Architecture of a Graphics Floating-Point Processing Unit with Multi-Word Load and Selective Result Store Mechanisms

Jiro Miyake†, Shigeo Kuninobu††, and Takaaki Baba†

†Graduate School of Information, Production and Systems, Waseda University, Kitakyushu-shi, 808-0135 Japan
††Information, Production and Systems Research Center, Waseda University, Kitakyushu-shi, 808-0135 Japan

Summary
We have proposed a graphics floating-point processing unit (G-FPU) with 48% reduction of hardware for a conventional processing unit that has both functions of a SIMD-type execution unit dedicated for multiply-accumulate operations and a general-purpose execution unit. The hardware reduction is obtained by realizing a dual-structured general-purpose execution unit that can handle both repeated operations of multiply-accumulate for geometry transformations and irregular operations such as ray-tracing in graphics processing with 9% increase in the hardware for a SIMD-type execution unit. To utilize multiple execution units that can operate in parallel, the high performance of data transfer is indispensable. Therefore, we have proposed a multi-word load mechanism and a selective result store mechanism to load and store data in parallel with executions. These mechanisms reduce the number of load/store instructions and achieve the high performance of data transfer required for parallel operations. Moreover, they remove a buffer memory of 7.9 K gates that temporarily stores data for executions. The effective data transfer reduces the processing cycles for intersection calculation by 26% and geometry transformation by 39%, compared with the case that conventional load/store instructions are used.

Key words:
Floating-point processor, Parallel operation, Data transfer, Graphics processing

1. Introduction

High performance for high quality and fast processing of computer graphics and small hardware for portable devices have been required for a floating-point processing unit. Generally, in a performance-oriented floating-point processing unit for graphics a single-instruction multiple-data (SIMD) architecture is adopted for the high-speed execution of multiply-accumulate operations such as geometry transformation [1][2]. The SIMD-type execution unit that performs the same instruction for multiple data in parallel is suitable for a geometry transformation, requiring four multiply-accumulate operations for input data repeatedly in parallel. However, since the SIMD-type execution unit cannot execute different operations in parallel, a general-purpose execution unit is added for irregular operations such as ray-tracing and shading processing.

Recently, high-performance general-purpose graphics processing units with many processor cores have been developed for workstations and personal computers that focus on programmability, flexibility, and high performance. These processing units achieve high performance by executing operations of multiple threads or multiple data with many processor cores in parallel [3][4].

The targets of our processing unit are low-cost consumer electric products, such as mobile products and personal audio-visual products. Consequently, it is difficult for our unit to implement many processor cores because of cost and power consumption constraints. Therefore, we have considered a general-purpose processing unit that can efficiently perform not only regular operations, such as multiply-accumulate, but also other irregular operations. By this, eliminating the SIMD-type execution unit, it can be widely applied to cost sensitive systems.

In graphics processing, it is necessary to implement plural execution units by taking account of the number of operations that can be executed simultaneously [5][6]. Moreover, to realize high performance and versatility, the following two mechanisms are necessary:

(1) A flexible parallel execution mechanism. It is necessary to have a mechanism to execute both repetition of multiply-accumulate operations and irregular operations efficiently by changing the structure of the execution units flexibly.

(2) A highly efficient data transfer mechanism. To achieve a high performance of data processing with plural execution units, a mechanism for the high performance of data transfer is indispensable to supply data for operations and to store execution results without delay.

In this paper, we describe a graphics floating-point processing unit (G-FPU) that can efficiently perform pipeline operations for repetition of multiply-accumulate operations, such as geometry transformation, keeping versatility for irregular operations, such as ray-tracing and shading processing. We propose new multi-word load and selective result store mechanisms that achieve a high

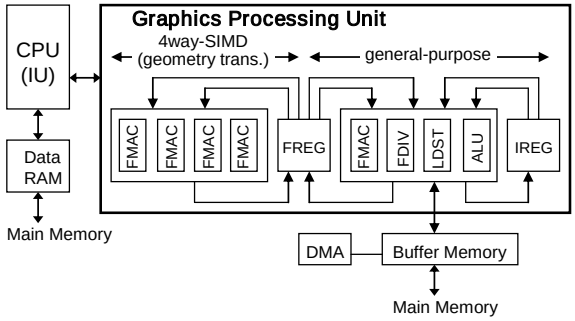
Manuscript received June 5, 2009
Manuscript revised June 20, 2009

performance of data transfer, reducing data transfer instructions.
We show examples of applying G-FPU in which data transfer for both repeated multiply-accumulate operations and irregular operations are performed efficiently.

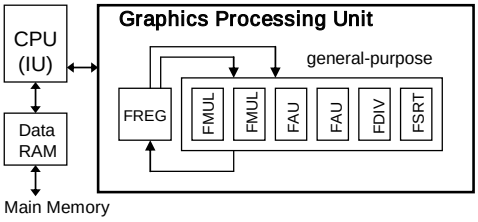
2. Graphics Floating-point processing unit (G-FPU)

2.1 Concept of G-FPU

Generally, a performance-oriented graphics processing unit has plural multiply-accumulate execution units controlled by SIMD-type instructions to execute operations, such as geometry transformation, at high speed. A geometry transformation is a product of a matrix and a coordinate vector and is calculated by repeating four multiply-accumulate operations. Therefore, a SIMD-type multiply-accumulate execution unit that can execute four multiply-accumulate operations in parallel by one instruction is suitable for a geometry transformation. Because repeated multiply-accumulate operations are executed in a pipeline manner with this execution unit, a high throughput can be achieved.



(a) Outline of a conventional graphics processing unit



(b) Outline of the proposed graphics processing unit

Fig. 1 Outline of a graphics processing unit..

The SIMD-type execution unit is suited for regular repeated operations. However, it cannot execute different operations in parallel. Because of this inflexibility, a general-purpose execution unit is added for irregular operations. Besides, a local buffer memory is often added to supply data for operations, and this data is transferred in parallel with execution using direct memory access (DMA). Fig.1(a) outlines a conventional graphics processing unit consisting of a SIMD-type execution unit and a general-purpose execution unit. A conventional processing unit like this one can achieve high execution performance, but its hardware is large.
To realize a graphics processing unit with small circuits, we eliminate the SIMD-type execution unit and adopt a structure that executes both regular repeated multiply-accumulate operations and irregular operations efficiently within the same execution unit. Furthermore, we remove the buffer memory that temporarily stores data for execution. Fig.1(b) outlines the proposed G-FPU.

2.2 Structure of the G-FPU

Fig.2 shows a SIMD-type execution unit and a general-purpose execution unit for parallel execution of three operations used in a conventional graphics processing unit.

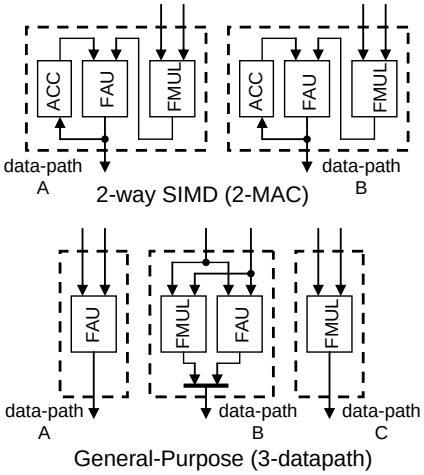


Fig. 2 SIMD-type and ge neral-purpose execution units used in a conventional graphics processing unit.

To use multiply-accumulate execution units for general-purpose processing, the structure shown in Fig.3(a) can be considered [7]. In this structure, there are two multiply-accumulate execution units that form two data-paths. The multiplier (FMUL) and adder (FAU) that compose each multiply-accumulate execution unit can be separated. To execute multiply-accumulate operations repeatedly, FMUL and FAU are connected serially and work in a pipelining manner. The results of multiply-accumulate are stored in the accumulation register (ACC). To make FMULs and FAUs versatile, two operations are selected arbitrarily from a group of two additions and two multiplications and then executed in parallel by separating FMULs and FAUs.

To execute more operations in parallel and remove the ACC, the structure shown in Fig.3(b) can be considered. Because each of two FMULs and two FAUs is used as an independent data-path, all of them can work in parallel. However, the ports for the register file increase to supply data to four data-paths from the register file and to store results to it.

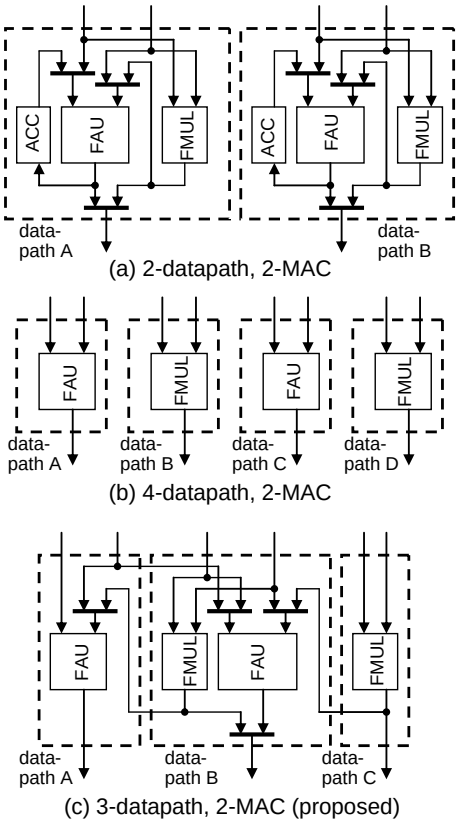


Fig. 3 Structure of execution unit for both multiply-accumulate and general-purpose operations.

We think that the degree of execution parallelism is not so high, about 2 or 3 in irregular processing such as ray-tracing and shading. Therefore, we propose the general-purpose execution unit with a dual-structure shown in Fig.3(c). In this structure, the execution units form three data-paths to execute, in parallel, three operations selected from a group of two additions and two multiplications. By connecting FMULs with FAUs and supplying each of two input data for the data-path A to each of two FAUs as accumulated data, two multiply-accumulates can be executed repeatedly in parallel.

Table 1 shows estimates of hardware size for each data-path structure. Estimates were obtained by result of logic synthesis and some consideration. The conventional structure in Table 1 refers to the structure that has both the multiply-accumulate execution unit and the general-purpose execution unit shown in Fig.2. The structures (a), (b), and (c) in Table 1 correspond to the structures (a), (b), and (c) in Fig.3, respectively, and are general-purpose execution units for both multiply-accumulate operations and irregular operations.

Our proposed dual-structured general-purpose execution unit (c) can handle both repetition of two multiply-accumulates and three flexible parallel operations with hardware increase of 7%, compared with the structure (a), by adding ports to the register file for one data-path and four selectors and eliminating the ACC. Moreover, the structure (c) is 9% larger than the SIMD-type execution unit, but reduces the hardware size of the conventional structure with both units from 48.1 K to 25.2 K gates, eliminating the SIMD-type execution unit.

Table 1: Estimation of the hardware size for each data-path structure including the register file. One gate is equivalent to one 2-input NAND.

Structure	Num. of gates
Conventional (Fig.2)	48.1 K
2-way SIMD (2-MAC)	23.1 K
General-purpose (3-datapath)	25.0 K
For general-purpose and MAC (Fig.3)	
(a) 2-datapath / 2-MAC	23.5 K
(b) 4-datapath / 2-MAC	26.7 K
(c) 3-datapath / 2-MAC	25.2 K

Furthermore, divider and square-root execution units, which are often used in graphics processing, are added to the structure above in G-FPU. Fig.4 shows a block diagram of G-FPU, which works as a coprocessor of the integer unit (IU). Floating-point instructions read out from the instruction memory are sent to G-FPU and executed. IU executes integer operations, load/store, and branch instructions.

G-FPU consists of two FAUs, two FMULs, a divider (FDIV), a square-root execution unit (FSRT), a floating-point register file (FREG), and an instruction decoder (DEC). For fast processing, FMUL, FDIV, and FSRT use a redundant binary expression in the intermediate execution of a mantissa part. Since there is no propagation of a carry in the addition of the redundant binary, the addition can complete within a constant time regardless of data length [8][9][10][11]. FAU1 is assigned to data-path A, FAU2 and FMUL1 to data-path B, and FMUL2, FDIV, and FSRT to data-path C. FREG consists of 32 32-bit registers (FR0-FR31) and has two read ports and one write port for each of three data-paths. FR0 is a special register from which zero is always read out. Furthermore, FREG has one 64-bit write port for data load from memory.

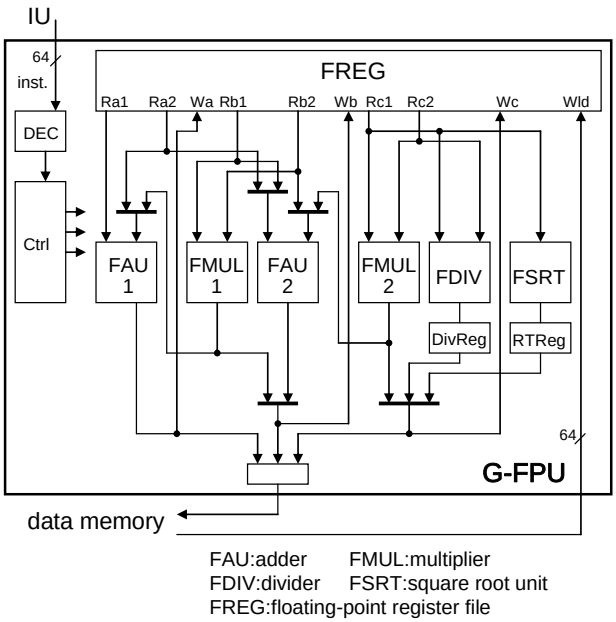


Fig. 4 Block diagram of G-FPU.

3. Data transfer mechanism

A maximum of four operations for the repetition of multiply-accumulate or a maximum of three for other operations can be executed in parallel. However, insufficient capability to supply data for operations and to transfer results could bring a large degradation of performance. To obtain a sufficient data transfer capability, there is a scheme to have a buffer memory and to transfer data needed for the next processing from a main memory to the buffer memory in parallel with data processing. For geometry transformation, sixteen words of matrix coefficients, four words of input coordinates, and four words of output coordinates are needed. Thus, a total of 24 words are needed and two banks of 24-word memory are required that consist of two read ports and one write port to transfer data during data processing. The buffer memory would be approximately 7.9 K gates, about 32% of structure (c) in Table 1.

For hardware reduction, FREG is used instead of buffer memory to supply data from the data memory to G-FPU. Since the capacity of FREG is small, a fine control for data transfer is necessary according to the flow of processing. Usually load/store instructions are used to transfer data between the register file and memory, but taking this approach could reduce performance because of increase of instructions. Therefore, we propose the multi-word load mechanism and the selective result store mechanism to achieve a high performance of data transfer. These new mechanisms make it possible to control data transfer finely with instructions and to reduce performance degradation caused by an increase of instructions.

3.1 Multi-word transfer mechanism

The multi-word transfer mechanism transfers multiple words from the memory to FREG in parallel with executions of floating-point operations by G-FPU. Fig.5 shows the mechanism that performs load and store for FREG through IU.

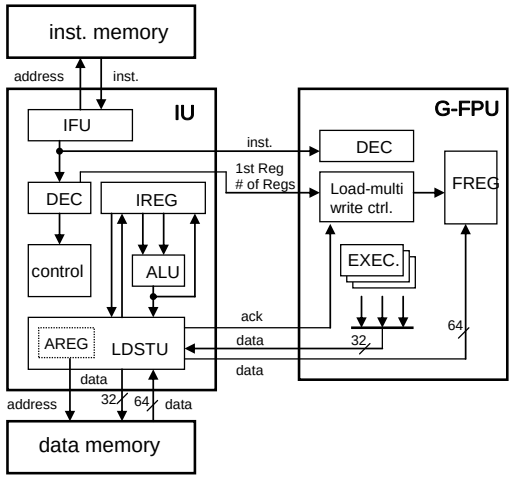


Fig. 5 Data transfer mechanism.

Data memory for load/store is accessed by the load/store unit (LDSTU) in IU. A new load-multi instruction (FLDM) is added for the multi-word load mechanism to transfer plural words.

The FLDM instruction specifies the IU register that stores the start address for data load, the number of words to be loaded, and the G-FPU register to which the first word is stored. According to this information, LDSTU reads the specified number of words from the data memory consecutively and supplies them to G-FPU. Then, G-FPU writes them to FREG sequentially, incrementing the register number specified by IU.

The read bus of the data memory is 64-bit wide, and two words are written to the two registers of FREG at one access.

The execution of floating-point operation instructions (FPU instructions) after FLDM can begin without waiting for the completion of the data load by FLDM. However, if an FPU instruction needs data which have not been loaded yet, the instruction must wait for the data to be loaded. Fig.6 shows the mechanism to detect the register dependency at the multi-word load. When an FLDM instruction is executed, FREG write-reservation bits that correspond to the registers to be loaded are set to "1". When data is loaded into the register, the corresponding write-reservation bit is cleared. If the FPU instruction after

FLDM uses the register with the corresponding write-reservation bit set to "1", then the register dependency is detected and the pipeline operation is locked.

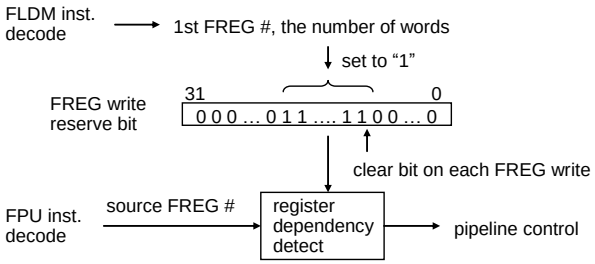


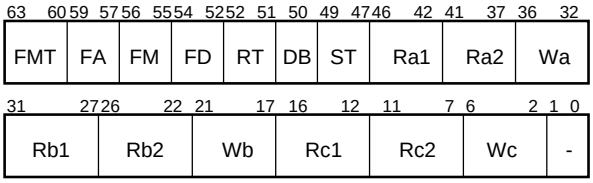
Fig. 6 Detection of Register Dependency on Multi-word Load

3.2 Selective result store mechanism

The selective result store mechanism stores the execution result to the memory at the same time that it is stored to FREG. By using this mechanism, a maximum of two data-paths out of three can be specified, and results of the specified data-paths are sent to LDSTU and stored to memory. The memory address of write is the value written in the address register by IU beforehand and is incremented after each memory write. This mechanism eliminates both extra store instructions needed to store data and a read port of FREG that is equivalent to about 0.8K gates.

4. Instruction format

Fig.7 shows the instruction format of G-FPU.
FMT is the field used to specify the type of instruction. The value "0xf" of FMT means that it is an FPU instruction that is 64-bit in length. Values other than "0xf" mean that an instruction with a 32-bit length is executed in the IU. FA, FM, FD, and RT fields are control fields for addition, multiplication, division, and square-root operation, respectively. They specify the presence of the operation and the register port to be used. The ports A, B, and C correspond to the data-paths A, B, and C respectively. For division and square-root operations, there is an execution start command and a result transfer command.
A function to double the result of addition is added to FAU1. DB field controls the double operation.
If FA specifies two additions for multiply-accumulate and FM specifies two multiplications, then each of the two adders adds the data of register port A and the result obtained by the multiplier at the previous cycle, and the two multipliers operate in parallel with the adders. Thus, two multiply-accumulate operations are executed.



FMT(inst. format) : 1111 = FPU inst.
FA(FAU ctrl.) : 100=fadd, 101=fadd + fadd, 010=fsub, 011=fsub + fsub, 110=fadd + fsub, 111=fadd(MAC) + fadd(MAC)
FM(FMUL ctrl.) : 1x=fmul(port B), x1=fmul(port C)
FD(FDIV ctrl.) :
010=fdiv start(port B), 011=fdiv start(port C), 100=fdiv result(port B), 101=fdiv result(port C)
RT(FSRT ctrl.) :
01=square root start, 10=square root result
DB(double ctrl.) : 1=double
ST(memory store) : result store port
xx1=port A, x1x=port B, 1xx=port C
Ra1, Ra2 : port A read register number
Wa : port A write register number
Rb1, Rb2 : port B read register number
Wb : port B write register number
Rc1, Rc2 : port C read register number
Wc : port C write register number

Fig. 7 Instruction format.

Field ST specifies whether an execution result is stored to the memory and to FREG simultaneously. This field can specify a maximum of two data-paths out of data-path A, B, and C.
Fields Ra, Ra2, and Wa specify two read and one write register numbers, respectively, for data-path A. Likewise, fields Rb1, Rb2, and Wb are for data-path B, and fields Rc1, Rc2, and Wc are for data-path C.

5. Pipeline operation

IU performs a pipeline operation consisting of five stages - F, D, E, M, and W. An instruction is fetched at stage F, decoded at D, and executed at E, and the result is stored to IREG at stage W. For load/store instructions, the memory is accessed at stage M, and data is written to IREG or the memory at W.
G-FPU performs a four-stage (F, D, X1, X2) pipeline operation for division and square-root instruction and a four-stage (F, D, X1, W) operation for other FPU instructions. Stages F and D are the same as those for IU. At stages X1 and X2, floating-point operation is executed. The cycle time of each of X1 and X2 is double the cycle time of IU. The execution result is stored to FREG at stage W. For a division and square-root operation, the result is stored to its own register and then transferred to FREG by another instruction.

Fig.8 shows the pipeline processing of a repetition of two multiply-accumulate operations in parallel with data transfer by FLDM instruction.

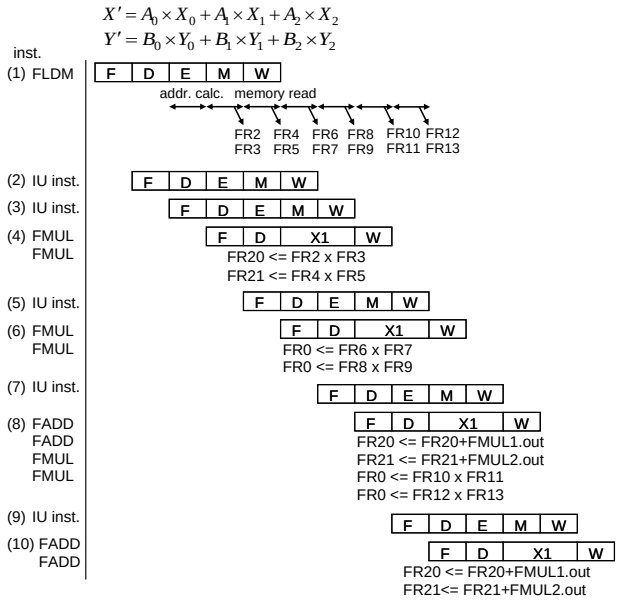


Fig. 8 Pipeline operation.

Instruction (1) is FLDM and transfers 12 word data to FREG that are necessary for multiply-accumulate operations. The data transfer is performed in parallel with execution of successive instructions. Two multiplications of instruction (4), $A_0 \times X_0$ and $B_0 \times Y_0$, need four word data and wait until they are loaded. Meanwhile, IU instructions (2) and (3) are executed.

Two multiplications of instruction (6), $A_1 \times X_1$ and $B_1 \times Y_1$, are executed after completion of stage X1 of instruction (4). The destination register, FR0, is a zero register to which the result is not written.

Instruction (8) executes two additions and two multiplications. Each of two additions adds the data from FREG, that is, the multiplication result of instruction (4) and the multiplication result of the previous cycle. Simultaneously, two multiplications, $A_2 \times X_2$ and $B_2 \times Y_2$, are executed, meaning that four execution units work in parallel and execute two multiply-accumulate operations. Thus, two multiply-accumulate operations are executed at every X1 stage without the ACC.

Data-path A is not used in instructions (4) and (6), and therefore, the adder can be used for another operation.

Besides, as the cycle time of G-FPU execution is double the cycle time of IU, an IU instruction can be inserted and executed between FPU instructions.

6. Application of graphics processing to G-FPU

Examples are shown that calculation of intersection in ray-tracing algorithm as irregular processing and geometry transformation as regular processing are applied to G-FPU.

6.1 Intersection calculation

A plane with a normal vector (a,b,c) is expressed in the following equation.

$$ax + by + cz + d = 1$$

In this equation, $(x,y,z) = Pray = t \times Vray + Peye$ is given and t is calculated as follows,

$$t = -\frac{aPeye.x + bPeye.y + cPeye.z + d}{aVray.x + bVray.y + cVray.z}$$

and the intersection (x,y,z) is obtained from t [12].

Fig.9(a) shows an example where the algorithm above is translated to the G-FPU program. FRn (N = 0 to 31) is the register of FREG. FMUL, FADD, FSUB, and FDIV are multiplication, addition, subtraction, and division, respectively. A ".st" added to the operation name means that the selective result store is performed at the operation, that the result is sent to LDSTU of IU, and that the result is stored to the data memory at the same time as it is stored to FREG.

Fig.9(b) shows processing cycles corresponding to the three structures in Fig.3. The CYC column shows the number of execution cycles and the right-hand columns indicate FPU operations which are to be executed at a cycle. Although some IU instructions can be executed between FPU instructions, they are not shown in Fig.9.

With the proposed structure (c), which flexibly assigns the execution units to the three data-paths, nine FPU instructions execute 19 operations, that is, two operations are, on average, executed efficiently by a single FPU instruction. This processing cycle is equivalent to the 4-datapath structure (b) and is one FPU clock cycle less than the 2-datapath and 2-MAC structure (a).

By using FLDM, one instruction loads ten words from the memory to FREG in parallel with the execution of the following instructions.

Furthermore, by using the selective result store mechanism, three stores (x,y,z) to the memory need no store instructions. Thus, seven instructions of loads and stores are eliminated and the processing cycles are reduced by 26% from 27 to 20 cycles in the IU clock.

(a) List of G-FPU program for the intersection calculation

(1) FLDM FR2, (IR2), 10
(10 words are loaded into FR2 to FR11 from the address specified by IR2 in parallel with the following instructions)
(2) FMUL FR12, FR2, FR3 $a \times Peye.x$
(3) FMUL FR15, FR2, FR4 $a \times Vray.x$
(4) FADD FR12, FR5, FR12 *FR5= d*
(5) FMUL FR13, FR6, FR7 $b \times Peye.y$
(6) FMUL FR16, FR6, FR8 $b \times Vray.y$
(7) FADD FR12, FR12, fmul1.out
(8) FADD FR15, FR15, fmul2.out
(9) FMUL FR0, FR9, FR10 $c \times Peye.z$
(10) FMUL FR0, FR9, FR11 $c \times Vray.z$
(11) FADD FR12, FR12, fmul1.out
(12) FADD FR15, FR15, fmul2.out
(13) FDIV.start FR12, FR15 *start FDIV*
(14) FDIV.get FR20 $FR20 = (-t)$
(15) FMUL FR21, FR20, FR4 $(-t) \times Vray.x$
(16) FMUL FR22, FR20, FR8 $(-t) \times Vray.y$
(17) FMUL FR23, FR20, FR11 $(-t) \times Vray.z$
(18) FSUB.st FR21, FR3, FR21 *(FR21=x, Store x to memory)*
(19) FSUB.st FR22, FR7, FR22 *(FR22=y, Store y to memory)*
(20) FSUB.st FR23, FR10, FR23 *(FR23=z, Store z to memory)*

(b) Execution cycles of the above program for each datapath structure

CYC	(a) 2-datapath/ 2-MAC	(b) 4-datapath/ 2-MAC	(c) 3-datapath/ 2-MAC(proposed)
1	(1)	(1)	(1)
4	(2), (3)	(2), (3)	(2), (3)
6	MAC(5,7), MAC(6,8)	(4), (5), (6)	(4), (5), (6)
8	MAC(9,11), MAC(10,12)	(7), (8), (9), (10)	(7), (8), (9), (10)
10		(11), (12)	(11), (12)
12	(4)	(13)	(13)
14	(13)	(14)	(14)
16	(14)	(15), (16)	(15), (16)
18	(15), (16)	(17), (18), (19)	(17), (18), (19)
20	(17), (18)	(20)	(20)
22	(19), (20)		

Fig.9 Processing of Intersection Calculation

6.2 Calculation of geometry transformation

We show the case that the matrix operation of 4×4 as the geometry transformation is applied to G-FPU.

$$\begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix} = \begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

The list in Fig.10 is an example where the above operation is translated to the G-FPU program.
In this list, the number of execution cycles is given at the left end of the line. In this example, the numbers of processing cycles are the same for the three structures in Fig.3. Two multiply-accumulates are executed at one cycle by operating four execution units in parallel, and 30 operations are executed by 9 FPU instructions. Furthermore, 20 words of data are loaded from the memory to FREG in parallel with executions, and four results are transferred to the memory without any store instructions by using the selective result store mechanism. Thus, by removing 13 instructions for 64-bit loads and 32-bit stores, the processing cycles are reduced by 39% from 33 to 20 cycles in the IU clock.

CYC
1: FLDM FR2, (IR2), 20;
(20 words are loaded into FR2 to FR21 from the address specified by IR2 in parallel with the following instructions)
4: FMUL FR22, FR6, FR2; $a_{00} \times x$
FMUL FR23, FR7, FR2; $a_{10} \times x$
6: FMUL FR0, FR8, FR3; $a_{01} \times y$
FMUL FR0, FR9, FR3; $a_{11} \times y$
8: FADD FR22, FR22, FMUL1.out;
FADD FR23, FR23, FMUL2.out;
FMUL FR0, FR10, FR4; $a_{02} \times z$
FMUL FR0, FR11, FR4; $a_{12} \times z$
10: FADD FR22, FR22, FMUL1.out
FADD FR23, FR23, FMUL2.out;
FMUL FR0, FR12, FR5; $a_{03} \times w$
FMUL FR0, FR13, FR5; $a_{13} \times w$
12: FADD.st FR22, FR22, FMUL1.out; *(Store x' to memory)*
FADD.st FR23, FR23, FMUL2.out; *(Store y' to memory)*
FMUL FR0, FR14, FR2; $a_{20} \times x$
FMUL FR0, FR15, FR2; $a_{30} \times x$
14: FADD FR24, FR0, FMUL1.out;
FADD FR25, FR0, FMUL2.out;
FMUL FR0, FR16, FR3; $a_{21} \times y$
FMUL FR0, FR17, FR3; $a_{31} \times y$
16: FADD FR24, FR24, FMUL1.out;
FADD FR25, FR25, FMUL2.out;
FMUL FR0, FR18, FR4; $a_{22} \times z$
FMUL FR0, FR19, FR4; $a_{32} \times z$
18: FADD FR24, FR24, FMUL1.out;
FADD FR25, FR25, FMUL2.out;
FMUL FR0, FR20, FR5; $a_{23} \times w$
FMUL FR0, FR21, FR5; $a_{33} \times w$
20: FADD.st FR24, FR24, FMUL1.out; *(Store z' to memory)*
FADD.st FR25, FR25, FMUL2.out; *(Store w' to memory)*

Fig.10 Processing of geometry transformation

As shown in the examples above, the proposed dual-structured general-purpose processing unit can perform both repeated multiply-accumulate operations and irregular operations efficiently. Furthermore, with the multi-word load and selective result store mechanisms, instructions for data transfer are eliminated and data needed for parallel executions are supplied without delay.

7. Conclusion

We have proposed a graphics floating-point processing unit (G-FPU) with 48% reduction of hardware for a conventional processing unit that has both functions of a SIMD-type execution unit dedicated for multiply-accumulate operations and a general-purpose execution unit. The hardware reduction is obtained by realizing a dual-structured execution unit that can handle

both repeated operations of multiply-accumulate for geometry transformations and irregular operations such as ray-tracing in graphics with 9% of hardware increase for a SIMD-type execution unit.
For irregular processing, three execution units of three data-paths are operated in parallel, and for repetition of multiply-accumulate operations, two multiply-accumulates are executed by connecting two adders and two multipliers. Compared with a general-purpose execution unit with a 4-datapath structure, this dual-structure scheme reduces hardware by 6% and achieves an equivalent performance for irregular processing such as ray-tracing.
The proposed mechanisms of the multi-word load and the selective result store eliminate a buffer memory of 7.9 K gates for temporary data store and a read port of register file for store equivalent to 0.8 K gates and remove instructions to transfer data between the register file and the memory. Processing cycles are reduced by 26% for the intersection calculation by eliminating 7 load/store instructions and 39% for the geometry transformation by eliminating 13 load/store instructions.

References

[1] M. Suzuoki et al., ``A Microprocessor with a 128-Bit CPU, Ten Floationg-Point MAC's, Four Floating-Point Dviders, and an MPEG-2 Decoder", IEEE J. Solid-State Circuits, Vol. 34, No.11, pp.1608-1618, 1999.
[2] N. Ide et al., ``2.44-GFLOPS 300-MHz Floating-Point Vector-Processing Unit for High-Performance 3-D Graphices Computing", IEEE J. Solid-State Circuits, Vol. 35, No.7, pp.1025-1033, 2000.
[3] E. Lindholm et al., ``NVIDIA Tesla: A Unified Graphics and Computing Architecture", IEEE Micro, Vol. 28, No.2, pp.39-55, 2008.
[4] L. Seiler et al., ``Larrabee: A Many-Core x86 Architecture for Visual Computing", ACM T. Graphics, Vol. 27, No.3, pp. 18:1-18:15, 2008.
[5] K. Maeda et al., `` Floating Point Accelerator Design Using Verilog HDL", 2004 Shikoku-section Joint Convention of the Institutes of Electrical and related Engineers, p.119, 2004. (in Japanese)
[6] M. Yokoyama et al., ``Design of Floating Point Unit For Graphic Processing". 2006 Shikoku-section Joint Convention of the Institutes of Electrical and related Engineers, p.84, 2006. (in Japanese)
[7] H. Kubosawa et al., ``A 2.5-GFLOPS, 6.5 Million Polygons per Second, Four-Way VLIW Geometry Processor with SIMD Instructions and a Software Bypass Mechanism", IEEE J. Solid-State Circuits, Vol. 34, No.11, pp.1619-1626, 1999.
[8] S. Kuninobu et al., ``Design of High Speed MOS Multiplier and Divider using Redundant Binary Representation", ISCA 87, pp.80-86, 1987.
[9] H. Edamatsu et al., ``A 33 MFLOPS Floating-Point Processor using Redundant Binary Representation", ISSCC 88, pp.152-153, 1988.
[10] T. Taniguchi et al., ``A high-speed floating-point processor using redundant binary representation", IEICE technical report. Computer systems, CPSY87-47, pp.43-48, 1988 (in Japanese)
[11] T. Taniguchi et al., ``High-speed multiplier and divider using redundant binary representation", IEICE technical report. Electron devices, ED88-48, pp.1-6, 1988 (in Japanese)
[12] S. Oishi and M. Makino, ``Information mathematics seminar Graphics", ISBN4-535-60816-4 Nippon-Hyoron-Sha Co.,Ltd. (in Japanese)

Jiro Miyake received the B.S. degree in electronic engineering from Kyushu University, Fukuoka, Japan, in 1983. He joined Matsushita Electric Industrial Co., Ltd., in 1983 where he has been working on the design of VLSI and microprocessors. Since 2008 he is a doctor student in Graduate School of Information, Production and System of Waseda University. His current interests include microprocessor architecture and digital signal processing. He is a member of the IEEE Computer Society.

Shigeo Kuninobu received the B.S. and M.S. degrees in Electrical Engineering and Ph.D. degree in Information Science from Kyoto University in 1968, 1970 and 1993, respectively. He joined the Central Research Laboratories of Matsushita Electric. in 1970. He was a visiting researcher at the University of California, Berkeley, from 1982 to 1984. His interests cover the development of the area of advanced VLSI Microsystems. He developed a computational algorithm for LSI, microprocessors, and floating-point processors. He was a professor at Information Science Division, Kochi University from 2001 to 2007. He is now a visiting professor at Graduate School of Information Science, Waseda University and an honorary professor at Kochi University.

Takaaki Baba was born in Aichi Japan on Jan. 1949. He received Ms. Degree and Dr. of Engineering from Nagoya University in 1973 and 1979, respectively. He joined Matsushita Electric Industrial Co., Ltd. in 1973. From 1983 to 2002 he worked for Matsushita Electric Co., Ltd of America, involving and conducting several strategic projects such as System LSI and ASIC application, wireless communication system and electronic devices. From 1980 to 1982, he was a research fellow at UC Berkeley. From 2002 to 2003, he was a visiting scholar at Stanford University. Since 2003 he is a professor in the system LSI field at Graduate School of Information, Production and System of Waseda University. He is a member of IEEE and served as an Executive Committee member of IEEE-ISSCC from 1995 to 2003.