Acceleration of Full-PIC simulation on a CPU-FPGA tightly coupled environment

Ryotaro Sakai, Naru Sugimoto, and Hideharu Amano
Graduate School of Science and Technology,
Keio University
3-14-1 Hiyoshi, Kouhoku-ku, Yokohama, Kanagawa, Japan
Email: cfd@am.ics.keio.ac.jp

Takaaki Miyajima and Naoyuki Fujita
Numerical Simulation Research Unit,
Aeronautical Technology Directorate,
Japan Aerospace Exploration Agency (JAXA)
7-44-1 Jindaiji-higashi, Chofu, Japan

***Abstract*—Hall thruster is a sort of electric propulsion and has been studied in many research institutes. In the design process of Hall thruster, a numerical simulation called Full-PIC (Particle-In-Cell) method is used. Although this simulation provides high accurate result, it is known as a very time consuming job. In this paper, we present a study of acceleration of Full-PIC simulation on a CPU-FPGA tightly coupled environment. A high-load part is selected and off-loaded to an FPGA. Zynq-7000 and Vivado HLS are used for implementation. To optimize the implemented design, every target process was divided into some parts for pipelining and adjustment interval.**

Three off-loaded subroutines named ”field_source”, “particle_att_ion” and “particle_att_ele” achieved 8.53 times, 12.78 times and 14.95 times performance compared with the software execution, respectively. The total execution time of target part is sped up 5.17 times compared with Cortex-A9 667MHz in Zynq.

I. Introduction

A Hall thruster is one of the most successful electric propulsion which has three major advantages as follows. First, it has high degree of thrust efficiency more than 50 % at wide specific impulse range (1,000s ~ 5,000s). Second, thrust density is high compared with ion thruster, and third, it is highly useful as a compact and simple propulsion system for versatile crafts. Consequently, Hall thrusters are considered to be an ideal propulsion system for satellite missions like station keeping, orbit transfer, or deep space exploration, and currently intensively studied and developed in different institutes and companies.

The thrust of Hall thruster is an emission of high density plasma ion. The mechanism of thrust generation is based on plasma physics and classified in electric propulsion (EP). EP is one of the propulsion systems mainly used for space missions. Although the mechanism of thrust generation of EPs is the same as that of the conventional chemical propulsion (CP) systems, usually EPs can achieve exhaust velocity and hence specific impulse higher than CPs by order of magnitude. EP can achieve much higher energy concentration and thus higher exhaust velocity than CP, which enables space missions can never be completed by CP like deep space exploration, for example.

Full-PIC (Particle-In-Cell) methods are developed for a kinetic description of plasma and is used for Hall thruster simulation. Full-PIC is classified as first-principles model since all species of the particles including the electron are tracked and simulated directly. The space and time resolution are respectively restricted by the Debye length and the plasma frequency, which makes the simulation extremely computationally expensive. Consequently, the computational cost is the major drawback of Full-PIC models which prevent them from being widely used despite their effectiveness. Japan Aerospace Exploration Agency (JAXA) has been researched Japanese Hall thruster and developing Full-PIC code called ”NSRU-Full-PIC”. Current NSRU-Full-PIC is written in Fortran and parallelized by OpenMPI. Practical simulation takes one week at least, thus shorten the simulation time is highly demanded. JAXA is investigating acceleration by GPU and FPGA.

In this paper, we focus on an acceleration of NSRU-Full-PIC on a CPU-FPGA tightly coupled environment. We first ported NSRU-Full-PIC to ARM CPU on Zynq-7000 AP SoC [1] and measured computational time. Then we choose a time consuming step called SOURCE1 that includes stencil computation and off-loaded to FPGA. High-level synthesis (HLS) which converts C or C++ into the register transfer level (RTL) was adopted. Although Vivado-HLS was used, the original code, of course, cannot be directly converted as it is. In order to optimize the design, first, we divided processes consisting of a number of double loops into some segments. Second, we designed stencil buffers for stencil computation and stream distributors by using HLS. Finally, a pipelined data path was built with divided segments and the input data timing was adjusted. As a result, three subroutines: “field_source”, “particle_att_ion” and “particle_att_ele” achieved 8.53 times, 12.78 times and 14.95 times performance compared with the software execution, respectively. The total execution time of SOURCE is sped up 5.17 times compared with Cortex-A9 667MHz in Zynq.

II. Numerical Simulation of Hall thruster by Full-PIC

A. Hall thruster

The thrust of Hall thrusters is generated by the electric static acceleration of propellant, whereas the thrust is transferred to the thruster by electromagnetic force generated by the Hall current. The Hall thruster's principle operation is illustrated in Fig. 1. The electrons emitted from the cathode (red line) are trapped by the radial magnetic field (green line) due to the

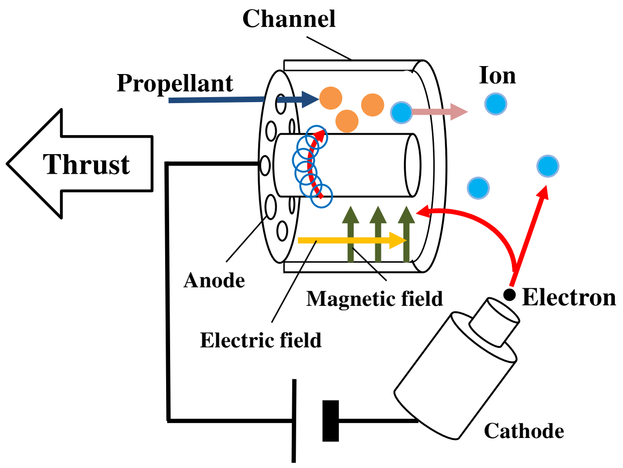


Fig. 1. Hall thruster's principle operation

$\mathbf{E} \times \mathbf{B}$ drift. Radial magnetic field is imposed throughout the channel by a magnetic circuit. Additionally, strong axial electric field (yellow line) is sustained throughout the discharge channel by the radial magnetic field. This azimuthal guiding center drift (blue circles) forms Hall current, which enables the heavy particle acceleration and thrust generation. That's why this propulsion mechanism is called "Hall thruster". The entrapped electrons are gradually accelerated toward the anode (pink line) due to the diffusion, and ionize the propellant supplied through the anode (dark blue line). The ionized propellant is accelerated by the axial electric field and exhausted in high speed generating thrust.

A Hall thruster has became one of the most successful electric propulsion which has three major advantages.

High thrust efficiency (more than 50 %) at wide specific impulse range (1,000 s ~ 5,000 s), high thrust density compared with ion thruster, and high degree of usability as a propulsion system (simple and compact, can use various propellant).

Japan developed 5 kW-class Hall thruster and has been developing larger ones. Not only Japan, Russia has already lunched more than 100 Hall thrusters for satellite station keeping. NASA is developing 50 kW-class high power Hall thruster, and ESA used their own Hall thruster for a lunar exploration mission.

B. *Full-PIC method*

Full Particle-In-Cell (PIC) method is used for simulation of plasma and nuclear fusion. Actual plasma consists of a massive number of particles (ion, electron and neutron), and it is impossible to simulate all of them. Thus, super-particle and background mesh are introduced. Super-particle represents multiple particles (100 ~ 1,000 particles), and background mesh represents magnetic field. PIC solves equation of motions for super-particles and Maxwell equations to find interactions between super-particles. It is very similar to the first-principle description of plasma as a system of charged particles. PIC can recreate an actual phenomenon, but its computational cost is very high. Most time consuming parts are tracing all the super-particles and the interaction between super-particles and a background mesh.

Governing equations of Full-PIC are Boltzmann equation and Maxwell equation. Boltzmann equation governs motion of charged particles, and Maxwell equation governs temporal/spatial variation of electromagnetic field. Assuming the time-invariant magnetic field, the system reduces to the Poisson equation (Equation 1) and the equation of motion for each particle (Equation 2) [2].

$$\Delta\phi = -\frac{\rho}{\varepsilon} \tag{1}$$

where, ϕ is potential, ρ is space charge, and ε is permittivity.

$$\begin{aligned} m\alpha &= F \\ m\alpha &= F_{electric} + F_{magnetic} \\ m\frac{d^2\boldsymbol{x}}{dt^2} &= q\boldsymbol{E} + q(\boldsymbol{v} \times \boldsymbol{B}) \end{aligned} \tag{2}$$

where, m is mass, q is charge of the species, $\boldsymbol{E}$ is electric field, $\boldsymbol{B}$ is magnetic flux density, and $\boldsymbol{v}$ is velocity of electron. The trajectories of all the particle species are directly traced according to the field, whereas the field is calculated according to the space charge weighted from the particles.

C. *NSRU-Full-PIC*

NSRU-Full-PIC is a Full-PIC code for Hall thruster simulation under development by JAXA. It is written in 7,000 lines of Fortran90 code. OpenMPI is used for parallelization, but OpenMP is not used. Similar to the other Full-PIC codes, processing step of NSRU-Full-PIC consists of the following main five steps. Additionally, the whole processing step is described in Fig. 2. SOURCE1, FIELD, PUSH, SOURCE2 and COLLISION respectively corresponds to step 4, step 5, step 6/7, step 8 and step 9.

SOURCE1 computes interaction between particles and magnetic field. FIELD computes electric field from magnetic field and distribute by MPI. PUSH updates position and velocity of all the charged particles. SOURCE2 is a pre-processing for COLLISION. COLLISION simulates inter-particle collisions modeled by Direct Simulation Monte Carlo.

Fig. 3 shows a processing time of NSRU-Full-PIC on an actual environment. We used CRAY XE6 [3] and 128 MPI processes. According to the figure, "SOURCE1" (step 4 in Fig. 2) accounts for 40% of the whole processing time. We focus on the most time consuming part and investigate an acceleration technique by using a CPU-FPGA tightly coupled environment.

D. *Related Work*

Miyajima et.al reported GPU implementation of NSRU-Full-PIC [4]. They focused on the same part and achieved only 1.1 times speed up. Their GPU implementation was straight forward and didn't optimize for the GPU architecture. PIConGPU [5] is a mult-GPU implementation of PIC method. We cannot compare it directly with NSRU-Full-PIC, since

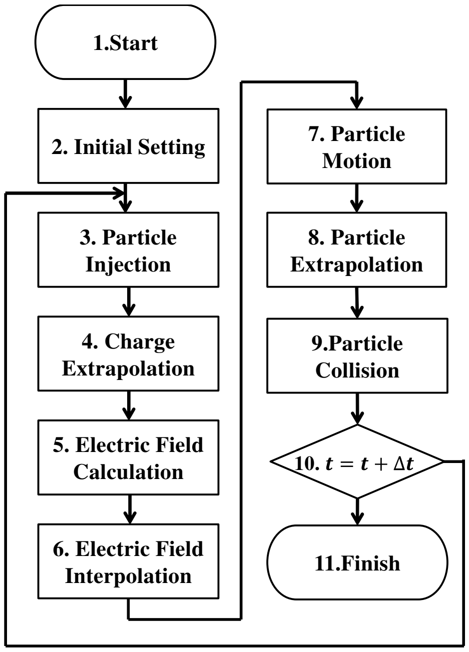


Fig. 2. Flow chart of NSRU-Full-PIC

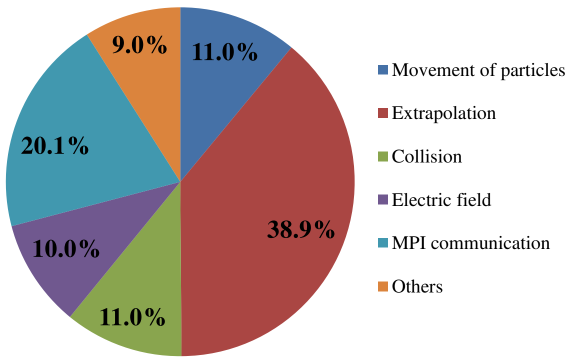


Fig. 3. Processing time profile of NSRU-Full-PIC on CRAY XE6

PIConGPU is based on another method of approximation. To authors knowledge, there are no research that accelerates Full-PIC code on CPU-FPGA environment.

Trials to use Zynq for scientifc application are also rare, since it is originaly developed for embedded usage. Zedwulf [6] is a 32-node Zynq SoC cluster which can be used for scientific computation. Sugimoto [7] also proposes a parametric survey by using Zynq cluster. All of them use a lot of Zed boards for enhancing their performance with parallel processing. Although our trial can be used on such clusters, we rather focus on the stand-alone execution on a single board equipped on a satellite.

III. CPU-FPGA tightly connected architecture in a satellite

A. Zynq

Zynq-7000 All Programmable SoC (Zynq) is a System-on-Chip (SoC) providing CPUs and FPGA by Xilinx. It consists of Processing System (PS) including ARM CPUs and Programmables Logic (PL) with an FPGA. The communication between PS and PL is done through ARM Advanced eXtensible Interface (AXI), and co-design with CPU and FPGA can be easily done.

PS is consisting of Application Processor Unit (APU) including CPU core, memory interface, I/O peripheral, and AXI interconnect, while PL part is Xilinx's 7 series FPGA. Here, Xilinx's Zed Board, a tiny evaluation board as shown in Fig. 4is used. Under the central heat sink, XC7Z020 chip, the second smallest chip in Zynq-7000 series is mounted.

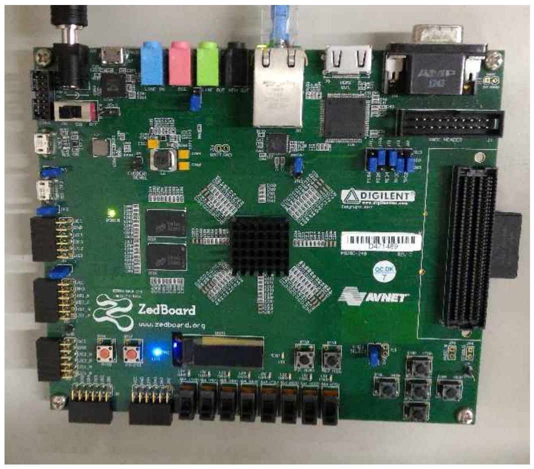


Fig. 4. Zed Board

Zynq is low energy, low cost and compactly implemented. CPU-FGA co-design is easily done by using Vivado HLS design environment.

B. Providing a Zynq-board on a satellite

As shown in the previous section, the Hall thruster simulation is essential for design and development of Hall thruster engines. Moreover, the simulation is used for analyzing the state of working engine. When engine trouble happens on the satellite or space craft traveling far from the earth, the analysis of the engine will be done on the base station using the data sent from the satellite.

However, the communication between satellite becomes often unstable to transmit a large amount of data. In such a case, the simulation of engine is needed to be done in a stand-alone computer in the satellite. According to the result of analysis, the satellite can control the engine by itself without the command from the base station. For such a purpose, an embedded SoC like Zynq is a good candidate as a versatile accelerator for energy-efficient computation in the satellite. Since it includes both general purpose computers (PS part) and FPGA (PL part), it can be used both for general purpose computation with Linux operating system and hardwired logic for controlling sensors and mechanical part in the satellite. Although the current SRAM-based FPGAs are not suitable to cosmic usage because of the soft errors by the cosmic rays, various fault tolerant techniques have been developed [8] [9], and some trials have been reported [10] [11]. Considering the benefit of in-system configuration, SoCs including CPUs and FPGAs will be equipped in the satellite in the future.

Considering the benefit of in-system configuration, SoCs including CPUs and FPGAs will be equipped in the satellite in the future.

IV. Implementation

Here, we implemented NSRU-Full-PIC on a Zynq board. First, we installed Linaro Ubuntu Linux OS which uses the SD card as its file system on ARM Cortex-A9 in Zynq. Then, the Fortran-90 code of NSRU-Full-PIC was complied on it. In five steps in NSRU-Full-PIC, the load of SOURCE1 is relatively high. Like previous research [4], we selected this part for acceleration. Then, we further profiled SOURCE1 on ARM Cortex-A9, and the result is shown in Fig. 5.

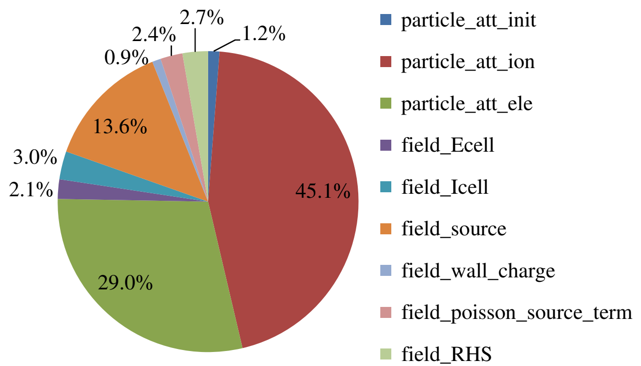


Fig. 5. The profiling of SOURCE1 on Cortex-A9

As shown in the figure, the 90% of execution time of SOURCE1 is occupied with three subroutines: (1) Assignment of charge to grids ("particle_att_ele") (2) Assignment of ion to grids ("particle_att_ion") and (3) Calculation of mutual interaction in the electric field ("field_source"). Here, we selected them as the first off-load target into FPGA because of their simple data structure. Although the ratio in SOURCE1 is still limited, the similar computation is used in other steps.

1) "field_source": "field_source" executes 2×2 stencil computation of the grid for simulating the surface of Hall thruster built by Boron Nitride (BN). When plasma encounters to the BN wall in a vacuum chamber, various interactions occur. By calculation of mutual interaction of ions in the electric field, the correction factor in the right-hand side of Poisson equation (Equation1) is given. Here, the wall is corresponding to the colored part in Fig. 6. The conditions used in the computation is different in inner-wall, surface of wall and outer-wall. "field_source" is consisting of three loops: "first loop","second loop", and "third loop", and the time consuming part is 2×2 stencil computation executed in "first loop" and "third loop".

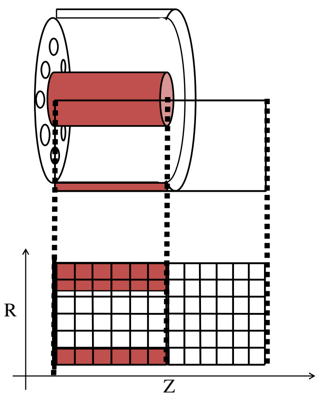


Fig. 6. The calculation area of Hall thruster

2) "particle_att_ion": "particle_att_ion" accounts for about half of execution time of SOURCE1, since some tough steps are needed to assign ion to grids. It is also consisting of three loops: "Cloud in Cell (CIC)", "boundary processing", and "interpolation". The structure of "CIC" can cause a lot of Read-After-Write (RAW) hazards because of self-assignments of grids. This problem leads to not only incorrect computation results but also waste of computational power for avoiding it. Moreover, the time consuming part, 2×2 stencil computation is in the "interpolation".

3) "particle_att_ele": Except for dealing with electron, this subroutine is almost the same as "particle_att_ion".

A. High Level Synthesis

Xilinx's Vivado HLS is used for implementing three subroutines on the PL part of Zynq. Like other practical scientific application codes, NSRU-Full-PIC executes a large complicated computation. Compared with describing such a code with HDL, HLS can much reduce the time to design. Especially, the following functions were useful in the design. (1) The bus control between PS part and PL part can be treated easily, (2) pipeline interval can be easily adjusted, and (3) resource can be estimated at the earlier step of the design.

First, the code described in Fortran90 was manually translated into C language, a front-end of HLS. Here, for three loop structures were designed as modules, respectively. Buffers for stencil computation were provided between "first loop" and "third loop" in the "field_source" and "interpolation" in the "particle_att_ion" and "particle_att_ele", respectively. The input/output port was unified to use 64bit-data for sending as AXI stream. Then, a pipeline structure was formed with three modules by using the directive. Finally, the design was optimized by setting the interval manually.

All modules work at 100MHz clock, and controlled by the ARM CPU through the AXI Lite. Designed modules and IP cores supported by Xilinx were connected with Vivado IP Integrator.

B. field_source module

The block diagram of ”first loop” is shown in Fig. 7. Although there are about 20 data arrays, 4 arrays “grd_zI”, “grd_rE”, “grd_rI” and “grd_zE” among them and results of computation with other arrays are transferred to “second loop”. “grd_zI” and “grd_rE” are updated once and “grd_rI” is updated twice, using parameters or other inputs, while “grd_zE” is not updated. “aa”, “bb”, and “xx” hold the results of 2×2 stencil computation of those inputted arrays. The computation results from “grd_en” are stored into “aa” and “bb”, while “xx” holds the results of “grd_Ab”.

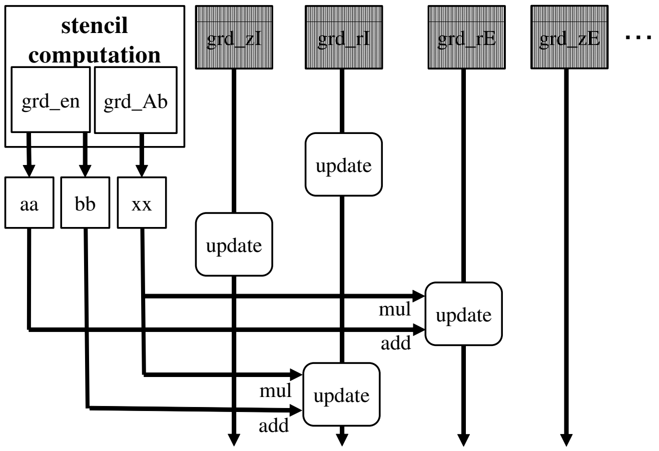
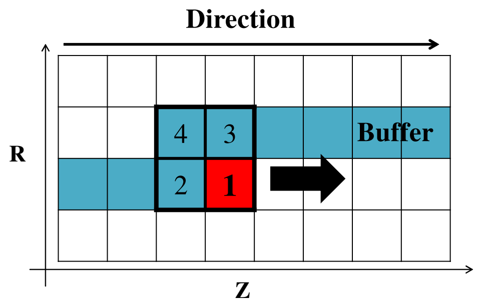


Fig. 7. The block diagram of ”first loop”

C. Stencil computation module with buffer

The stencil computation, the main processing of ”field_source” and “particle_att_ion(ele)” computes the value of target grid and values of surrounding grids as shown in Fig. 8. In this example, values of red colored target grid, left, upper and upper-left are added and stored into the other matrix. For implementing on the FPGA, blue colored buffer is required for storing the values of past used grids. Thus, the module requires buffer IPs.



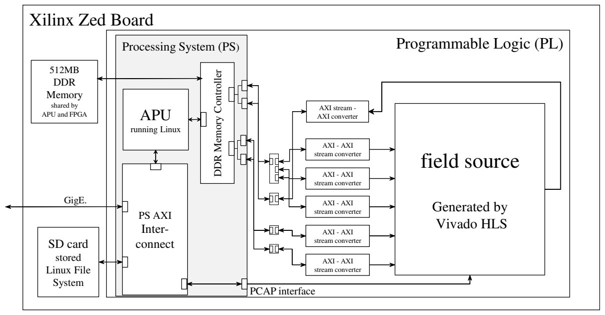
$$\mathrm{aa} = \alpha(\boldsymbol{grd\_en}[\boldsymbol{Z}][\boldsymbol{R}] + grd\_en[Z-1][R] +$$
$$grd\_en[Z][R+1] + grd\_en[Z-1][R+1])$$

Fig. 8. 2×2 stencil computation with buffer of “first loop”

D. Overall implemented design

Fig. 9 shows an overall design implemented to the Xilinx Zed Board. In the PS, Linux is running on the APU with ARM Cortex-A9, and processes not off-loaded to the PL are executed. Furthermore, DDR3 SDRAM which is under control of Linux takes a role of data transfer between PS and PL. In the PL, Intellectual Property (IP) cores provided by Xilinx and modules generated by Vivado HLS (shown in Fig. 10) are connected by Vivado IP Integrator, and 64bit AXI and AXI stream are adopted for transferring protocol. Moreover, some other modules are controlled by APU via AXI Lite protocol, although they are omitted in Fig. 9.

21 types of data are inputted to the field_source module, and 2 types of data are outputted. Transferring them individually through the dedicated AXI Stream is impractical because of the constraints of resources and complexity. Therefore, in this study, we used demultiplexers (DEMUX) as shown on the left side of Fig. 10 to improve the efficiency of data transfer between the CPU and FPGA. One AXI Stream is shared up to four types of data and it is divided by DEMUX. Each input data is transferred through the AXI Stream sequentially after being stored to the DDR memory as the form of Array Of Structures (AoS). At last, it is written to the target register to compute. Additionally, although DEMUX was generated by Vivado HLS, it is not included in the evaluation described later.



Xilinx Zed Board
Processing System (PS)
Programmable Logic (PL)
512MB DDR Memory shared by APU and FPGA
APU running Linux
DDR Memory Controller
GigE.
PS AXI Inter-connect
SD card stored Linux File System
AXI stream - AXI converter
AXI - AXI stream converter
AXI - AXI stream converter
AXI - AXI stream converter
AXI - AXI stream converter
AXI - AXI stream converter
field source
Generated by Vivado HLS
PCAP interface

Fig. 9. Overall implemented design

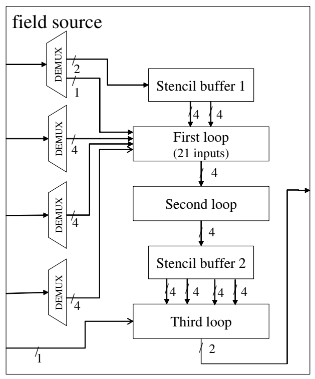


Fig. 10. The detail of “field_source” module

V. Evaluation

A. Evaluation Setup

Table I shows the evaluation setup. The number of grids and particles decide the accuracy and execution time of the simulation. Here, the number of grid is set to be $271 \times 311 = 84281$, that is, the target region is divided into about $0.1mm^2$ grid. When computational speed is more important than the accuracy, we can reduce the number into a half. The number of particle was limited by the memory size on Zynq.

TABLE I
Evaluation environment.

Zynq	XC7Z020-CLG484-1
FPGA part	Artix-7 100MHz
CPU part	ARM Cortex-A9 667MHz
Memory	DDR3 512MB
OS	Linaro Ubuntu 14.04.3 LTS 32bit
Compiler	GNU Fortran 4.8.4, gcc 4.8.4
Number of grids (R×Z)	$271 \times 311 = 84281$
Number of particles	151500
Number of steps	250

B. Execution Time

First of all, the execution time of the off-loaded part is evaluated. Here, the number of grids is 84281, and a data element is represented with the 8-byte. For each grid, 23 reads and writes are required in total in the "field_source". The DDR memory bandwidth on Zynq is 4.2GB/s, that is, the theoretical processing time of "field_source" is as below.

$$84281 \times 8 \times 23 \div 4.2 \times 10^{-3} = 3692.31\mu s$$

Likewise, 51 reads and writes are required in the "particle_att_ion" and that is 28 in the "particle_att_ele". Calculated theoretical processing times and the measured execution times are shown in Table. II.

TABLE II
The execution time (μsec)

	field_source	particle_att_ion	particle_att_ele
CPU	42300.94	140679.94	90325.72
FPGA	4960.26	N/A	N/A
theoretical value	3692.31	8187.30	4494.99

Compared with the theoretical processing time, when we pay attention to "field_source", it appears that the implementation uses about 74.4% of the ideal DDR memory bandwidth, which is close to the practical limitation. Although the design of the remaining parts is ready, the execution time we could measured on the real board is that of "field_source" because of a lack of resources in the current Zed board. XC7Z020 on the current Zed board has only small FPGA logics, and we could not off-load all of SOURCE1 on the board. Thus, we estimated the time of remaining part assuming that the similar memory bandwidth of "field_source" can be used for other parts.

Compared with ARM Cortex-A9 667MHz, the off-loaded part "field_source" achieved about 8.53 times performance. However, considering the communication overhead, the performance improvement of the whole SOUCE1 is about 1.14 times when only "field_source" is off-loaded. On the other hand, as shown in Fig. 11, "particle_att_ion(ele)" achieved 12.78 and 14.95 times performance, respectively assuming the same memory bandwidth as the case of "field_source". Overall execution time of SOURCE1 can achieve 5.17 times performance if three subroutines could be off-loaded to the FPGA part.

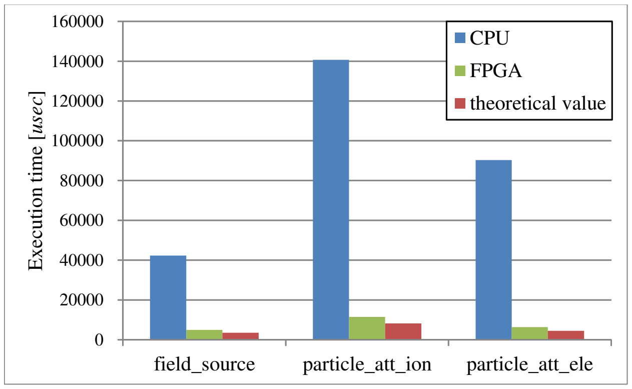


Fig. 11. The execution time which includes estimated value

C. Resource Utilization

The resource utilization ratio of the actual implementation is shown in Table III, and the breakdown results of the resource for "field_source", "particle_att_ion" and "particle_att_ele" are shown in Table IV, V, and VI. As shown in below tables, the utilization of BRAM in Table III is remarkably high, because buffer modules for stencil computations are implemented in the overall design.

TABLE III
The total resource utilization ratio (%)

	FF	LUT	BRAM	DSP48
total	55.4	71.7	87.1	44.6

TABLE IV
The breakdown of resource for field_source (%)

	FF	LUT	BRAM	DSP48
first loop	18.2	12.4	0	30.5
second loop	1.9	0.95	0	9.1
third loop	8.9	6.3	0	5.0
field_source	29.0	19.7	0	44.6

TABLE V
THE BREAKDOWN OF RESOURCE FOR PARTICLE_ATT_ION (%)

	FF	LUT	BRAM	DSP48
CIC	25.4	55.0	0	44.5
boundary processing	9.3	11.7	0	25.0
interpolation	3.0	5.8	0	7.7
particle_att_ion	37.7	72.5	0	77.2

TABLE VI
THE BREAKDOWN OF RESOURCE FOR PARTICLE_ATT_ELE (%)

	FF	LUT	BRAM	DSP48
CIC	21.2	50.0	0	44.5
boundary processing	4.5	8.9	0	22.0
interpolation	1.4	3.5	0	6.4
particle_att_ele	27.1	62.4	0	72.9

Since the PL (FPGA) part of XC7Z020 is rather small, we must use larger Zynq chip for this application. The available resource of Zynq is shown in Table VII.

TABLE VII
THE RESOURCE OF ZYNQ

	FFs	LUTs	extensible BRAM	DSP Slices
XC7Z020	106400	53200	560 KB	220
XC7Z045	437200	218600	2180 KB	900

According to the above table, XC7Z045 adopted by Xilinx ZC706 board has much resource than XC7Z020. Thus, we can implement all of the modules we developed in this study.

VI. CONCLUSION

A Full-PIC simulation code for Hall thruster used in space crafts was implemented with software/hardware co-operation on a simple embedded board with Zynq for future stand-alone simulation in a satellite.

The complete NSRU-Full-PIC code was ported on a Zed Board with Linux OS. By the profiling of the running code, the time consuming step SOURCE1 was selected as a target of off-loading and three subroutines, "field_source", "particle_att_ion" and "particle_att_ele" were implemented on the PL (FPGA) part by using HLS. 2×2 stencil computation module was pipelined with buffer modules. The off-loaded "field_source", "particle_att_ion" and "particle_att_ele" achieved 8.53 times, 12.78 times and 14.95 times performance as the software execution,respectively. The total execution time of SOURCE1 becomes 5.17 times of Cortex-A9 667MHz in Zynq.

The result of our study suggests that using FPGA is more suitable to Full-PIC method than using GPU. On the other hand, for off-loading all subroutines, the current Zed board is not enough. We are now trying to reduce the required hardware amount as well as preparing a powerful test board with a large size Zynq ZC7Z045.

REFERENCES

[1] "Zynq-7000 All Programmable SoC, Xilinx Inc. ," http://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html.

[2] S. Cho, K. Komurasaki, and Y. Arakawa, "Kinetic particle simulation of discharge and wall erosion of a hall thruster," *Physics of Plasmas*, vol. 20, no. 6, 2013. [Online]. Available: http://scitation.aip.org/content/aip/journal/pop/20/6/10.1063/1.4810798

[3] "SystemA (Camphor) — Super Computer System — Institute for Information Management and Communication, Kyoto University ," http://www.iimc.kyoto-u.ac.jp/ja/services/comp/supercomputer/.

[4] T. Miyajima, S. Cho, and N. Fujita, "A study of gpu acceleration of "source" part in hall-thruster simulation," in *IEICE Tech. Rep.*, ser. CPSY2015-62, vol. 115, no. 342, Dec. 2015, pp. 7–12.

[5] "PIConGPU - A Many-GPGPU Particle-in-Cell Code - Helmholtz-Zentrum Dresden-Rossendorf, HZDR ," https://www.hzdr.de/db/Cms?pNid=3227.

[6] P. Moorthy and N. Kapre, "Zedwulf: Power-Performance Tradeoffs of a 32-Node Zynq SoC Cluster," in *IEEE 23rd Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, May 2015, pp. 68–75.

[7] N.Sugimoto, T.Miyajima, Y.Osana, N.FUjita and H.Amano, "Zynq Cluster for CFD Parametric Survey," in *International Symposium on Applied Reconfigurable Computing*, March 2016.

[8] A. Doumar and H. Ito, "Detecting dianosing and tolerating faults in SRAM-based field programmable gate arrays: A Survey," *IEEE Transactions on VLSI Systems*, vol. 11, no. 3, pp. 386–405, 2003.

[9] D. Fay , A. Shye, S.Bhattacharya, D.Connors, and S.Wichmann, "An Adaptive Fault-Tolerant Memory System for FPGA-based Architecutures in the Space Environment," *Second NASA/ESA Conference on Adaptive Hardware and Systems*, pp. 250–257, 2007.

[10] M.Ibrahim, A.Tobal, M.Nahas, M.Refai, "FPGA Based on Board Computer for LEO Satellites," in *IEEE International Conference on Space Science and Communication*, 2011, pp. 314–319.

[11] E.Hartley, J.Maciejowski, "Predictive control fro spacecraft rendezvous in an elliptical orbit using an FPGA," in *European Control Conference*, 2013, pp. 1359–1364.