Evaluation of Matrix FMA Operation for Array Processor Using Free Cell Library

Okuyama Yuichi and Nakamura Mitsuhiro
Graduate school of the University of Aizu, Japan
okuyama@u-aizu.ac.jp

Abstract

We implement new arithmetic unit for matrix operation named matrix FMA. This unit read three matrices at one cycle and output the result of $AB + Cf$ in every cycle. We employ this unit to array processor for matrix multiplication and evaluate this unit in 45nm NanGate process. The result shows throughput become double though ×1.3 area penalty in the case of 4×4 matrix operation.

1. Introduction

Matrix operations have a key role in large-scale scientific computations such as fluid dynamics, and structural analysis [1, 2]. These applications require a tremendous amount of matrix operations. Since early times, a lot of techniques for matrix operations have been proposed [3]. Many modern processors have a fused multiply-add (FMA) instruction as a solution, such as Intel Itanium2 [4], and Sony/Toshiba/IBM Cell B.E. [5]. An FMA instruction calculates $a \times b + c$ within a single latency of processor cycle. Though FMA helps fast matrix processing, the instruction must be executed at least n^3 times for $n \times n$ matrix multiplication. This means the clock frequency dominates the speed of matrix processing in a single processor core. On the other hand, some architectures for matrix multiplication have been proposed to overcome this problem. These architectures have multiple processing elements (PEs) connected by linear, ring, mesh, torus, and hypercube topologies. RapidMatriX [6] is an architecture organized by a two-dimensional PE array with the torus topology. This architecture achieves fast matrix operations executing the FMA in parallel. Throughput of the multiplication is scalable, which means we can get better performance by increasing PEs. We considered to change the grain of PEs [7] to use DSPs in FPGA efficiently. Hereby, we aimed to accelerate matrix operations. In this paper, we propose to merge scalar FMAs into an integrated matrix FMA of 2×2 and 4×4 in ALU of the PE in order to use the DSP blocks of the FPGA efficiently and achieve fast matrix operations.

From the result of [7], the performance of matrix multiplication and usage of the logic element is increased by the size of matrix FMA operations in proportion with little overhead. We expect we can use this technology to optimize matrix processing using usual semiconductor devices using NanGate FreePDK45 Open Cell Library [8].

In this paper, we introduce array processor for matrix multiplication in section 2. Then matrix multiplication using course grained PEs are explained in section 3. In section 4, we evaluate implemented processors using Open Cell Library. Finally, we show the summary of this paper in section 5.

2. Array processor for matrix multiplication

We introduce an array processor described in [6], which is a special processor for matrix operations. This processor consists of SIMD PE array. Figure 1 shows a schematic diagram of the array processor.

PE array employs the structure of the systolic array. This structure has the following characteristics.

1) Structure organized by simple cells.
2) Control flow for simple regular data and control.
3) High-throughput calculation by parallel and pipeline.
4) Convolution of input, output and processing.

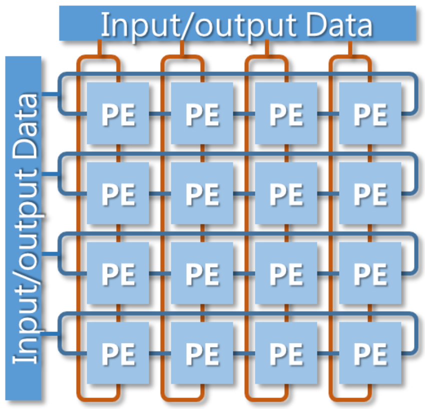

Figure 1. Array processor for matrix multiplication

2.1 Two-dimensional torus

The PE array is organized by a two-dimensional torus. All PEs connect with adjacent PEs in the two-dimensional torus. Each PE executes FMA operation. Calculation of matrix multiplication is achieved by executing FMA operations as shifting a_{ij} and b_{ij} of matrix elements in a horizontal and vertical direction respectively. This calculation steps is repeated n times for n×n matrix multiplication. Finally, the results of the matrix multiplication is kept in PEs.

$$FMA(a, b, c): d = a \times b \pm c \qquad (1)$$

An I/O port on the left and top edge can input data from outside by two-dimensional torus topology. In input data operation, this processor inputs data from the left edge and the top edge of torus topology shifting data in all PEs. It means this array processor can input to PE array of n × n from outside in n steps.

2.2 Structure of PE

As shown in Figure 2, a PE contains register file, instruction decoder and Fused Operation Unit (FOU). A PE has input ports for data from the south and east cells and instructions. A PE has three output ports for data to west and north cells. The register file consists of four register groups. Three of them store data from the input, and the last one stores result in data from the calculation in FOU. The instruction decoder determines the behavior of circuits by issuing control wires to register file, FOU, and several selector, after instruction decode.

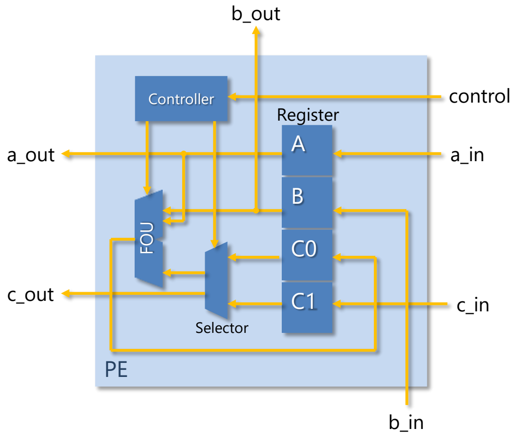

Figure 2. Array processor for matrix multiplication

2.3 Instruction Set

PE is controlled by SIMD operations. Control Unit issues same instructions to all PEs and PEs execute same processes. The array processor supports four instruction types as ***LOAD***, ***MOVE***, ***FOU*** and ***NOP***.

- ***LOAD*** : Input data to the PE
- ***MOVE*** : Data transfer between adjacent PEs
- ***FOU*** : Calculation and data transfer between adjacent PEs
- ***NOP*** : No operation

In both processes, the array processor can store an immediate data to an intended PE by selecting register type, register number, and immediate data in the instruction field. Also, (b) selects PE for writing data by two parameters. To transfer data to adjacent PEs, MOVE instruction selects a source and a forwarding address of the register. This instruction does not select a forwarding address of the PE because data is only transferred to adjacent PEs. FOU instruction selects a register of input and output, to execute calculation and the process of MOVE instruction simultaneously.

2.4 Procedure of matrix multiplication

This section shows the calculation procedure of the array processor. The calculation procedure of matrix operations is as follows.

1) Execute skew operation to matrix multiplicand and matrix multiplier
2) Input matrix multiplicand and matrix multiplier to each PE by MOVE instructions
3) Execute FOU instructions

First, the matrix is transformed by a skew operation to execute matrix operation on torus topology. Skew operation changes general matrix form into matrix form for matrix operations utilizing torus. In skew operation, the row (i) of an A of matrix multiplicand shifts in $(i-1)$ left direction while the line (j) of a B matrix multiplier shifts in $(j-1)$ under direction. Therefore, after skew operation, if A and B are $n \times n$ matrices, these matrix elements are represented as following.

$$\begin{aligned} a'_{i,j} &= a_{i,j'(i,j)} \\ b'_{i,j} &= b_{i'(i,j),j} \end{aligned} \qquad (2)$$

where

$$\begin{aligned} j'(i,j) &= \begin{cases} j-i, & j-i \geq 1 \\ j-i+n, & \text{otherwize} \end{cases} \\ i'(i,j) &= \begin{cases} i-j, & i-j \geq 1 \\ i-j+n, & \text{otherwize} \end{cases} \end{aligned} \qquad (3)$$

On $n \times n$ PE array, the processor can calculate matrix multiplication by n times executing FOU operations, after transfer two matrices data that are performed by MOVE operations.

3.1 Block Matrix Multiplication

This section shows notations for block matrix multiplication. Matrix multiplication is show as

$$D = A \cdot B \qquad (4)$$

were A, B,and D are $n \times n$ matrices. We consider the block matrix notation for $n/2 \times n/2$ matrix as follows

$$\begin{aligned} D &= \begin{pmatrix} D_{11} & D_{12} \\ D_{21} & D_{22} \end{pmatrix} \\ A &= \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \end{aligned} \qquad (5)$$

$$B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$$

where A_{ij}, B_{ij}, D_{ij} are $(n/2 \times n/2)$ matrices. Block matrix multiplication is described as

$$\begin{pmatrix} D_{11} & D_{12} \\ D_{21} & D_{22} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix} = \qquad (6)$$

$$\begin{pmatrix} A_{11}B_{11} + A_{12}B_{21} & A_{11}B_{12} + A_{12}B_{22} \\ A_{21}B_{11} + A_{22}B_{21} & A_{21}B_{12} + A_{22}B_{22} \end{pmatrix}$$

that are compatible with equation 4.

3.2 Matrix FMA

Data transfer and FMA operation of each PEs are performed as scalar data. In this section, we propose a scheme to combine multiple PEs. We define A, B, C and D as the $n \times n$ matrix data, and matrix elements as a_{ij}, b_{ij}, c_{ij}, and d_{ij}, in addition to multiply-accumulation of matrices $D = AB + C$ is defined as a matrix FMA. Matrix FMA is shown in following equation

$$FMA(A, B, C)\colon\ d_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj} \pm c_{ij} \qquad (7)$$

where $1 \le i, j \le n$.

If n is divided by m, eq(x) is rewritten in block matrix notation as

$$FMA(A, B, C)\colon D = AB + C$$
$$: D_{ij} = \sum_{k=1}^{n/m} A_{ik} B_{kj} \pm C_{ij} \qquad (8)$$

where $1 \le i, j \le n/m$.

The calculation procedure of $D = AB + C$ can be described same form as calculation procedure for scalar matrix operations. Altogether, FMA operation is shown in the following equation.

By using equations 1, 7 and 8, we can replace scalar FMA to matrix FMA with same array control. Moreover, we can change the grain size of PE array by replacing scalar data with matrix data.

Figure 3 shows the structure of PE array for 4×4 matrix operation using 2×2 PEs. In this array, transferred scalar data become 2×2 matrix data. The topology and a structure of control data are the same as the current PE array for 2×2 matrix operation. In the case of 4×4 FMA, the processor will have a single PE to manipulate 4×4 matrices.

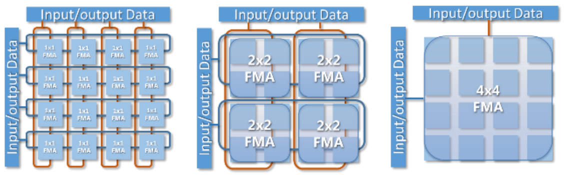

Figure 3. Array processor using matrix FMA for 4x4 matrix multiplication. 4x4 scalar FMA, 2x2-2x2 matrix FMA, and 1x1-4x4 matrix FMA.

Calculation Procedure with matrix FMA We must change the calculation procedure to execute the calculation using matrix FMA. First, we must change skew operation by replacing scalar data between each PE. We simply apply block matrix defined in equations 2 and 3. Next, we change scalar data of FOU instruction to matrix data in transferring data.

4. Implementation and evaluation

In this research, we evaluated three networks that are PE array for 4x4 matrix multiplication with 1x1, 2x2 and 4x4 matrix FMA on ASIC. In this research, to evaluate the only performance of networks, networks don't include data I/O. Also, each PEs is macro. We collocated PEs manually based on figure 3. Mainly, we modify following three parts of each PE.

1) **Data bus**: Matrix elements are represented in 16-bit integer in this implementation. PEs contain 2×2 matrix, and 4 × 4 matrix FMA, the data buses contain 64 bits and 256 bits of data to connect each PE.
2) **Register File**: Register file described in Section 2 contains 64 bit and 256bit registers for 4 and 16 elements of 16-bit data. Register file sends 192bit and 768-bit data as three matrices to calculate D = AB + C in FOU at one cycle of latency.
3) **ALU**: We implement three ALUs (FOUs); scalar FMA, 2×2 matrix FMA, and 4×4 matrix FMA. The scalar FMA applies the FMA operation in 16 bits. On the other hand, 2×2 and 4×4 matrix FMA divide input data between 16 bits. Here, these data are matrix elements. Finding the results of calculation of each matrix element, ALU outputs matrix data of bit length same as input data.

We used following tools for logic synthesis, wiring and macro.

- NSL CORE version 20120210
  Language translator from new synthesis language (NSL) to Verilog HDL
- Design Compiler version B-2008.09
  Logic synthesizer to get a netlist from circuit description from Verilog HDL
- Encounter RTL-to-GDSII version 10.10
- Virtuoso Abstract Generator version IC6.1.5.72
  Tools for layout including place and route for semiconductor devices

We used following cell library for this evaluation.

- Nangate45nmOpenCellLibrary
  support FreePDK v1.3 version v2010_12

Table1 is the result of the synthesis of a single macro PE with 1×1, 2×2 and 4×4 matrix FMA. The chip area and frequency increases in proportion to the grain size of matrix FMA.

Table 1. Result of synthesis of a single PE

FMA	**Chip area (μm^2)**	**Chip area gain**	**Frequency (MHz)**	**Frequency gain**
1 × 1	6524	1	186	1
2 × 2	29106	4.46	165	0.89
4 × 4	142007	12.7	94	0.50

Table 2 is the result of the synthesis of PE array for 4x4 matrix multiplication with 1×1, 2×2 and 4×4 matrix FMA. The unit is the throughput of 4×4-16bit integer matrix multiplication and add per second (described 4×4 matrix/sec). To estimate results of throughput, we use following equation.

$$T_h = \frac{F}{N_{step}} \qquad (7)$$

Here, *Th* is Throughput, *F* is clock frequency, and *Nstep* is the number of the necessary clock for 4x4 matrix multiplication. PE array with m × m matrix FMA become1/m times in comparison with PE array with 1x1 matrix FMA.

The chip area doesn't widely change. Also, the amount of throughput gain doesn't increase in proportion to the gain size of matrix FMA

Table 2. Result of synthesis of PE array for 4x4 matrix multiplication

FMA	**Chip area (μm^2)**	**# of PE**	**# of steps**	**Frequency (MHz)**	**Throughput (4x4 matrix/ sec)**	**Throughput gain**
1 × 1	107484	**16**	**4**	**216**	**54.0**	**1**
2 × 2	120149	**4**	**2**	**184**	**91.9**	**1.7**
4 × 4	143784	**1**	**1**	**115**	**115**	**2**

From this result, 4×4 FMA consumes 1.33 times of area than 4×4 scalar FMA. However, we can reduce clock frequency almost half than scalar FMA. Also, throughput is increased to double. We can say matrix FMA can save power consumption and increase performance with 30% of the additional area in a chip.

5. Summary

In this paper, we discussed and evaluated matrix FMA for array processors. PEs of the array processor execute scalar multiply and accumulation. This array processor accelerates matrix operation PEs connected with torus topology. To implement functional units for matrix operation to array structure, we proposed combine scalar FMAs to matrix FMA to change the grain size of PEs. We implement array processors using different grain of PEs using NanGate FreePDK45 Cell Library. Also, we evaluate the clock frequencies and the clock cycles of calculation. As results, the throughput of PE of 4×4 matrix FMA is two times, and area penalty is almost 30%.

Acknowledgement

This work is supported by VLSI Design and Education Center (VDEC), the University of Tokyo in collaboration with Synopsys, Inc. and Cadence Design Systems, Inc.

References

[1] T. kunihiko, "Proper Orthogonal Decomposition in Fluid Flow Analysis," Fundamental Technology
[2] Research Center, Honda R&D Co., Ltd., 14 January, 2011
[3] Chaojiang Fu, "Parallel Computing For Finite Element Structural Analysis On Workstation Cluster," International Conference on Computer Science and Software Engineering, 2008
[4] S. Y. Kung, "VLSI Array Processors," Prentice Hall, 1987
[5] "Intel Itanium Processor," Internet: http://www.intel.com/content/www/us/en/processors/itanium/itanium-processor-9000-sequence.html
[6] M.Gschwind, H.P.Hofstee, B.Flachs, M.Hopins, Y.Watanabe, and T.Yamazaki, "Synergistic Processing in Cell's Multicore Architecture," IEEE Micro, Vol. 26, Issue2, pp.10-24, March-April 2006
[7] Y. Sato, Y. Nomoto, T.Miyazaki, and S. G. Sedukhin, "2-D Array Processor Featuring Ternary Input Fused Operations," IPSJ SIG Tech. Reports, 2007-SLDM-131, pp.7-12
[8] Y. Okuyama, S. Takano, and T. Shirai, "Design of a Coarse-grained Processing Element for Matrix Multiplication on FPGA," IEEE Proceedings of the 8th International Symposium on Embedded Multicore/Many-core SoCs (MCSoC-14), pp.237-241 , Sept. 2014.
[9] "NanGate FreePDK45 Open Cell Library," http://www.nangate.com/?page_id=2325, NanGate Inc.