PGR : A SOFTWARE PACKAGE FOR RECONFIGURABLE SUPER-COMPUTING

Tsuyoshi Hamada * *and Naohito Nakasato* †

Computational Astrophysics Laboratory
The institute of physical and chemical research (RIKEN)
2-1, Hirosawa, Wako, Saitama 351-0198, Japan
email: thamada@riken.jp, nakasato@riken.jp

ABSTRACT

In this paper, we describe a methodology for implementing FPGA-based accelerator(FBA) from a high-level specification language. We have constructed a software package specially tuned for accelerating particle-based scientific computations with an FBA. Our software generates (a)a suitable configuration for the FPGA, (b) the C source code for interfacing with the FBA, and (c) a software emulator. The FPGA configuration is build by combining components from a library of parametrized arithmetic modules; these modules implement fixed-point, floating-point and logarithmic number system with flexible bitwidth and pipeline stages. To make certain our methodology is effective, we have applied our methodology to acceleration of astrophysical N-body application with two types of platforms. One is our PROGRAPE-3 with four XC2VP70-5 FPGAs and another is a minimum composition of CRAY-XD1 with one XC2VP50-7 FPGA. As the result, we have achieved peak performance of 324 Gflops with PROGRAPE-3 and 45 Gflops with the minimum CRAY-XD1, sustained performance of 236 Gflops with PROGRAPE-3 and 34 Gflops with the CRAY-XD1.

1. INTRODUCTION

Recently, scientific computation using FPGAs begins to be promising and attract a several group of astrophysicists. Previously, many works related to scientific computation using FPGAs have been reported and, roughly speaking, those works can be classified into following three subjects; (A) hardware implementations, (B) construction of arithmetic modules and (C) programming technique for generating a computing core for FPGAs. In this work, we present results of our current efforts for each of those subjects. Specifically, we have constructed a software package specially tuned for accelerating scientific computations with FPGA-based systems. Although we will show details of our methodology in the following sections, first we briefly describe each subject as follows.

(A) **Hardware Implementations**: In the first subject of hardware implementations, PROGRAPE-1(PROgrammable GRAPE-1)[1] is an earliest example of the use of a reconfigurable hardware for scientific computation. PROGRAPE-1 is an FPGA-Based Accelerator(FBA) for astrophysical many-body simulations. It is implemented with two Altera EPF10K100 FPGAs. Comparing with modern FPGAs, the EPF10K100 is old-fashioned and has only 100k gates per one chip. On the PROGRAPE-1 board, they have implemented a computing core that calculates gravitational force $\mathbf{a}_i$ of i-th particle exerted by all other particles used in astrophysical many-body simulations:

$$\mathbf{a}_i = \sum_j \frac{m_j \mathbf{r}_{ij}}{(r_{ij}^2 + \varepsilon^2)^{3/2}} \tag{1}$$

where $\mathbf{r}_{ij} = \mathbf{r}_j - \mathbf{r}_i$ is a relative vector between i and j-th particles, m_j is mass of j-th particle, and ε is a softening parameter that prevents a zero-division. They have obtained peak throughput performance of 0.96 Gflops for this calculation. An essential point of the PROGRAPE-1 is using fixed-point and short-length logarithmic number formats instead of the IEEE standard floating-point (FP) format. In the processor core of PROGRAPE-1, they have adopted 20-bit fixed-point format in a first stage for subtraction, 14-bit logarithmic number format(7-bit exponent, 5-bit mantissa, sign flag and non-zero flag) in inner stages for division and square root etc., and 56-bit of fixed-point format in a last stage for summation. That is though available resource of FPGA systems is limited, FBA systems can be attractive and competitive if an application does not need high accuracy such as double precision that is used in general computing with conventional CPUs. After the PROGRAPE-1, a several group have reported similar FBA systems ([2][3][4]). In all of those works, they have used HDL as system programming language for constructing a computing core and mentioned the need of automation the task of programming. We will show our answer to such demands in this paper. To prove our methodology described in this paper to be correct,

*This work is supported by the Exploratory Software Project 2004,2005 of Information-Technology Promotion Agency, Japan.

†Special Postdoctoral Researcher at RIKEN.

we have developed new FBA board (PROGRAPE-3 board in Figure 4) with modern FPGAs as a target hardware.

(B) **Arithmetic Modules**: In the second subject, a main task is to construct a library of parametrized arithmetic modules (AM). For example, [5][6] have presented parametrized FP adders and multipliers, and [2][7] presented parametrized FP square-roots and divisions. However, even if these basic AMs are existed, it is insufficient to construct a computing core for scientific computations. In most of those works, square root and divisions are implemented using subtracters and shifters. This approach is not suitable especially in pipelined data-path, because the carry propagation of ripple-carried subtracter becomes long. Instead of standard algorithm above, using Function Evaluation Units(FEUs), which calculate arbitrary functions using some approximation methods, is efficient in most cases. One can use FEUs for calculating arbitrary functions such as $\sqrt{x}, \log(x), \exp(x)$ that are commonly appeared in scientific computations. In implementing a FEU, one can have three approximation methods such as a table look-up method (0-th order), a polynomial approximation method (n-th order), or the hybrid method[8][9]. With HDL that is static in its nature except generic parameters such as `GENERIC` or `ATTRIBUTE`, implementation of a general FEU that support arbitrary functions, different methods, and variable precision is highly difficult task. To solve this problem, one can construct a software that dynamically generate HDL description for a FEU. The approach of dynamic generation is applicable for generating not only FEUs but general FP-AMs that is also better to support variable precisions. Here in this work, we have constructed such software for our purpose. At run-time, by selecting a desired functions and an approximation method (in case of the FEU generation) or desired precision and number format (in case of the AM generation), our software generate a corresponding module. Details will be described in Section 2.

(C) **Generation of a Computing Core**: Even if the FP-AMs are existed as libraries or generated from a software, one needs a deep understanding of details of such AMs to construct a computing core for their purpose. The task in the third subject is to solve the this issue of generating a computing core using the AMs. A real concern here is how to convert a scientific application written in a high-level language such as C, Java or C++ into an FPGA configuration.

PAM-Blox[10] and JHDL[11] are first examples to allow such conversion from C++ and Java, respectively. The PAM-Blox concentrates on automation and simplicity to explore the design space. The JHDL treats circuits as objects and but has only a few elemental FP-AMs so that it seems not to be interested in scientific computations so much. In both work, authors have emphasized the importance of automatic generation of APIs that is needed to communicate between FBAs and a host processor. In general, an software between the FBA and the host processor should be implemented to maximize data transfer rate and minimize latency. Even if a smart tool can generates an FPGA configuration of a computing core, one can't necessarily write such high performance communication software.

As well as communication software, the performance of a particular application is very sensitive to detailed *architecture* of a computing core. Here, we consider a computing core consists of inner and outer core. Specifically, in astrophysical many-body simulations, one can think an inner core is a calculation unit of gravitational force between two particles shown in eq. (1) and an outer core is a memory unit that feeds data of two particles to the inner core and fetch results from the inner core. An obvious implantation of such core is that the outer core feeds position and mass of $i-$th and $j-$th particles, and and fetch partial force in each step of the summation. This implementation is most flexible but worst efficient in terms of data transfer. In the PROGRAPE-1 (and other family of GRAPE[12]), the computing core is implemented such that (a) data of $i-$th particle is stored inside the inner core (b) the inner core accumulate partial sums inside and (c) the outer core feeds $j-$th data continuously and fetch only the accumulated force a_i. Clearly, even with a same inner core, performance can change drastically depending on detailed *memory architecture* of an outer core.

In the PAM-Blox[10], authors have introduced an important concept of *domain specific compiler* [13]. In such domain specific compiler, a promising application is limited by the compiler. The point is that there is no almighty architecture for any problem. Specifying the application domain produces good results for the tool developer and the tool users. For the tool developer, the language specification becomes compact so it can be easy to implement a compiler that specializes to an application domain if once the module generator has been completed as a core component of a programming tool. For the tool users, it becomes clear whether the tool agrees with a purpose and easy to understand such a compact language specification. Because PAM-Blox seems very wonderful tool, we feel they might appeal more positively and very regrettable not to try to accelerate scientific computations such as gravitational many-body simulations.

For domains within scientific many-body simulations, we have developed PGR(Processors Generator for Reconfigurable system) package. Our purpose is to put the FBA design working under *user*'s control. Here, we consider scientists in physics, astrophysics, chemistry or bioscience as the target *user*. Our methodology doesn't target specialists in electronics. In contrast, previous design methodologies seem to concentrate on hardware developers. Our methodology realizes following requirements that is not sufficiently satisfied by the previous works.

(1) Target user only has to write a description as short as possible to realize user's requirements.

(2) All of the hardware configurations, software emulator, communication libraries are generated by some clever tool.

(3) At least one or more hardware platforms are actually available.

(4) The absolutely high performance is shown on a hardware platform.

The remainder of this paper is organized as follows. Section 2 gives an overview of our methodology. In section 3, we show implementation results for astrophysical N-body simulations using PGR package. Section 4 is devoted to comparison to relevant works. Finally we conclude this paper in section 5.

2. PGR : PROCESSORS GENERATOR FOR RECONFIGURABLE SYSTEMS

2.1. Design flow in PGR

Figure 1 shows a schematic outline of design flow of PGR package. As a first step, a user must write a source code for the user's application in PGR Description Languages(PGDL; described in Section 2.4). Specifically, the PGDL source code define dataflow for the application and variable names used for generating interface library to a FBA. Inside PGR package, four components (parser, module generator, dataflow generator and compiler) are doing real jobs as following way;

stage 1 the parser analyzes the PGDL source code and outputs results as an internal expressions used in the following stages.

stage 2 the module generator convert arithmetic modules definitions into VHDL source codes and source codes of software emulator for each arithmetic modules.

stage 3 the dataflow generator compute the dataflow graph while it inserts delay register between modules.

stage 4 the compiler generates final outputs such as top-level VHDL code, VHDL library code for arithmetic modules, and source code of the library interface and software emulator.

Note that the software emulator and the interface library source codes are written in C and both have a special top-level function with exactly same arguments each other. Namely, one can easily switch between software emulation and real runs on the FBA by liking one of both source code. After the user satisfied with emulation results, the user finally synthesize the VHDL code using a CAD software and setup a FBA using the obtained configuration. Then, the user can test performance of the processors running on the FBA.

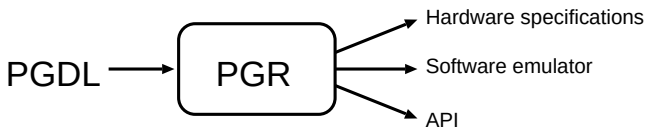

Fig. 1. Design flow of PGR package

2.2. PgModules : PGR Parametrized Arithmetic Modules

PgModules(PGR parametrized arithmetic modules) implement fixed-point, FP and logarithmic number system (LNS) AMs and are the most low-level components for PGR package. These modules include addition, subtraction, multiplication, division, and square-root, etc. Currently, PGR package supports 29 parametrized AMs as shown in table 1.

In this table, modules with `pg_float` correspond to FP arithmetics. We define internal floating-point format with 1-bit for a sign flag, 1-bit for a non-zero expression, m-bit for exponent, and n-bit for mantissa, here after we will use FP_{m+n+2}(m,n) as the notation for this format [1]. The m and n can be changed arbitrary up to FP_{33}(8, 23) which corresponds to the IEEE single precision.

For example, in the PGDL, we can generate an FP_{26}(8,16) addition module as follows;

```
pg_float_add(x,y,z,26,16,1);
```

This example calculates $z = x + y$. Here, the arguments `x` and `y` are the inputs, the output is `z`. And `26` and `16` express total and mantissa bitwidth of FP_{26}(8,16)

And the last, `1` specifies the number of pipeline stages of this module. Note that for the rounding operation, we have implemented nine types of several options that also can be changed by a hidden argument in the PGDL. If this argument is omitted like above example, a rounding to the nearest even is selected.

Modules `pg_fix_addsub` and `pg_fix_accum` are FIX adder/subtracter and accumulator, respectively. Moreover, modules `pg_log_muldiv` and `pg_log_add` are LNS multiplier/divider and adder, respectively.

In the format of LNS, a positive, non-zero real number x is represented by its base-2 logarithm y as $x = 2^y$. The LNS is useful because operation such as multiplication and square root are easier to implement than in the usual FP format. For more details of the LNS, see GRAPE-5 paper[14].

In tables 2, 3, we show resource consumption and clock frequency of FP square root and LNS unsigned add AMs. Despite we have implemented all of the parametrized AMs from full scratch, the obtained performance results are almost same as other implementations such as [2].

[1] LNS_{m+n+2}(m,n) is notation for the logarithmic format with 1-bit sign, 1-bit nonzero, m-bit exponent and n-bit mantissa. FIX_n is notation for the signed fi xed-point format with n-bit.

Table 1. List of PgModules

fbating-point format	
pg_float_add	$+$
pg_float_unsigned_add	$+$
pg_float_sub	$-$
pg_float_unsigned_sub	$-$
pg_float_mult	$\times$
pg_float_div	$/$
pg_float_sqrt	$\sqrt{x}$
pg_float_square	x^2
pg_float_recipro	x^{-1}
pg_float_expadd	$x \cdot 2^{\pm N}$
pg_float_negate	$-x$
pg_float_compare	$==$
pg_float_compare_abs	$==$
pg_float_compz	> 0
pg_float_compz_abs	> 0
pg_float_accum	$+=$
pg_float_unsigned_accum	$+=$
pg_float_fixaccum	$+=$
fi xed-point format	
pg_fix_addsub	$+, -$
pg_fix_mult	$\times$
pg_fix_unsigned_mult	$\times$ (unsigned)
pg_fix_accum	accumulate
LNS format	
pg_log_add	$+$
pg_log_unsigned_add	$+$ (unsigned)
pg_log_muldiv	$\times, /$
pg_log_shift	$\sqrt{x}, x^2$
pg_log_expadd	$x \cdot 2^{\pm N}$
format conversion	
pg_conv_fixtofloat	fi x $\Rightarrow$ fbat
pg_conv_floattofix	fbat $\Rightarrow$ fi x
pg_conv_ftol	fi x $\Rightarrow$ log
pg_conv_ltof	log $\Rightarrow$ fi x

In some relevant works, Ho et.al.[15] has used Symmetric Table Addition Method(STAM)[16] for their FEUs. In the STAM, the multiplications for polynomial are replaced by adders and tables. This method is suitable for old-fashioned FPGAs which have no embedded multiplier. On the other hand, in PgModules we use the first order or second order Chebyshev polynomial to take advantages of embedded multipliers.

To show effectiveness of our approach, figure 2 shows the resources consumption as a function of mantissa bitwidth of LNS unsigned adder mapped on the Xilinx XC2VP FPGA. In this figure, "TABLE" and "TABLE+POLY" show results using only look-up table and sectional polynomial approximation with look-up tables, respectively. The merit of "TABLE+POLY" is that we can reduce address bitwidth for look-up table, in compensation for extra additions and multiplications. It is found that the tradeoff point exists at 5-bit in the case of our LNS unsigned adder.

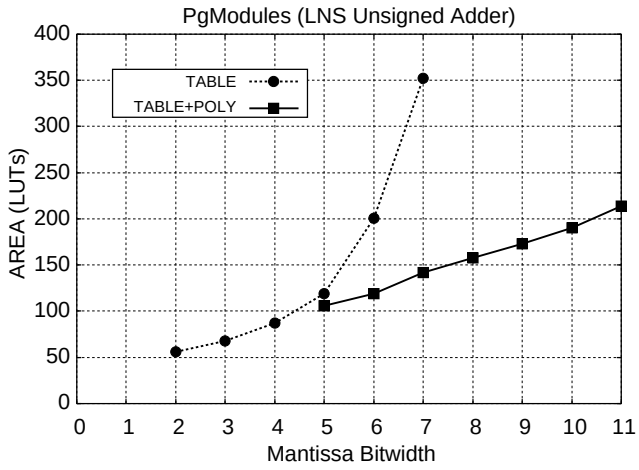

Fig. 2. TABLE vs TABLE+POLY

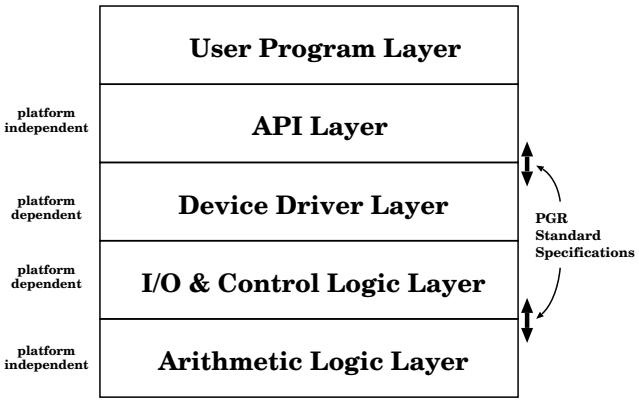

Fig. 3. PGR five layers model.

2.3. PGR five layers model

To make PGR software independent on a specific hardware, we create PGR five layers model which divide an FBA into five parts. Figure 3 shows PGR five layers model, and it's composed of User Program Layer(UPL), API Layer(APL), Device Driver Layer(DDL), I/O & Control Logic Layer(ICL) and Arithmetic Logic Layer(ALL).

The UPL is a user application which communicates with an FBA through the APL. The APL contains the top level API implementations that doesn't depend on an individual FBA. The DDL consists of both a low level communication library and a device driver software. The ICL is a glue logic such as the PCI interface logic and local I/O logic on an FBA. The ALL corresponds to a *computing core* explained in Section 1 and is composed of AMs and control logics.

2.4. PGDL: PGR Description Language

In this subsection, we illustrate how pipeline processors(or computing cores) are described in the PGDL and how such

Table 2. FP Square Root

exp.	mant.	stages	MHz	slices
8	8	5	215.517	86
		4	157.754	71
		1	79.971	51
8	16	5	188.964	140
		3	127.959	116
		1	56.500	84
8	23	5	141.243	425
		3	104.998	392
		1	70.210	374

Table 3. LNS Unsigned Adder

exp.	mant.	stages	MHz	slices
7	5	5	201.077	94
		3	151.207	72
		1	64.545	57
7	8	6	195.369	116
		4	156.961	100
		1	58.828	86
20	11	7	218.293	191
		4	148.721	125
		1	53.562	115

description is converted into HDL sources for pipelined processors.

With the current version of PGR package, it is specially tuned to generate pipelined processors for particle-based simulations as explained already. It is expressed as the following summation form:

$$f_i = \sum_{j=1}^{n} F(p_i, p_j), \tag{2}$$

where f_i is summation for i-th data, p_i are some values associated with i-th data, and F expresses calculations where i-th and j-th data are as inputs.

As an example target, we consider the following artificial calculations;

$$f_i = \sum_{j}^{n} a_i a_j. \qquad (i = 1, ..., n) \tag{3}$$

Figure 5 shows the block diagram for this example. Here, a_i and a_j are scalar values for i-th and j-th elements, respectively. This target simply calculates a product of a_i and a_j, and sums the product up for all j.

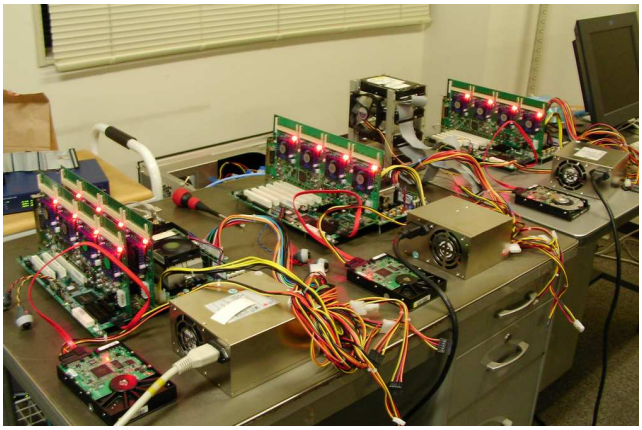

Fig. 4. PROGRAPE-3 prototype system: Four Xilinx XC2VP70 FPGA devices are mounted on single board. We use Opteron 2.4GHz machines as host computers and the PCI64 interface between the host and PROGRAPE-3 board.

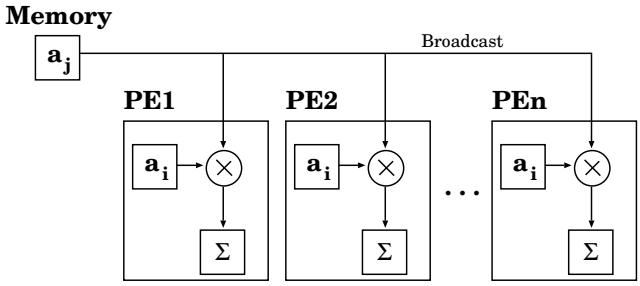

Fig. 5. Block diagram of the example processors(PEs).

Figure 6 shows the PGDL description of this target function. In this example, one already sees essential ingredients of an FBA: data, their representation, functional form of arithmetic operations between data i and j. The lines 1 and 2 define the bit-length as FP_{26}(8,16). These definitions are actually used in the next block (lines 3, 4 and 5 starting with "/"), which defines the layout of registers and memory unit. For the data a_i (and a_j), we use FP_{26}(8,16) format. The line "/NPIPE" specifies a number pipeline processors (10 processors in this case). The final part describes the target function itself using parametrized AMs. It has C-like appearance, but actually defines the hardware modules and their interconnection.

From such PGDL description, the following ALL (as shown in figure 5) is generated; the i-th data is stored in the on-chip memory, and new data (j-th data) is supplied at each clock cycle. The i-th data is unchanged during one calculation, and the result (f_i) is stored in the register.

```
1 #define NFLO 26
2 #define NMAN 16
3 /JPSET x,  aj[], float, NFLO, NMAN;
4 /IPSET y,  ai[], float, NFLO, NMAN;
5 /FOSET z,  fi[], float, NFLO, NMAN;
6 /NPIPE 10;
7 pg_float_mult  (x,   y,  xy, NFLO, NMAN, 1);
8 pg_float_accum (xy,  z,      NFLO, NMAN, 1);
```

Fig. 6. An example of design entry file written in PGDL

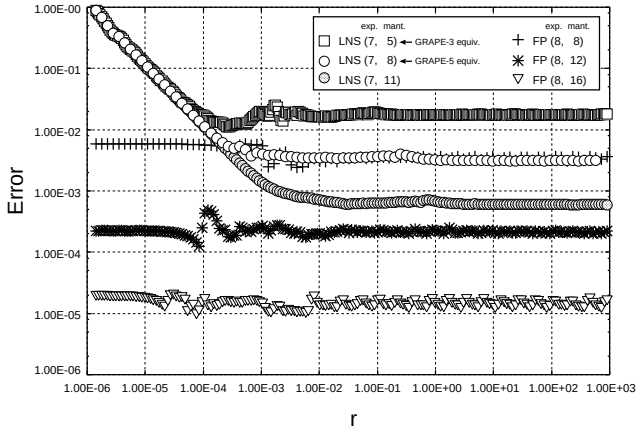

Fig. 7. Pair-wise relative error for gravitational force calculation in the N-body problem.

3. APPLICATION

To show the possibility of PGR package, we have implemented gravitational force (as shown in eq. (1)) pipeline for astrophysical many-body simulations Figure 9 and 10 show PGDL descriptions for this gravitational force pipeline using FP_{26}(8,16) operations and LNS_{17}(7,8) operations, respectively. Note there are a several differences between two descriptions. We check accuracy and resource consumption of the different implementations to test whether PGR package generates effective implementations. In Figure 7, we present relative error($= \frac{|\vec{f_{\mathrm{host}}} - \vec{f_{\mathrm{fba}}}|}{|\vec{f_{\mathrm{host}}}|}$, where $\vec{f_{\mathrm{host}}}$ is the double precision result, $\vec{f_{\mathrm{fba}}}$ is the result by FBA) of our implementations. And Figure 8 shows their resource consumptions.

Figure 11 shows a data flow graph that corresponds to the FP pipeline. PGR package automatically inserts delay register, which are indicated by bold circles, for synchronizing each operation.

All of our implementations have been correctly working at 133.3MHz on PROGRAPE-3 and 120 MHz on CRAY-XD1[2]. Here, *correctly working* means that gravity force calculated by both of PROGRAPE-3 and CRAY-XD1 for 16384 particles is exactly same with results obtained by software emulator. In table 4, we compare two implementations

[2]For PROGRAPE-3, we use Synplify Pro for a backend tool of PGR package. For CRAY-XD1, the backend tool is Xilinx ISE6.3i.

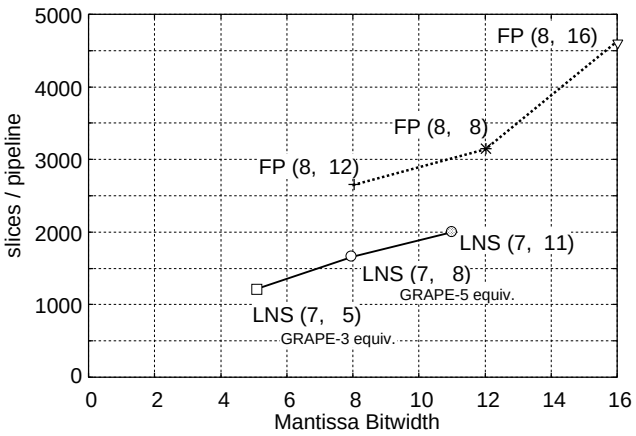

Fig. 8. Area comparison of gravitational N-body implementations.

of FP_{26}(8,16) and LNS_{17}(7,8) with the GRAPE-5 system[14]; actually, the LNS_{17}(7,8) pipeline is actually equivalent to the GRAPE-5. The peak performance of PROGRAPE-3 is 324.2 Gflops and 45.6 Gflops for the minimum composition of CRAY-XD1 [3].

Here, the number of floating-point operations per one interaction is 38[12]. The peak performance of the LNS pipelines on single PROGRAPE-3 board is five times faster than single GRAPE-5 board. And even if we use twice accuracy (i.e., the FP_{26}(8,16) pipelines), the performance is the still two times better than single GRAPE-5.

Finally, figure 12 shows the sustained performance of single PROGRAPE-3 board and minimum composition of CRAY-XD1 with the LNS implementation as a function of number of particles N. Here, we show the results for the direct-summation algorithm. Because the calculation cost and data transfer cost scale $\propto N^2$ and N respectively, the sustained performance gradually approaches the peak throughput as the number of particles increases.

4. COMPARISON TO RELEVANT WORKS

In scientific computations on FBAs, type conversions and scaling operations are confusing and designing APIs becomes troublesome inevitably.

We note that a similar package to PGR has been reported in [15]. The CAST(Computer Arithmetic Synthesis Tool) is a tool for implementing astrophysical many-body simulations. We concern that the CAST seems not to meet the requirements (2,4,5,6) in our introduction. On the other hand, using PGR package, possible users, who want to accelerate their particle simulations with an FBA, can concentrate on only writing a PGDL code. That is PGR package can drastically reduce the amount of work for such user.

[3]The result of CRAY-XD1 is just preliminary.

Table 4. Implementation result and comparison with other implementation

	GRAPE-5	PROGRAPE-3		CRAY-XD1
Device	ASIC	FPGA(XC2VP70-5)		FPGA(XC2VP50-7)
Device technology	0.5 μm	0.13 μm		0.13μm
Chips/board	8	4		1
Format :input	FIX_{32}	FIX_{32}	FP_{26}(8,16)	FIX_{32}
Format :internal	LNS_{17}(7,8)	LNS_{17}(7,8)	FP_{26}(8,16)	LNS_{17}(7,8)
Format :accumulation	FIX_{64}	FIX_{64}	FIX_{64}	FIX_{64}
Pair-wise error	$10^{-2.4}$	$10^{-2.4}$	$10^{-4.8}$	$10^{-2.4}$
PEs/chip	2	16	6	10
Frequency (MHz)	80	133	133	120
Peak Gflops/board	48.6	324.2	121.6	45.6

```
/* ---------------------------------------- MACRO */
#define NFLO  26
#define NMAN  16
#define NFIX  57
#define NACC  64
#define NEXAD 39
#define FSCALE (pow(2.0,NEXAD))
/* ---------------------------------- API DEFINITION */
/JPSET xj[3], x[][],  float (NFLO, NMAN);
/JPSET mj,    m[],    float (NFLO, NMAN);
/IPSET xi[3], x[][],  float (NFLO, NMAN);
/IPSET e2,    eps2,   float (NFLO, NMAN);
/FOSET sx[3], a[][],  fix   (NACC);
/CONST_FLOAT  fshif,  FSCALE, NFLO,NMAN;
/SCALE sx : -1.0/(FSCALE);
/NPIPE 6;
/* ---------------------------------------- PIPELINE */
pg_float_sub(xi,xj,dx,    NFLO,NMAN,              4);
pg_float_mult(dx,dx, x2,  NFLO,NMAN,              2);
pg_float_unsigned_add(x2[0],x2[1],xy, NFLO,NMAN,  4);
pg_float_unsigned_add(x2[2],e2,ze,    NFLO,NMAN,  4);
pg_float_unsigned_add(xy,ze,r2,       NFLO,NMAN,  4);
pg_float_sqrt(r2,r1,      NFLO,NMAN,              3);
pg_float_mult(r2,r1,r3,   NFLO,NMAN,              2);
pg_float_recipro(r3,r3i,  NFLO,NMAN,              2);
pg_float_mult(r3i,mj,mf,  NFLO,NMAN,              2);
pg_float_expadd(mf,mfo,   NEXAD,NFLO,NMAN,        1);
pg_float_mult(mfo,dx,fx,  NFLO,NMAN,              2);
pg_float_fixaccum(fx,sx,  NFLO,NMAN,NFIX,NACC,    4);
```

Fig. 9. a PGDL for gravitational force calculation (using 26-bit FP arithmetics)

```
/* ---------------------------------------- MACRO */
#define NPOS 32
#define NLOG 17
#define NMAN 8
#define NCUT 6
#define NFOR 57
#define NACC 64
#define NEXAD -31
#define fshift pow(2.0,-1.0*(double)NEXAD)
/* ---------------------------------- API DEFINITION */
/JPSET xj[3], x[][], fix,       FIX (NPOS);
/JPSET mj,    m[],   log,       LNS (NLOG,NMAN);
/IPSET xi[3], x[][], ufix,      FIX (NOPS);
/IPSET ieps2, eps2,  log,       LNS (NLOG, NMAN);
/FOSET sx[3], a[][], fix,       FIX (NACC);
/SCALE mj : pow(2.0,95.38);
/SCALE sx : -(fshift)/(mj);
/NPIPE 16;
/* ---------------------------------------- PIPELINE */
pg_fix_addsub(SUB,xi,xj,xij,           NPOS,            1);
pg_conv_ftol(xij,dx,                   NPOS,NLOG,NMAN,  2);
pg_log_shift(1,dx,x2,                  NLOG);
pg_log_unsigned_add_itp(x2[0],x2[1], x2y2,   NLOG,NMAN, 4,NCUT);
pg_log_unsigned_add_itp(x2[2],ieps2, z2e2,   NLOG,NMAN, 4,NCUT);
pg_log_unsigned_add_itp(x2y2,z2e2,   r2,     NLOG,NMAN, 4,NCUT);
pg_log_shift(-1,r2,       r1,          NLOG);
pg_log_muldiv(MUL,r2,r1,  r3,          NLOG,            1);
pg_log_muldiv(SDIV,mj,r3,  mf,         NLOG,            1);
pg_log_muldiv(MUL,mf,dx,  fx,          NLOG,            1);
pg_log_expadd(fx,fxs,                  NEXAD,NLOG,NMAN, 1);
pg_conv_ltof(fxs,  ffx,                NLOG,NMAN,NFOR,  2);
pg_fix_accum(ffx,  sx,                 NFOR,NACC,       1);
```

Fig. 10. a PGDL for gravitational force calculation (using 17-bit LNS)

5. CONCLUSION

We have developed PGR package, a software which automatically generate communication software and the hardware descriptions (the hardware design of pipeline processors) for FBAs from a high-level description PGDL. Using PGR package, we have implemented gravitational force pipelines used in astrophysical many-body simulations. The PGDL description for the gravitational force pipelines is only a several tens of lines of a text file. Regardless of a very simple description, the gravitational force pipelines are implemented successfully and the obtained performance reaches 236 Gflops on our hardware PROGRAPE-3 and 34 Gflops on minimum composition of CRAY-XD1.

We specially acknowledge Jun Yatabe, Ryuichi Sudo and others at CRAY Japan for our access to CRAY-XD1 and technical helps.

6. REFERENCES

[1] T. Hamada, T. Fukushige, A. Kawai, and J. Makino, "PROGRAPE-1: A Programmable, Multi-Purpose Computer for Many-Body Simulations," *Publication of Astronomical Society of Japan*, vol. 52, pp. 943–954, 2000.

[2] G. L. Lienhart, A. Kugel, and R. Männer, "Using Floating Point Arithmetic on FPGAs to Accelerate Scientific N-Body Simulations," in *Proc. of IEEE Symposium on Field-Programmable Custom Computing Machines*, Apr. 2002, pp. 182–191.

[3] W. D. Smith and A. R. Schore, "Towards an RCC-based accelerator for computational fluid dynamics applications," in *Proc. of the 2003 International Conference on Engineering Reconfigurable Systems and Algorithms*, 2003.

[4] N. Azizi, I. Kuon, A. Egier, A. Darabiha, and P. Chow, "Reconfigurable Molecular Dynamics Simulator," in *Proc. of*

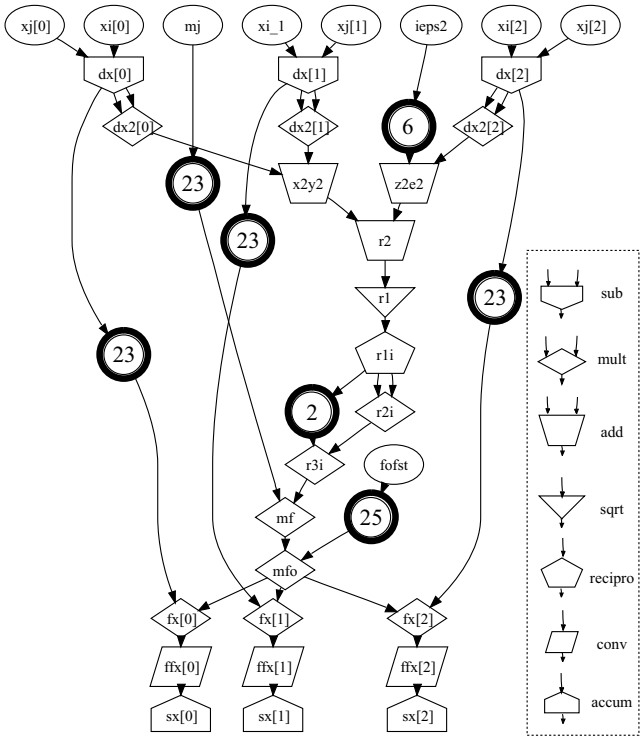

Fig. 11. Data flow of the gravitational force pipeline. The bold circles are delay registers which are automatically inserted by PGR package.

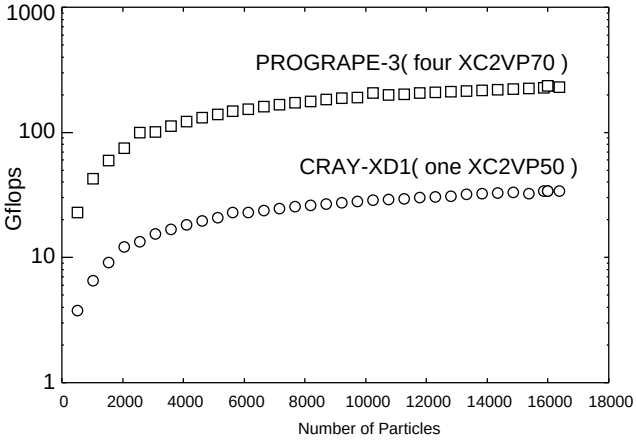

Fig. 12. The sustained calculation speed of single PROGRAPE-3 board in Gflops for the direct-summation algorithm, plotted as functions of the number of particles, N.

IEEE Symposium on Field-Programmable Custom Computing Machines, Apr. 2004, pp. 197–206.

[5] A. Jaenicke and A. Luk, "Parametrized Floating-Point Arithmetic on FPGAs," in *Proc. of IEEE ICASSP*, vol. 2, 2001, pp. 897–900.

[6] G. Leyva, G. Caffarena, C. Carreras, and O. Nieto-Taladriz, "A Generator of High-speed Floating-point Modules," in *Proc. of IEEE Symposium on Field-Programmable Custom Computing Machines*, Apr. 2004, pp. 306–307.

[7] X. Wang and B. E. Nelson, "Tradeoffs of Designing Floating-Point Division and Square Root on Virtex FPGAs," in *Proc. of IEEE Symposium on Field-Programmable Custom Computing Machines*, Apr. 2003, pp. 195–203.

[8] J. M. Flynn and F. S. Oberman, "Advanced Computer Arithmetic Design," in *John Wiley & Sons*, 2001.

[9] M. J. Muller, "Elementary Functions: Algorithms and Implementation," in *Brikhauser Verlag AG*, 1997.

[10] O. Mencer, M. Morf, and J. M. Flynn, "PAM-Blox: High Performance FPGA Design for Adaptive Computing," in *Proc. of IEEE Symposium on Field-Programmable Custom Computing Machines*, Apr. 1997, pp. 167–174.

[11] P. Bellows and B. Hutchings, "JHDL - An HDL for Reconfigurable Systems," in *Proc. of IEEE Symposium on Field-Programmable Custom Computing Machines*, Apr. 1998, pp. 175–184.

[12] J. Makino and M. Taiji, "Scientific Simulations with Special-Purpose Computers — The GRAPE Systems," in *John Wiley & Sons*, 1998.

[13] O. Mencer, M. Platzner, M. Morf, and J. M. Flynn, "Object-Oriented Domain-Specific Compilers for Programming FPGAs," 2001.

[14] A. Kawai, T. Fukushige, J. Makino, and M. Taiji, "GRAPE-5: A Special-Purpose Computer for N-body Simulation," *Publication of Astronomical Society of Japan*, vol. 52, pp. 659–676, 2000.

[15] K. Tsoi, C. Ho, H. Yeung, and P. Leong, "An Arithmetic Library and its Application to the N-body Problem," in *Proc. of IEEE Symposium on Field-Programmable Custom Computing Machines*, Apr. 2004, pp. 68–78.

[16] J. Stine and M. Schulte, "The symmetric table addition method for accurate function approximation," 1999, pp. 167–177.