FPGA circuits in a short time. OpenCL is a parallel programming framework that can be used in a multiprocessor environment composed of various processors such as CPU, GPU, and FPGA. Altera SDK for OpenCL introduces two kinds of code, kernel code and host code. The kernel code is for the operation of an arithmetic processor (OpenCL device), and the host code is for an operation of the control processor (host). They are described in OpenCL C language and C++ with OpenCL runtime Application Programming Interface (API). In addition, Altera SDK for OpenCL provides a board support package (BSP) that supports peripheral circuits such as PCIe bus between the OpenCL device and host, and an interface with external memory. By using the BSP, a user can operate FPGA without designing the peripheral interface.

The OpenCL programming model consists of two hierarchical layer, work-group and work-item. Work-group is a set of work-items, and the OpenCL device executes work-item based processing. Global memory and constant memory can be accessed from all work-groups. The global memory is readable and writable, but the constant memory is read only. On the other hand, local memory is used for work-group and can be accessed from all work-items belonging to it. Also, private memory is work-item specific.

There are two types of kernel program in the Altera SDK for OpenCL, a Single work-item kernel and a NDRange kernel. The Single work-item kernel corresponds to task parallel model. There is only one work-group and one work-item in Single work-item kernel, and so kernel code can be described like sequential programming. The compiler extracts the parallelism in the kernel code, and makes pipelines in the loop. On the other hand, the NDRange kernel corresponds to data parallel model. Each work-item corresponds to a thread space and is executed in a pipelined manner. With NDRange kernel, it is possible to specify kernel pipeline multiplexing and vectorization for multiple work-items, which contribute to improvement of throughput. However, it can cause an increase in FPGA resource usage.



(a) Aggregation process on multiple particles sequentially could cause RAW hazards.

(b) Reduction and temporary value are used.

Figure 4: The aggregation process in a cell

3. IMPLEMENTATION

3.1 Avoiding the RAW hazards

As described in Section 1, the step including the aggregation process requires high computational cost, and the avoidance of the RAW hazards is essential for parallelization. Figure 4(b) shows the outline of our implementation. Here, in order to avoid the RAW hazards, particle basis computing is changed into cell by cell computing in the source code level. That is, values of particles are added to four corners of the cell. As shown in Figure 4(b), the value of GP[0] is updated by a temporal variable which gathers the values of all particles in the cell. Since the update is done at once after values of all particles are added into the temporal variable, the RAW hazard never occurs. The same processing is performed for GP[1-3]. In this case, the reduction which is a common computation pattern in high speed computing can be used for computing temporal variables. Although the parallelism of the reduction is decreased at the later steps of communication, there are a lot of cells and the tree structure implemented on FPGA logic can be used in the pipelined manner.

3.2 Reduction Kernel

We implemented two-types of reduction codes, a Full-Unroll and a Read16. Both code adopted Single work-item kernel. In the implementation, the reduction is single precision floating point operations.

3.2.1 Full-Unroll implementation

A Full-Unroll implementation adopts eight-loops structure as shown in the following partial pseudo code. Each level of reduction was described in independent loop. For each loop, unrolling is performed by adding *#pragma unroll* in the source code, and all the loops are fully expanded. Additionally, we added *volatile* to the input argument of the kernel code. caching generation was invalidated, which can reduce the FPGA resource usage.

Listing 1: Partial pseudo code of Full-Unroll

```
1  kernel void
2  full_unroll(__global volatile const float* restrict input,
3                    __global float* restrict output)
4  ...
5  #pragma unroll
6  for (i = 0; i < 256; i++) // level1 (256->128)
7    buf1[i] = input[i*2 + 0] + input[i*2 + 1];
8  #pragma unroll
9  for (i = 0; i < 128; i++) // level2 (128->64)
10   buf2[i] = buf1[i*2 + 0] + buf1[i*2 + 1];
11 ...
12 #pragma unroll
13 for (i = 0; i < 2; i++) // level8 (2->1)
14   sum = buf7[i*2 + 0] + buf7[i*2 + 1];
15 ...
```

3.2.2 Read16 implementation

A Read16 implementation adopts single loop structure as shown in the following partial pseudo code. It use a vector-type called Read16 so as to increase input bandwidth from DDR memory on the FPGA. 16-elements reduction was performed in a single loop and the loop runs 16 times. Each level of reduction is manually described in tree-type structure. We evaluated the number of unrolls N.

Listing 2: Partial pseudo code of Read16

```
1  kernel void
2  read16(__global volatile const float16* restrict input,
3                    __global float* restrict output)
4  ...
5  #pragma unroll N
6  for (int i = 0; i < 16; i++){
7    // level1 (16->8)
8    level2[0] = input[i].s0 + input[i].s1;
9    level2[1] = input[i].s2 + input[i].s3;
10   ...
11   level2[7] = input[i].sE + input[i].sF;
12
13   // level2 (8->4)
14   level3[0] = level2[0] + level2[1];
15   ...
16   level3[3] = level2[6] + level2[7];
17
18   // level3 (4->2)
19   level4[0] = level3[0] + level3[1];
20   level4[1] = level3[2] + level3[3];
21
22   // level4 (2->1)
23   sum += level4[0] + level4[1]; }
```

3.3 Evaluation

We compare the computation speed of the reduction circuits implemented on the FPGA and that of the software execution on two CPUs, an ARM Cortex-A9 1.5GHz on Arria 10 SoC and a Xeon E5-2660 2.90 GHz. The evaluation environment is as follows. For the FPGA, we used Altera SDK for OpenCL 64-bit Offline Compiler ver. 16.0.2 with -v, -g and –fp-relaxed option for the kernel code compilation, and gcc compiler ver. 4.8.3 with -O3 option for the host code compilation. For the ARM Cortex-A9, we used gcc compiler ver.4.8.4 with -O3 option. For the Xeon E5-2660, we used gcc compiler ver. 4.4.7 with -O3 option.

First we analysed generated code and compared the two-types of reduction kernels. We examined the kernels adding –dot option to Altera SDK for OpenCL Kernel Compiler (*aoc* command). In the case of Read16, four DSPs and inner products are used to calculate as shown in Figure 5. Inner product of 8-elements (__acl_fp_hard_dot8), and inner product of 4-elements, and then inner product of 2 were performed. Finally, simple addition was performed. The latency of each computation was 11, 8, 6 and 4. Input interval (II) was 1. In the case of Full-Unroll, similar logics were generated.

Figure 6 shows the computation time with our implementation and the above CPUs when the size of the input data is 270 x 310 sets of 256 particles. As a result, the Full-Unroll implementation was about 72.4 times faster than that of an ARM Cortex-A9, and about 13.3 times faster than that of a Xeon E5-2667. the Read16 implementation of which the number of unroll is 4 was about 44.9 times faster than that of an ARM Cortex-A9, and about 8.3 times faster than that of a Xeon E5-2667. In the Read16 implementation, the number of unroll with the fastest computation speed is 4. Table 1 shows the performance of our implementations.

In our implementation we added *volatile* to the input argument of the kernel code, which can reduce the ALUTs by 50%, the Registers by 29%, and the BRAMs by 50%. In

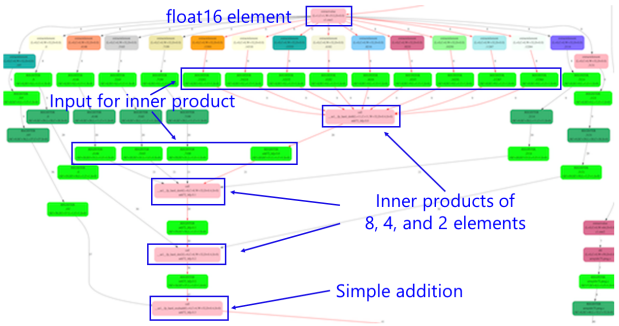


Figure 5: Generated schimatic view of Read16. Inner product was used instead of simple addition.

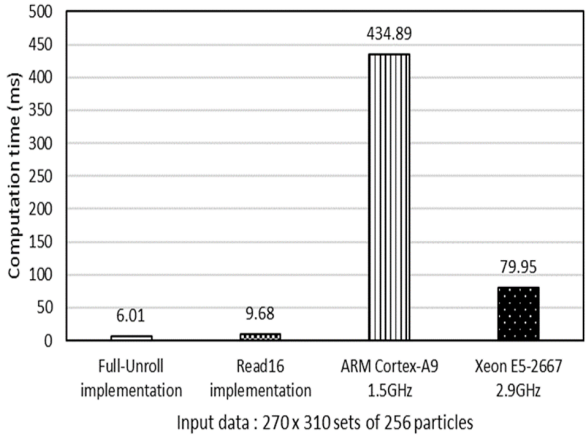


Figure 6: The computation time with Arria 10 SoC and the above CPUs when the size of the input data is 270 x 310 sets of 256 particles

the host code, We use the clEnqueueWriteBuffer and clEnqueueReadBuffer, which is OpenCL runtime API. They contributed to the increase of input BW compared with the case using shared memory of SoC.

Here, we estimate the speed up ratio in the overall offloading of NSRU-Full-PIC incorporating the Full-Unroll implementation. According to the NSRU-Full-PIC profiling described in Section 2, the aggregation process accounted for 38.9% of the total processing time. Therefore, we can estimate that the speed up ratio is about 1.62 times compared with the execution by an ARM Cortex-A9 and about 1.56 times compared with the execution by a Xeon E5-2667.

4. CONCLUSION

We offloaded an aggregation process of NSRU-Full-PIC which is a particularly high computation cost to an Arria 10 SoC using Altera SDK for OpenCL. The reduction computation is adoped to avoid the RAW hazards in aggregation process. We implemented and evaluated two-types Sinlge work-item kernel, a Full-Unroll and a Read16. The Full-Unroll was the fastest implementation, but resource usage is high. On the other hand, the Read16 is slower than the Full-Unroll, but resource usage is much smaller than Full-Unroll. We compared the fastest FPGA implementation with single-thread execution on an ARM Cortex-A9 (1.5 GHz) and a Xeon E5-2660 (2.9 GHz). As a result, our implementation was about 72.4 times faster than that of an ARM Cortex-A9, and about 13.3 times faster than that of a Xeon E5-2667.

Table 1: Performance of our implementations

	Unroll	Computation time [ms]	BW [MB/s]	Logics [%]	ALUTs [%]	Regs [%]	BRAMs [%]	DSPs [%]
Full-Unroll	Full	6.01	14544.2	9	4	5	8	16
Read16	1	9.98	1008.6	2	1	1	3	1
	2	10.13	1617.8	3	2	2	4	2
	3	1026.95	210.4	5	3	3	7	3
	4	9.68	2974.1	4	2	2	6	4
	5	964.95	11.2	7	3	3	10	5
	6	480.26	22.4	7	3	4	10	6

As a future work, we plan to extend the current implementation of aggregation process to all cells in the code. We also plan to evaluate the overall offloading of NSRU-Full-PIC incorporating the parallelization of the aggregation processing to the cell base.

ACKNOWLEDGMENT

The present study was supported in part by the JST/CREST program entitled "Research and Development on Unified Environment of Accelerated Computing and Interconnection for Post-Petascale Era" in the research area of "Development of System Software Technologies for post-Peta Scale High Performance Computing".

5. REFERENCES

[1] Yokota Shigeru, Komurasaki Kimiya, and Arakawa Yoshihiro, . Plasma Density Fluctuation Inside a Hollow Anode in an Anode-layer Hall Thruster . In *42th Joint Propulsion Conference and Exhibit, AIAA-2006-5170*, 2006.

[2] Hirakawa Miharu . Electron Transport Mechanism in a Hall Thruster . In *IEPC-97-021*, 1997.

[3] Justin M. Fox . *Advances in Fully-Kinetic PIC Simulation of a Near-Vacuum Hall Thruster and Other Plasma Systems* . PhD thesis, 2007.

[4] James Joseph Szabo . *Fully Kinetic Numerical Modeling of a Plasma Thruster* . PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, 2001.

[5] Takaaki Miyajima, Shinatora Cho, and Naoyuki Fujita. A study of gpu acceleration of "source" part in hall-thruster simulation. In *IEICE Tech. Rep.*, Vol. 115 of *CPSY2015-62*, pp. 7–12, Dec. 2015.

[6] R. Sakai, N. Sugimoto, T. Miyajima, N. Fujita, and H. Amano. Acceleration of full-pic simulation on a cpu-fpga tightly coupled environment. In *2016 IEEE 10th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSOC)*, pp. 8–14, Sept 2016.

[7] supercomputer system (from October, 2016) — Supercomputer System —Academic Center for Computing and Media Studies, Kyoto University. http://www.iimc.kyoto-u.ac.jp/ja/services/comp/supercomputer/. 2016/12/29/20:50.

[8] Intel Corporation. Arria 10 SoC - Features:. https://www.altera.com/products/soc/portfolio/arria-10-soc/features.html. 2017/01/29/14:04.

[9] Intel Corporation. Intel FPGA SDK for OpenCL Programming Guide - aocl_programming_guide. https://www.altera.com/en_US/pdfs/literature/hb/opencl-sdkaocl_programming_guide.pdf. 2016/11/17/14:00.