NAIST-IS-DD1261201

Doctoral Dissertation

Performance Evaluation of a 3D-Stencil Library for Distributed Memory Array Accelerators

Yoshikazu Inagaki

September 16, 2015

Department of Information Science
Graduate School of Information Science
Nara Institute of Science and Technology

A Doctoral Dissertation
submitted to Graduate School of Information Science,
Nara Institute of Science and Technology
in partial fulfillment of the requirements for the degree of
Doctor of ENGINEERING

Yoshikazu Inagaki

Thesis Committee:

Professor Yasuhiko Nakashima	(Supervisor)
Professor Michiko Inoue	(Co-supervisor)
Assistant Professor Shinya Takamaeda-Yamazaki	(Co-supervisor)
Assistant Professor Tran Thi Hong	(Co-supervisor)

Performance Evaluation of a 3D-Stencil Library for Distributed Memory Array Accelerators*

Yoshikazu Inagaki

Abstract

The Energy-aware Multi-mode Accelerator eXtension [24, 25] (EMAX) is equipped with distributed single-port local memories and ring-formed interconnections. The accelerator is designed to achieve extremely high throughput for scientific computations, big data, and image processing as well as low-power consumption. However, before mapping algorithms on the accelerator, application developers require sufficient knowledge of the hardware organization and specially designed instructions. They also need significant effort to tune the code for improving execution efficiency when no well-designed compiler or library is available. A similar problem exists in EMAX. To address this problem, we focus on library support for stencil (nearest-neighbor) computations that represent a class of algorithms commonly used in many partial differential equation (PDE) solvers. In this dissertation, we address the following topics: (1) system configuration, features, and mnemonics of EMAX; (2) instruction mapping techniques that reduce the amount of data to be read from the main memory; (3) performance evaluation of the library for PDE solvers. With the features of a library that can reuse the local data across the outer loop iterations and map many instructions by unrolling the outer loops, the amount of data to be read from the main memory is significantly reduced to a minimum of 1/7 compared with a hand-tuned code. In addition, the stencil library reduced the execution time 23% more than a general-purpose processor, and it was shown that EMAX and the 3D-Stencil Library have the superior performance compared with GPGPU.

* Doctoral Dissertation, Department of Information Science, Graduate School of Information Science, Nara Institute of Science and Technology, NAIST-IS-DD1261201, September 16, 2015.

Keywords:

CGRA, coarse grained reconfigurable architecture, accelerator, library, stencil, optimization

Contents

1. Introduction **1**
1.1 Performance enhancement of computations 1
1.2 Partial Differential Equations (PDE) 1
1.3 CGRA for Stencil Computations 2
1.4 3D-Stencil Library for EMAX 2
1.5 Organization 3

2. Related Work **4**
2.1 CGRA for Stencil Computations 4
2.2 Linear Array Pipelined Processor (LAPP) 6

3. Stencil Computation **10**
3.1 3D-Stencil Kernels 10

4. Overview of EMAX **11**
4.1 System Diagram of EMAX 11
4.2 Interface with Host Computer 11
4.3 EMAX mnemonics 11
4.4 Instruction Mapping 13

5. 3D-Stencil Library for EMAX **18**
5.1 User Interface of 3D-Stencil Library 18
5.2 Structure of 3D-Stencil Library 18
5.3 Basic Instruction Mapping 22
5.4 Packing of Instructions 23
5.5 Evaluation of Packing 23
5.6 Parallel Mapping in 3D-Stencil Library 24

6. Results and Analysis **30**
6.1 Simulation Model for Performance Measurement 30
6.2 Evaluation of 3D-Stencil Library 33
6.3 Comparison with general-purpose processors 35
6.4 Comparison with GPGPU 35

7. Conclusion **40**

Acknowledgements **41**

References **42**

Publications **48**

List of Figures

1 3D-Stencil Kernels 6
2 The structure of LAPP 9
3 EMAX configuration 12
4 EMAX mnemonics 14
5 Instruction shift by "dist" 15
6 Source code of unsharp() 16
7 EMAX mnemonics of unsharp() 17
8 Interface of stencil kernel 19
9 Source Code of Stencil Computing with EMAX mnemonics [jacobi] 19
10 Source Code of Stencil Computing with EMAX mnemonics [FD6] 20
11 Structure of 3D-Stencil Library 21
12 Divided 3D-Stencil space 22
13 Basic instruction mapping 26
14 Parallel mapping 27
15 Data transmission to LMM on stencil computing 28
16 Instruction mapping of stencil [$degree = 1$] 29
17 Execution sequence of EMAX 32
18 Execution time of 3D-Library [degree=1] 34
19 Execution time of 3D-Library [degree=3] 34
20 Comparison of execution between CPUs and EMAX 36
21 3D stencil space divided according to number of threads 37
22 Execution time of GPGPU [degree=1] 38
23 Execution time of GPGPU [degree=3] 38
24 Comparison of execution between GPGPU and EMAX 39

List of Tables

1 Simulation parameters 30
2 General-purpose processors for comparison 35
3 Specification of GPGPU 37

1. Introduction

1.1 Performance enhancement of computations

To speed up the scientific and technological computations required in such fields as image processing and 3-dimensional simulation, many studies and developments have been reported from the views of hardware and software. From the view of the former especially in the high performance computations (HPC) field, computer systems have employed high-performance processors equipped with SIMD units, such as SPARC64 with HPC-ACE [7], Intel with SSE/AVX extensions [3], and ARM with Neon extensions [4], or such general-purpose accelerators as GPGPU [6, 9] and Xeon Phi [8]. Recently, the continuous performance improvements by advancing both the on-die transistor density and the switching frequency have been facing an increasing challenge from the power constraints, which is also known as the utilization wall. Therefore, the computing industry has shifted from this exponential scaling in the clock frequency toward chip multiprocessors (CMPs) in order to better trade-off among performance, energy efficiency and reliability. Other than the traditional CMP architecture such as many-core general platforms [40], coprocessors including GPGPU and Xeon Phi have also gained their positions of importance in HPC field for their high peak GFlops density and state-of-art ability of managing thread level parallelism [41, 42, 43]. The single instruction multiple threads (SIMT) in GPGPU and SIMD targeted vectorization in Xeon Phi [44] have been popularly used to provide a vector-like processing to accelerate high performance computations. Other than the computation itself, the memory characteristics in HPC programs should also be well taken care of, as it usually defines the interleaved thread division boundaries, and the physical allocation of the divided data sets, especially for the memory bound applications. A special class of algorithms to access nearest neighbor data, which is known as stencil computation [39, 46, 45, 16], has been recently gaining more importance.

1.2 Partial Differential Equations (PDE)

In general, SPEC [1] and NPB [2] are widely used to represent the performance of computer systems. However, in the HPC field, the importance is growing

of measuring the performance obtained by several standard partial differential equations (PDE) [11]. The PDE solver's kernel code, which is obtained by some finite differential method, is called stencil computation because of the memory access pattern that forms predefined stencils across each dimension of the data array. Since stencil computation fundamentally has both spatial and temporal localities, the performance depends significantly on the quality of the instruction scheduling where the size of the data does not fit in the cache memories. However, it is difficult to schedule complicated patterns in the memory access so that traditional cache-based multi-core systems can avoid the contentions of cache lines. To address this problem, optimization schemes have been proposed for parallel resource allocation, such as data reuse [12] and a domain-specific language compiler for stencil computing [13, 14, 16].

1.3 CGRA for Stencil Computations

In contrast to previous studies, we proposed for stencil computations Coarse Grained Reconfigurable Architectures (CGRAs) [20, 21, 24, 25, 22, 23, 26], which have many processing elements, local memories, and inter-connection networks so that many operations can be executed simultaneously on several data streams. We also focused on how to design general-purpose stencil libraries on plural specific CGRAs. The optimization scheme of stencil libraries for CGRA depends closely on the stencil pattern and the sequence of calculations, the frequency and width of the memory system, and the frequency of PEs.

1.4 3D-Stencil Library for EMAX

In this dissertation, we propose a 3D-Stencil Library that can receive input parameters, such as the number of CGRA columns and the degree of stencil calculation, and generate optimized code. For quantitative evaluation, we assume a specific CGRA that we call the Energy-aware Multi-mode Accelerator eXtension (EMAX), which is equipped with distributed single-port local memories and a 2-dimensional interconnection network. To understand easily such a stencil library, it is necessary to grasp the inter-PE structure of CGRA, including columns (horizontal connections), rows (vertical connections), and memory hierarchy. The

EMAX structure will be described below.

1.5 Organization

The rest of this dissertation is organized as follows:

- Section 2 describes the related works about CGRA.
- In section 3, an overview of stencil computing is given.
- In section 4, some key features of EMAX are shown.
- In section 5, the techniques in a 3D-Stencil Library for optimizing and generating codes for EMAX are described.
- In section 6, the performance as measured by certain benchmarks is presented.
- In section 7, we conclude this dissertation

2. Related Work

This section describes the previous works those are either used in, or directly related to this dissertation. Section 2.1 presents an overview of stencil computation and recaps work already done in acceleration it by general purpose accelerators such as Xeon Phi. In section 2.2 describes about Liner Array Pipelined Processor (LAPP) that we have designed for boosting performance under a given power budget.

2.1 CGRA for Stencil Computations

Most stencil computations can be abstracted into a cascaded loop based algorithm, similar to the example in Fig. 1, where the data accesses per each calculation task is generally focusing on a small window of nearest neighbor data elements along the multi-dimensional directions. A high memory access to calculation rate of 7:6 can be observed from the algorithm in in Fig. 1, indicating a memory bound performance caused by expensive L2 misses and off-chip bandwidth.

However, unlike other kinds of memory intensive HPC applications such as Sparse Matrix-Vector Multiplication (SpMV) [30, 31], the most important feature in stencil computation is the possibility of data reuse. Furthermore, taking another feature that the small window or cube is usually firstly moving along the X direction (as shown in Fig. 1) into account, the algorithm of stencil computation can be regarded as a stream data processing one. It is relatively easier to use memory blocking and cache blocking to accelerate them in GPGPU and Xeon Phi. Specifically, paper [39] has provided circular queue and thread blocking as multicorespecific stencil optimizations, where blocking refers to dividing the original $X \times Y \times Z$ data set into small $RX \times RY \times RZ$ blocks that sweep through each thread block. The result in paper [39] indicates that on an eight-core CMP architecture Sun Victoria Fall, all blocking methods including register/core/thread blocking can provide more than 2x speed-up. And a fine thread blocking of 1024 blocks on CUDA achieves 1.4x speed-up as compared to the 64-block division.

This blocking optimization partially coincides the concept of neighbor data access behavior by aggregating a small set of adjacent data into the small L1 cache

for a better data allocation. However, it still does not fully explore the concept of sliding window to ultimately indicate the data re-use in the X,Y,Z directions. In this dissertation, we show a reconfigurable architecture to specifically take care of the data allocation and the reuse in our memory system. Compared to the thread/cache blocking on general purpose accelerators, our work on special hardware has provided the following contributions:

- A memory hierarchy has been proposed to use only singleported RAM to present vector-like per cycle multi-word load/store throughput. An easy-to-program physical memory blocking assignment has been added to instructively match the X-direction neighboring data access pattern and several other well-used patterns in stencil computations and other vectorize-able HPC applications.

- A reconfigurable PE array has been designed beside the memory hierarchy to accelerate the computation of the data-flow-graphs (DFGs) of the innermost loop kernels in these HPC applications, which can provide an IPC near to the number of operations in the loop kernel DFG.

- This 2-D PE array additionally supports to explore the moving window effect along the Y direction for the stencil computations. Data can thus be largely reused in both the X,Y directions for the first and second innermost loops in these applications. For most 3-D stencil algorithms, the data reuse rate is 66%, which accordingly results in an optimally high computation density per each off-chip data load.

Many reconfigurable architectures, including TRIPS[32, 33], CGRA[34], and ADRES[37], have been proposed to accelerate the algorithm DFGs. However, they are not targeting at the stencil applications so that the data movement and reuse are not tuned for this neighboring access pattern. In addition, compared to these reconfigurable architectures, we specially paid attention to the hardware complexity, wire spacing and delay in the proposed special purpose accelerator. Simple network has been used to only assure necessary data bypassing in the stencil DFGs. Extra hardware supports are designed to isolate the wire-delay of the bypassing network from the critical path, which makes it easily to up-scale

the capacity of PE array for a larger problem without influencing the working frequency.

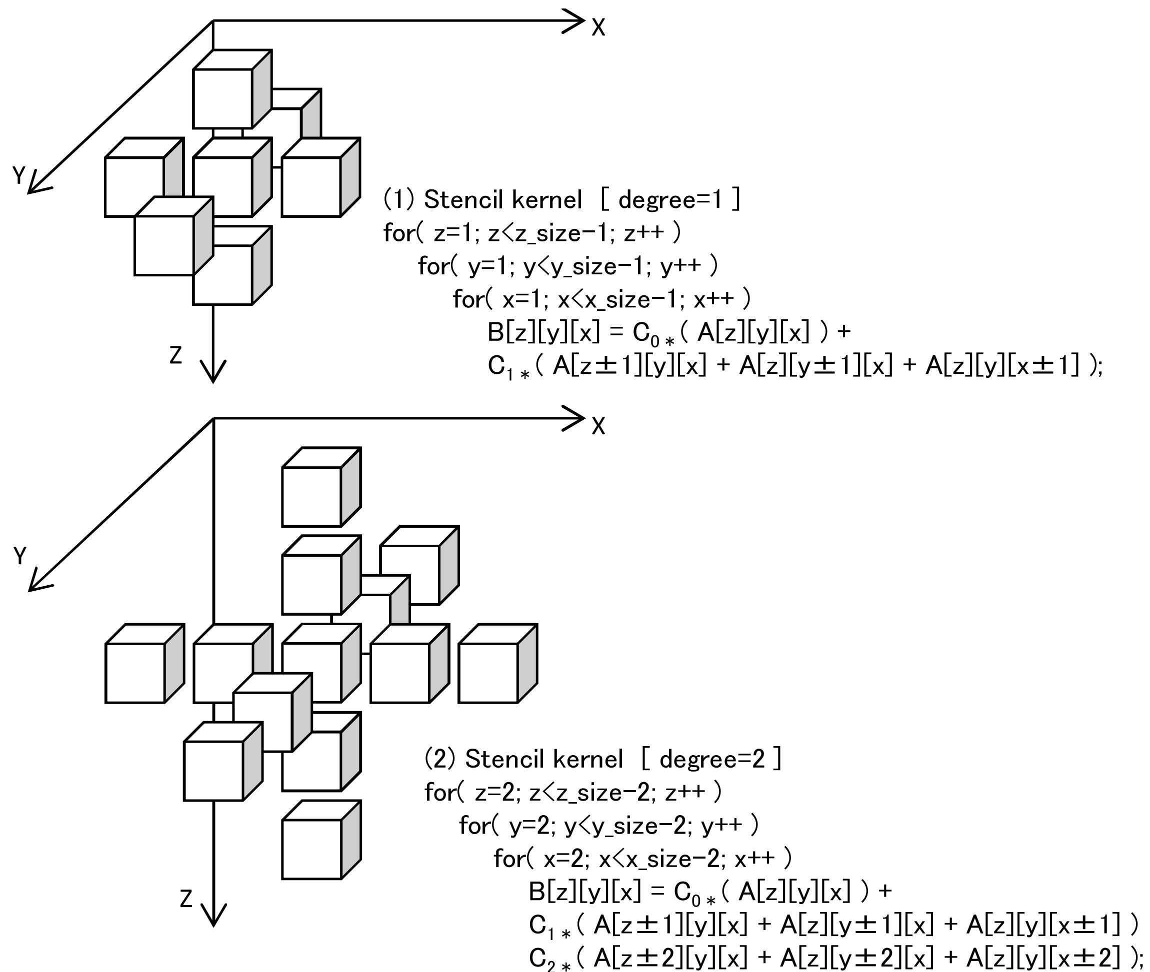


Figure 1. 3D-Stencil Kernels

2.2 Linear Array Pipelined Processor (LAPP)

To focus on both the energy efficiency and the flexibility, the CGRA has been widely studied[35, 36, 38]. However, for most current CGRAs a special compiler is desired to generate the good reconfiguration code. To solve this problem, we

have designed the Liner Array Pipelined Processor (LAPP).

LAPP has been designed and implemented to achieve high power/performance efficiency by ultimately exploiting the parallelism between program loop iterations with a functional unit (FU) array. LAPP includes two parts, as the normal VLIW pipeline part and the FU array part as shown in Fig. 2. The VLIW pipeline makes it enable to take normal VLIW binaries as input, without requiring a special compiler to generate special instruction set architecture (ISA) based binaries. The VLIW pipeline works under a normal execution mode for program parts that have no parallelism. When a loop kernel's iterations can be executed without dependency, the LAPP triggers the array execution mode by mapping the loop kernels vertically along its FU array part.

The FU array takes a structure of multiple array pipeline stages, each of which can map a single VLIW instruction. The multiple array pipeline stages form an array pipeline, which executes instructions in the loop kernel in sequential cycles. Meanwhile, based on the assumption that the mapped loop does not have dependency between its iterations, when the array pipeline stage finishes the execution of the current loop iteration, it can start the execution of the mapped instruction of the next loop iteration.

As shown in paper[27], after filling the array pipeline, the FU array can finish one loop iteration per cycle, resulting in an extremely high speed-up of the loop execution. As in paper[29], LAPP has about nine times power/performance efficiency, as compared to a normal many-core processor with the same chip area.

However, the architecture of LAPP which includes a VLIW pipeline and an FU array also brings some limitations, as:

- The VLIW pipeline processor contains an L1 cache for the data access use, which is an unified memory for all the LAPP processor. During the array execution, the FU array stage also takes data from this L1 cache. For this purpose, L0 buffers are required to propagate data from L1 cache to the actual position of the data load inside the array stages. For an FU array with many stages, the data propagation along the L0 buffers are long and the L0 buffers occupy large amount of areas, which lowers the power and area efficiencies.

- LAPP uses VLIW ISA, which has a very limited amount of LD/ST op-

erations per each instruction. To map memory intensive programs, other than the extensive use of the L0 buffer, the data-flow-graph (DFG) will be largely extended along the vertical direction, which sometimes makes the mapping impossible inside a given number of array stages.

- LAPP generates the mapping information when the loop start instruction is detected by a hardware mapping scheme. The VLIW pipeline, together with this mapper, consumes a large part of area. This further worsens the area efficiency.

- In addition, to pass the data to any register possible in LAPP, it uses a crossbar like interconnection between the array stages. The wire amount of LAPP is thus very huge. This large wire amount gives a drawback in the physical design of the chip, making the critical path long and hard for the place and route tool to get an optimal design[47].

To address these problems, we designed and implemented a novel array accelerator, Energy-aware Multimode array accelerator (EMAX). The feature of EMAX is described in section 4.

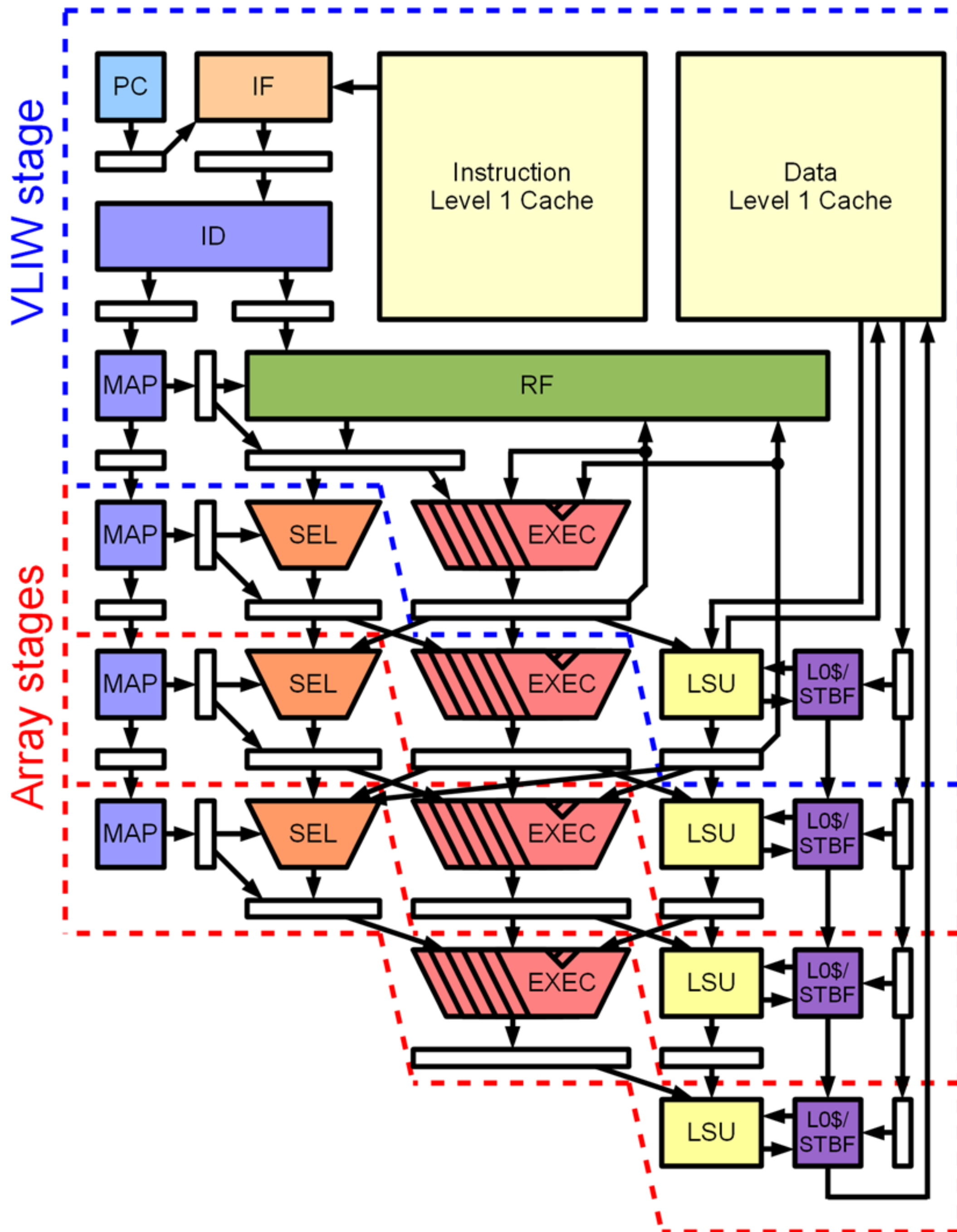


Figure 2. The structure of LAPP

3. Stencil Computation

This section describes the structure of stencil computing that is supported by the 3D-Stencil Library.

Stencil computation is a class of iterative kernels which update array elements according to some fixed pattern, called stencil. They are most commonly found in the codes of computer simulations, e.g. for computational fluid dynamics in the context of scientific and engineering applications. Other notable examples include solving partial differential equations, the Jacobi kernel, the Gauss-Seidel method, image processing and cellular automata. The regular structure of the arrays sets stencil codes apart from other modeling methods such as the Finite element method. Most finite difference codes which operate on regular grids can be formulated as stencil codes.

3.1 3D-Stencil Kernels

Figure 1 shows two examples of 3D-stencil kernels represented by C language. Each of the kernels has neighbors that spread from the center in three directions along the X, Y, and Z axes. The difference between these two examples is the number of elements in each of the three directions. In this dissertation, the number of elements (the distance) from the center is called the "degree". A 7-point stencil kernel has $degree = 1$, and a 13-point stencil kernel has $degree = 2$ (Fig. 1). A 3D-stencil kernel with "$degree = 1$" expresses a 3D Jacobi solver from the Rodinia benchmark suite [17], which is used for the evaluation of heterogeneous computing [19, 18].

4. Overview of EMAX

In this section, such general features of CGRA as columns, rows, mnemonics, and memory hierarchy are described as is our proposed EMAX configuration [24, 25].

4.1 System Diagram of EMAX

As shown in Fig. 3, EMAX consists of two or more basic processing elements (PEs) arranged in the shape of a matrix. Each PE has several arithmetic logical units (e.g., EX1, EX2), distributed single-port local memories (LMM), an effective address generator (EAG), and several FIFOs that can hold a certain amount of recent data read from LMM. In EMAX, the LMM and FIFOs in the same row are connected with a common data path. In the case of the configuration shown in Fig. 3, the data in an LMM can be sent to a maximum of 8 FIFOs in the same row. Additionally, the PEs at the top are connected with the PEs at the bottom in a ring fashion. The number of rows is defined by the number of stages required by specific application programs.

4.2 Interface with Host Computer

In general, accelerators have a local DDR3 memory and are connected to host computers through an external I/O bus (Fig. 3). Consequently, the overhead for sending instructions and data to the accelerators should be taken into consideration for modeling performance.

4.3 EMAX mnemonics

For describing the 3D-Stencil Library, it is convenient to prepare some typical mnemonics for controlling the EMAX. The framework of the mnemonics is shown in Fig. 4. The preceding combination of *row* and *col* specifies the logical location of PE to be assigned with the succeeding function. To explain the EMAX operation, we describe three load instructions in the same Z coordinates (Fig. 5(1) at $Y = 0, 1, 2$ and Fig 5(2) at $Y = 1, 2, 3$). Each load instruction can increment the load address in the X axis *count* times. Therefore, one load instruction can read the data of the *count* points from the stream data along the X axis. In the 1st

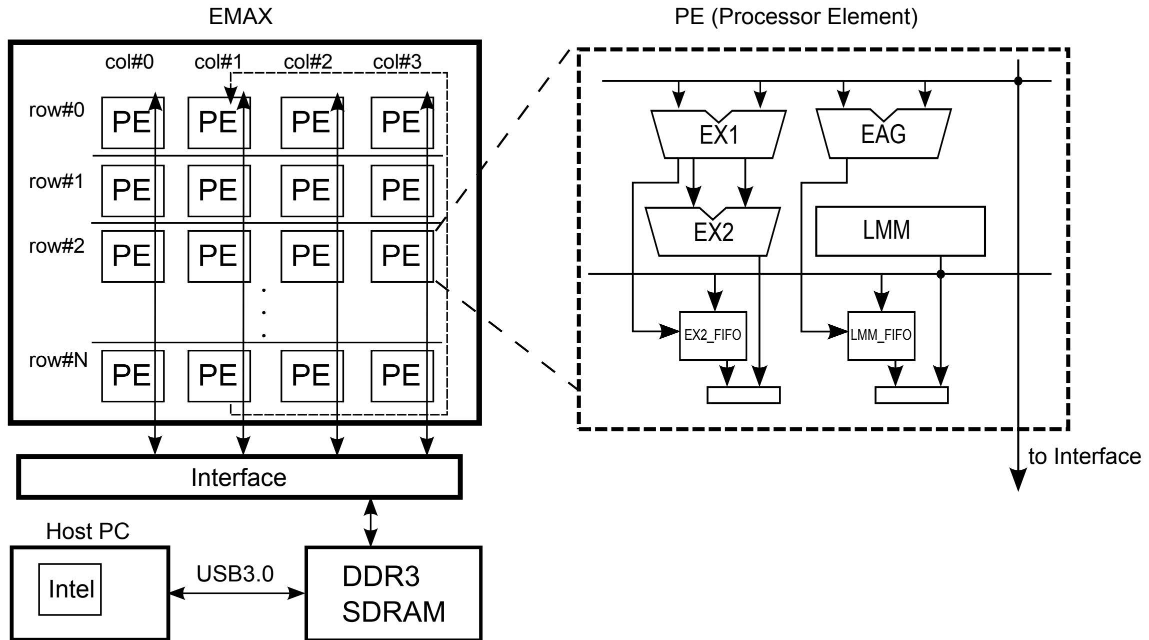


Figure 3. EMAX configuration

processing on EMAX at $Y = 0, 1, 2$ (Fig 5(1)), three load instructions read the *count* points data from the stream data along the X axis (three stream data are $Y = 0$,$Y = 1$, and $Y = 2$). Then these three load instructions are mapped on PE at *rows 0, 2, 4* (Fig. 5(3)). After the 1st processing is completed, the next processing starts to read the *count* points data from the three stream data at $Y = 1, 2, 3$ (Fig. 5(2)). Then the two data streams at $Y = 1$ and $Y = 2$ on the X axis are used on PE at *rows 2 and 4* in the 1st processing. However, because these data exist in the LMM of another PE when the instruction is similarly arranged, it cannot be read. To reuse these data streams, the *Dist* field exists in the EMAX mnemonics. *Dist* specifies the distance of the vertical location between LMMs that hold neighbor streams for each other for the stencil computation. After EMAX has finished processing on a stencil stream at $Y = 0, 1, 2$ along the X axis (Fig. 5(1)), some LMMs can supply data for the next processing on the next stencil stream at $Y = 1, 2, 3$, because the neighbor stencil streams overlap each other at $Y = 1$ and $Y = 2$ (Fig. 5(2)). If the LMM that holds the stream

corresponding to $Y = 1$ is located on *row 2*, and $Y = 2$ is located on *row 4*, then *dist* should be 2. For the next processing with $Y = 1, 2, 3$, the previous contents of LMMs are kept and the mapping of instructions is shifted by *dist*, as shown in Fig. 5(4). Consequently, the next processing can reuse the streams at $Y = 1$ and $Y = 2$ (black LMMs in Fig. 5(4)). The operations for EX1 and EX2 are specified by *ALU_OP*, as shown in Figs. 4(b) and (c). The load operation from EX2_FIFO can be specified as EX1 operations. The *MEM_OP* (Fig. 4(d)) is the load/store operations from LMM or LMM_FIFO. The initial value of the register for *ALU_OP* and *MEM_OP* can be specified by *RGI*.

4.4 Instruction Mapping

In this section, we describe instruction mapping method to EMAX with unsharp kernel that used for image processing. Figure 6 shows unsharp() program written in C language, and Fig 7 expresses EMAX mnemonics for executing that unsharp() program. First, each PEs of col#0 loads input array data to own LMM (col#0 raw#0, col#0 raw#1, col#0 raw#2). In 2-dimension stencil kernel like unsharp, it is used three contiguous data to calculate a stencil ([p1, p5, p2], [p6, p0, p7] and [p3, p8, p4] in Fig. 6. Therefore it is mapped three load instructions in same row. Then, each instruction of PEs of col#1 and col#2 can load data from LMM-FIFO that PE of col#0 stored. Nine loaded data is propagated by a register, and each data are operated by each PE. The final operation result is stored by PE of raw#8, col#0 to LMM. All PEs execute operation *count* times. In this case, *count* is set to the number of stencil stream size(WD). Each PE works in parallel, therefore the result to be provided by 9 load instructions, 21 ALU operations, and 1 store instruction is obtained by 1 cycle with EMAX. *mauh/mauh3* expresses the instruction that add upper 16bit of Xr, Yr, Zr and lower 16bit respectively. *mluh* is an instruction to multiply Yr and upper 16bit and lower 16bit of Xr as 8bit data. *mh2bw* is to merge upper 16bit of Xr, Yr and lower 16bit as 8bit data.

Case 1:@row#, col#, dist [count] ALU_OP RGI & MEM_OP RGI LMM_CONTROL
Case 2:@row#, col#, dist [count] ALU_OP RGI
Case 3:@row#, col#, dist [count] & MEM_OP RGI LMM_CONTROL

(a) Instruction Format

32 bit operations	add/add3/sub/sub3
16 bit[2] operations	mauh/mauh3/msuh3
misc operations	mluh/mmrg3/msad/minl/minl3/mh2bw/mcas/ mmid3/mmax/mmax3/mmin/mmin3
load from FIFO	ldb/ldub/ldh/lduh/ld
floating-point operations	fmul/fma3/fadd

(b) EX1 operations

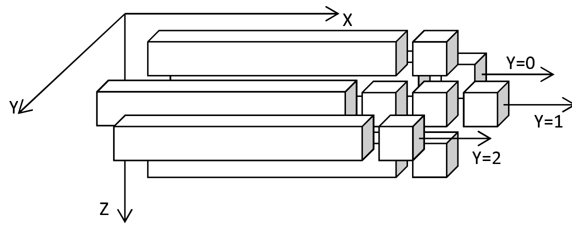
32 bit operations	and/or/xor
16 bit[2] operations	mauh/mauh3/msuh3

(c) EX2 operations

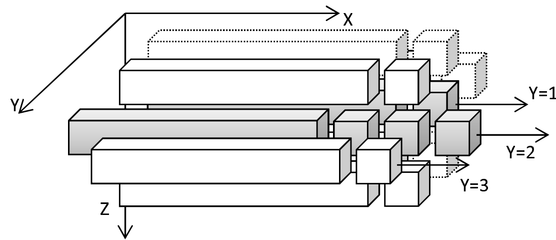
load from LMM or LMM_FIFO	ldb/ldub/ldh/lduh/ld
store to LMM	stb/sth/st/cst

(d) Memory operations

Figure 4. EMAX mnemonics



(1) 1st processing on EMAX at Y=0,1,2



(2) 2nd processing on EMAX at Y=1,2,3

	Instruction	LMM
row 0	Load [stream : Y=0]	stream : Y=0
row 1		
row 2	Load [stream : Y=1]	stream : Y=1
row 3		
row 4	Load [stream : Y=2]	stream : Y=2
row 5		
row 6		

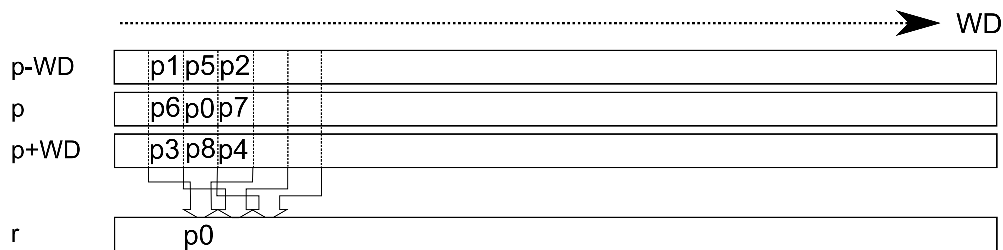
(3) Instruction location in 1st processing

	Instruction	LMM
row 0		
row 1		
row 2	Load [stream : Y=1]	stream : Y=1
row 3		
row 4	Load [stream : Y=2]	stream : Y=2
row 5		
row 6	Load [stream : Y=3]	stream : Y=3

mapping of Instructions is shifted by “dist” (dist=2)

(4) Instruction location in 2nd processing

Figure 5. Instruction shift by “dist”




```
void unsharp(unsigned char *p, unsigned char *r)
{
    int t0,t1,t2,  j, k;
    int p0 = ((0  )*WD+(1  ))*4;
    int p1 = ((0-1)*WD+(1-1))*4;
    int p2 = ((0-1)*WD+(1+1))*4;
    int p3 = ((0+1)*WD+(1-1))*4;
    int p4 = ((0+1)*WD+(1+1))*4;
    int p5 = ((0-1)*WD+(1  ))*4;
    int p6 = ((0  )*WD+(1-1))*4;
    int p7 = ((0  )*WD+(1+1))*4;
    int p8 = ((0+1)*WD+(1  ))*4;
    for (j=0; j<WD; j++) {
        r[p0+0] = 0;
        t0 = p[p0+1];
        t1 = p[p1+1] + p[p2+1] + p[p3+1] + p[p4+1];
        t2 = p[p5+1] + p[p6+1] + p[p7+1] + p[p8+1];
        r[p0+1] = limitRGB(( t0 * 239 - t1 * 13 - t2 * 15 - t2/4) >> 7);

        t0 = p[p0+2];
        t1 = p[p1+2] + p[p2+2] + p[p3+2] + p[p4+2];
        t2 = p[p5+2] + p[p6+2] + p[p7+2] + p[p8+2];
        r[p0+2] = limitRGB(( t0 * 239 - t1 * 13 - t2 * 15 - t2/4) >> 7);

        t0 = p[p0+3];
        t1 = p[p1+3] + p[p2+3] + p[p3+3] + p[p4+3];
        t2 = p[p5+3] + p[p6+3] + p[p7+3] + p[p8+3];
        r[p0+3] = limitRGB(( t0 * 239 - t1 * 13 - t2 * 15 - t2/4) >> 7);

        p0+=4; p1+=4; p2+=4; p3+=4; p4+=4; p5+=4; p6+=4; p7+=4; p8+=4;
    }
}
unsigned char limitRGB(int c) {
    if (c<0x00) return 0x00;
    if (c>0xff) return 0xff;
    return c;
}
```

Figure 6. Source code of unsharp()

```
//EMAX2 @0,0,0 [WD]                                   & ld (r10+=,4),r1 rgi[p1]  lmr[.p-WD:,WD,0]
//EMAX2 @0,1,0 [WD]                                   & ld (r10+=,4),r2 rgi[p2]
//EMAX2 @0,2,0 [WD]                                   & ld (r10+=,4),r5 rgi[p5]

//EMAX2 @1,0,0 [WD] mauh  (r1.l,r2.l),r11             & ld (r10+=,4),r6  rgi[p6] lmr[.p:,WD,0]
//EMAX2 @1,1,0 [WD]                                   & ld (r10+=,4),r7  rgi[p7]
//EMAX2 @1,2,0 [WD] mauh  (r1.h,r2.h),r12             & ld (r10+=,4),r0  rgi[p0]

//EMAX2 @2,0,0 [WD] mluh  (r0.l,ri),r20   rgi[,.i239c0:]& ld (r10+=, 4),r3 rgi[p3] lmr[.p+WD:,WD,0]
//EMAX2 @2,1,0 [WD] mluh  (r0.h,ri),r21  rgi[,.i239c1:]& ld (r10+=, 4),r4 rgi[p4]
//EMAX2 @2,2,0 [WD] mauh  (r5.l,r6.l),r15             & ld (r10+=, 4),r8 rgi[p8]
//EMAX2 @2,3,0 [WD] mauh  (r5.h,r6.h),r16

//EMAX2 @3,0,0 [320] mauh3 (r11,r3.l,r4.l),r11
//EMAX2 @3,1,0 [320] mauh3 (r12,r3.h,r4.h),r12

//EMAX2 @4,0,0 [320] mluh  (r11,ri),r13   rgi[,.i13c0:]
//EMAX2 @4,1,0 [320] mluh  (r12,ri),r14   rgi[,.i13c1:]
//EMAX2 @4,2,0 [320] mauh3 (r15,r7.l,r8.l),r15
//EMAX2 @4,3,0 [320] mauh3 (r16,r7.h,r8.h),r16

//EMAX2 @5,0,0 [320] | or  (r15,0)>M2,r7
//EMAX2 @5,1,0 [320] mluh  (r15,ri),r17   rgi[,.i15c0:]
//EMAX2 @5,2,0 [320] | or  (r16,0)>M2,r8
//EMAX2 @5,3,0 [320] mluh  (r16,ri),r18   rgi[,.i15c1:]

//EMAX2 @6,0,0 [320] msuh3 (r20,r7,r17),r20
//EMAX2 @6,2,0 [320] msuh3 (r21,r8,r18),r21

//EMAX2 @7,0,0 [320] msuh (r20,r13) | or (-,0)>M7,r20
//EMAX2 @7,2,0 [320] msuh (r21,r14) | or (-,0)>M7,r21

//EMAX2 @8,0,0 [320] mh2bw (r21,r20)                  & st -,(ri+=,4)   rgi[.p0_out:,] lmw[.r:,320,0]
```

```
mauh ( Xr.{fhl}, Yr.{fhl} )
  16bit[2] Xr + Yr

mauh3 ( Xr.{fhl}, Yr.{fhl} )
  16bit[2] Xr + Yr + Zr

mluh  ( Xr.{fhl}, Yr.{fhl}, Zr.{fhl}  )
   8bit[2] * 9bit → 16bit[2]

mh2bw ( Xr, Yr )
   merge sat(Xr.H16bit).sat(Xr.L16bit)
             .sat(Yr.H16bit).sat(Yr.L16bit)
------------------------------------------------------------
{fhl}   f:fullword  h:byte3,byte2→H16bit,L16bit
        l: byte1,byte0→H16bit,L16biy
```

Figure 7. EMAX mnemonics of unsharp()

5. 3D-Stencil Library for EMAX

This section describes the user interface of the 3D-Stencil Library and a technique for generating instructions for EMAX. For the best use of EMAX, we must reduce the amount of data transmission between LMM and the main memory (DDR3). The optimization scheme must focus on how to map local memory to the stencil streams.

5.1 User Interface of 3D-Stencil Library

As shown in Fig. 1, typical templates for stencil kernels are easily written in C language. The parameters for customizing the templates and implementation on EMAX are the "degree" of the stencil and the "number of columns" of EMAX. Fig. 8(1) is an example of a customizable stencil library. When the degree is one, the Jacobi 3D stencil computation is expressed, as described in Section 3. When the degree is three, a FD6 kernel is expressed. Application developers can embed various stencil computations on various EMAXs by customizing 3D-Stencil Libraries with specific parameters (Fig. 8(2)). When we use EMAX without the 3D-Stencil Library, we have to write the program with EMAX mnemonics shown in Fig. 9 and Fig. 10. With 3D-Stencil Library, we can write the program for executing stencil computing with only one line.

5.2 Structure of 3D-Stencil Library

The structure of the 3D-Stencil Library is shown in Fig. 11. When an application program calls the stencil library, the library first generates EMAX instructions based on the given parameters. Then the instructions and the data are sent to EMAX through DDR3. Finally, EMAX executes the instructions mapped on PEs simultaneously and stores the result in DDR3. The following is the processing flow of using the 3D-Stencil Library: efdn

(1) The stencil application allocates three memory arrays in the main memory of the host PC. A is the input 3D-array, B is the output 3D-array, and C is the symmetric-constant coefficients of the stencil [5].

(1) Interface of 3D-Stencil Library

```
stencil_3d( double ***A, double ***B, double *C, int size_x, int size_y, int size_z,
            int degree, int e_stage );
```

---

(2) Parameters

```
double ***A       : input 3D array
double ***B       : output  3D array
double    *C      : symmetrical constant coefficient
int       size_X  : dimension size of X-direction
int       size_Y  : dimension size of Y-direction
int       size_Z  : dimension size of Z-direction
int       degree  : number of elements from center point
int       e_stage : number of EMAX columns
```

Figure 8. Interface of stencil kernel

```
EMAX2 @0,0,1 [320]  add (ri+=,4),r0      rgi[.emax_rgi_p0____jacobi:,]  &
EMAX2 @0,1,1 [320]                                          & ld (ri+=,4),r1     rgi[.emax_rgi_CURR_A0_jacobi:,] lmr[.emax_lmr_CURR_A0_jacobi:,320,0]
EMAX2 @1,1,1 [320]  fmul (ri,r1),r10     rgi[.emax_rgi_C20_jacobi:,]    & ld (r0,-1280),r2   lmr[.emax_lmr_PREV_A1_jacobi:,320,0]
EMAX2 @2,1,1 [320]  fma3 (ri,r2,r10),r10 rgi[.emax_rgi_C21_jacobi:,]    & ld (r0,4),r5       lmr[.emax_lmr_CURR_A1_jacobi:,320,0]
EMAX2 @2,2,1 [320]                                          & ld (r0,0),r4
EMAX2 @2,3,1 [320]                                          & ld (r0,-4),r3
EMAX2 @3,1,1 [320]  fma3 (ri,r5,r10),r10 rgi[.emax_rgi_C22_jacobi:,]    & ld (r0,1280),r6    lmr[.emax_lmr_NEXT_A1_jacobi:,320,0]
EMAX2 @3,2,1 [320]  fmul (ri,r4),r11     rgi[.emax_rgi_C10_jacobi:,]    & ld (ri+=,4),r7     rgi[.emax_rgi_CURR_A2_jacobi:,] lmr[.emax_lmr_CURR_A2_jacobi:,320,0]
EMAX2 @3,3,1 [320]  fmul (ri,r3),r12     rgi[.emax_rgi_C23_jacobi:,]    &
EMAX2 @4,1,1 [320]  fma3 (ri,r6,r10),r10 rgi[.emax_rgi_C24_jacobi:,]    &
EMAX2 @4,2,1 [320]  fma3 (ri,r7,r11),r11 rgi[.emax_rgi_C25_jacobi:,]    &
EMAX2 @5,1,1 [320]  fadd (r10,r11),r10                      &
EMAX2 @6,1,1 [320]  fadd (r10,r12),r10                      &
EMAX2 @7,1,1 [320]                                          & st r10,(ri+=,4)    rgi[.emax_rgi_store_jacobi:,] lmw[.emax_lmw_store_jacobi:,320,0]■jacobi
EMAX2 @0,0,1 [320]  add (ri+=,4),r0      rgi[.emax_rgi_p0____jacobi:,]  &
EMAX2 @0,1,1 [320]                                          & ld (ri+=,4),r1     rgi[.emax_rgi_CURR_A0_jacobi:,] lmr[.emax_lmr_CURR_A0_jacobi:,320,0]
EMAX2 @1,1,1 [320]  fmul (ri,r1),r10     rgi[.emax_rgi_C20_jacobi:,]    & ld (r0,-1280),r2   lmr[.emax_lmr_PREV_A1_jacobi:,320,0]
EMAX2 @2,1,1 [320]  fma3 (ri,r2,r10),r10 rgi[.emax_rgi_C21_jacobi:,]    & ld (r0,4),r5       lmr[.emax_lmr_CURR_A1_jacobi:,320,0]
EMAX2 @2,2,1 [320]                                          & ld (r0,0),r4
EMAX2 @2,3,1 [320]                                          & ld (r0,-4),r3
EMAX2 @3,1,1 [320]  fma3 (ri,r5,r10),r10 rgi[.emax_rgi_C22_jacobi:,]    & ld (r0,1280),r6    lmr[.emax_lmr_NEXT_A1_jacobi:,320,0]
EMAX2 @3,2,1 [320]  fmul (ri,r4),r11     rgi[.emax_rgi_C10_jacobi:,]    & ld (ri+=,4),r7     rgi[.emax_rgi_CURR_A2_jacobi:,] lmr[.emax_lmr_CURR_A2_jacobi:,320,0]
EMAX2 @3,3,1 [320]  fmul (ri,r3),r12     rgi[.emax_rgi_C23_jacobi:,]    &
EMAX2 @4,1,1 [320]  fma3 (ri,r6,r10),r10 rgi[.emax_rgi_C24_jacobi:,]    &
EMAX2 @4,2,1 [320]  fma3 (ri,r7,r11),r11 rgi[.emax_rgi_C25_jacobi:,]    &
EMAX2 @5,1,1 [320]  fadd (r10,r11),r10                      &
EMAX2 @6,1,1 [320]  fadd (r10,r12),r10                      &
EMAX2 @7,1,1 [320]                                          & st r10,(ri+=,4)    rgi[.emax_rgi_store_jacobi:,] lmw[.emax_lmw_store_jacobi:,320,0]
```

Figure 9. Source Code of Stencil Computing with EMAX mnemonics [jacobi]

```
EMAX2 @0,0,1 [320]  add (ri+=,4),r10     rgi[.emax_rgi_p0____fd6:,]  & ld (ri+=,4),r0     rgi[.emax_rgi_CURR_A0_fd6:,] lmr[.emax_lmr_CURR_A0_fd6:,320,0]
EMAX2 @0,1,1 [320]                                               & ld (ri+=,4),r1     rgi[.emax_rgi_CURR_A1_fd6:,] lmr[.emax_lmr_CURR_A1_fd6:,320,0]
EMAX2 @0,2,1 [320]                                               & ld (ri+=,4),r2     rgi[.emax_rgi_CURR_A2_fd6:,] lmr[.emax_lmr_CURR_A2_fd6:,320,0]
EMAX2 @1,0,1 [320]  fmul(ri,r0),r20      rgi[.emax_rgi_C40_fd6:,]    & ld (r10,-3840),r3  lmr[.emax_lmr_PREV3_A3_fd6:,320,0]
EMAX2 @1,1,1 [320]  fmul(ri,r1),r21      rgi[.emax_rgi_C30_fd6:,]    &
EMAX2 @1,2,1 [320]  fmul(ri,r2),r22      rgi[.emax_rgi_C20_fd6:,]    &
EMAX2 @2,0,1 [320]  fma3(ri,r3,r20),r20  rgi[.emax_rgi_C41_fd6:,]    & ld (r10,-2560),r4  lmr[.emax_lmr_PREV2_A3_fd6:,320,0]
EMAX2 @2,1,1 [320]  fadd(r21,r22),r21                              &
EMAX2 @3,0,1 [320]  fma3(ri,r4,r20),r20  rgi[.emax_rgi_C31_fd6:,]    & ld (r10,-1280),r5  lmr[.emax_lmr_PREV1_A3_fd6:,320,0]
EMAX2 @3,1,1 [320]                                               &
EMAX2 @4,0,1 [320]  add (ri+=,4),r10     rgi[.emax_rgi_p1____fd6:,]  & ld (r10,12),r12    lmr[.emax_lmr_CURR_A3_fd6:,320,0]
EMAX2 @4,1,1 [320]                              ld (r10,4),r10  & ld (r10,8),r11
EMAX2 @4,2,1 [320]                              ld (r10,-4),r8  & ld (r10,0),r9
EMAX2 @4,3,1 [320]                              ld (r10,-12),r6 & ld (r10,-8),r7
EMAX2 @5,0,1 [320]  fma3(ri,r5,r20),r20  rgi[.emax_rgi_C21_fd6:,]    & ld (r10,1280),r13  lmr[.emax_lmr_NEXT1_A3_fd6:,320,0]
EMAX2 @5,1,1 [320]  fma3(ri,r10,r21),r21 rgi[.emax_rgi_C22_fd6:,]    &
EMAX2 @5,2,1 [320]  fmul(ri,r8),r22      rgi[.emax_rgi_C23_fd6:,]    &
EMAX2 @5,3,1 [320]  fmul(ri,r6),r23      rgi[.emax_rgi_C42_fd6:,]    &
EMAX2 @6,0,1 [320]  fma3(ri,r12,r20),r20 rgi[.emax_rgi_C43_fd6:,]    & ld (r10,2560),r14  lmr[.emax_lmr_NEXT2_A3_fd6:,320,0]
EMAX2 @6,1,1 [320]  fma3(ri,r11,r21),r21 rgi[.emax_rgi_C32_fd6:,]    &
EMAX2 @6,2,1 [320]  fma3(ri,r9,r22),r22  rgi[.emax_rgi_C10_fd6:,]    &
EMAX2 @6,3,1 [320]  fma3(ri,r7,r23),r23  rgi[.emax_rgi_C33_fd6:,]    &
EMAX2 @7,0,1 [320]                                               & ld (r10,3840),r15  lmr[.emax_lmr_NEXT3_A3_fd6:,320,0]
EMAX2 @7,1,1 [320]                                               & ld (ri+=,4),r16    rgi[.emax_rgi_CURR_A4_fd6:,] lmr[.emax_lmr_CURR_A4_fd6:,320,0]
EMAX2 @7,2,1 [320]  fma3(ri,r13,r22),r22 rgi[.emax_rgi_C24_fd6:,]    & ld (ri+=,4),r17    rgi[.emax_rgi_CURR_A5_fd6:,] lmr[.emax_lmr_CURR_A5_fd6:,320,0]
EMAX2 @7,3,1 [320]  fma3(ri,r14,r23),r23 rgi[.emax_rgi_C34_fd6:,]    & ld (ri+=,4),r18    rgi[.emax_rgi_CURR_A6_fd6:,] lmr[.emax_lmr_CURR_A6_fd6:,320,0]
EMAX2 @8,0,1 [320]  fma3(ri,r15,r20),r20 rgi[.emax_rgi_C44_fd6:,]    &
EMAX2 @8,1,1 [320]  fma3(ri,r16,r21),r21 rgi[.emax_rgi_C25_fd6:,]    &
EMAX2 @8,2,1 [320]  fma3(ri,r17,r22),r22 rgi[.emax_rgi_C35_fd6:,]    &
EMAX2 @8,3,1 [320]  fma3(ri,r18,r23),r23 rgi[.emax_rgi_C45_fd6:,]    &
EMAX2 @9,1,1 [320]  fadd(r20,r21),r21                              &
EMAX2 @9,2,1 [320]  fadd(r22,r23),r22                              &
EMAX2 @10,2,1 [320] fadd(r21,r22),r22                              &
EMAX2 @11,0,1 [320]                                              & st r22,(ri+=,4)    rgi[.emax_rgi_store_fd6:,] lmw[.emax_lmw_store_fd6:,320,0]
```

Figure 10. Source Code of Stencil Computing with EMAX mnemonics [FD6]

(2) The stencil application calls the 3D-Stencil Library. Arguments `A`, `B`, and `C` are the address pointers allocated in the main memory by the stencil application (1), and `size_X`, `size_Y`, and `size_Z` are the dimension sizes of the 3D-array.

(3) The 3D-Stencil Library automatically generates instructions (mapping data) of the EMAX including input 3D-array data from the input parameter. If the data size of the X-direction exceeds the LMM capacity, the 3D-Stencil Library divides the 3D-Stencil space and executes EMAX multiple times. Fig. 12 shows the divided image.

(4) The 3D-Stencil Library transmits the instruction data including the data for the EMAX activation to EMAX's DDR3 with DMA.

(5) When the activation data are written on DDR3, EMAX is activated automatically and prefetches the input 3D-array data from the DDR3 to the

LMMs. Then each PE of EMAX executes the instructions and sends the result to DDR3.

(6) The result of the stencil computation is transmitted by DDR3 to the pointer of array B in the main memory of the host PC.

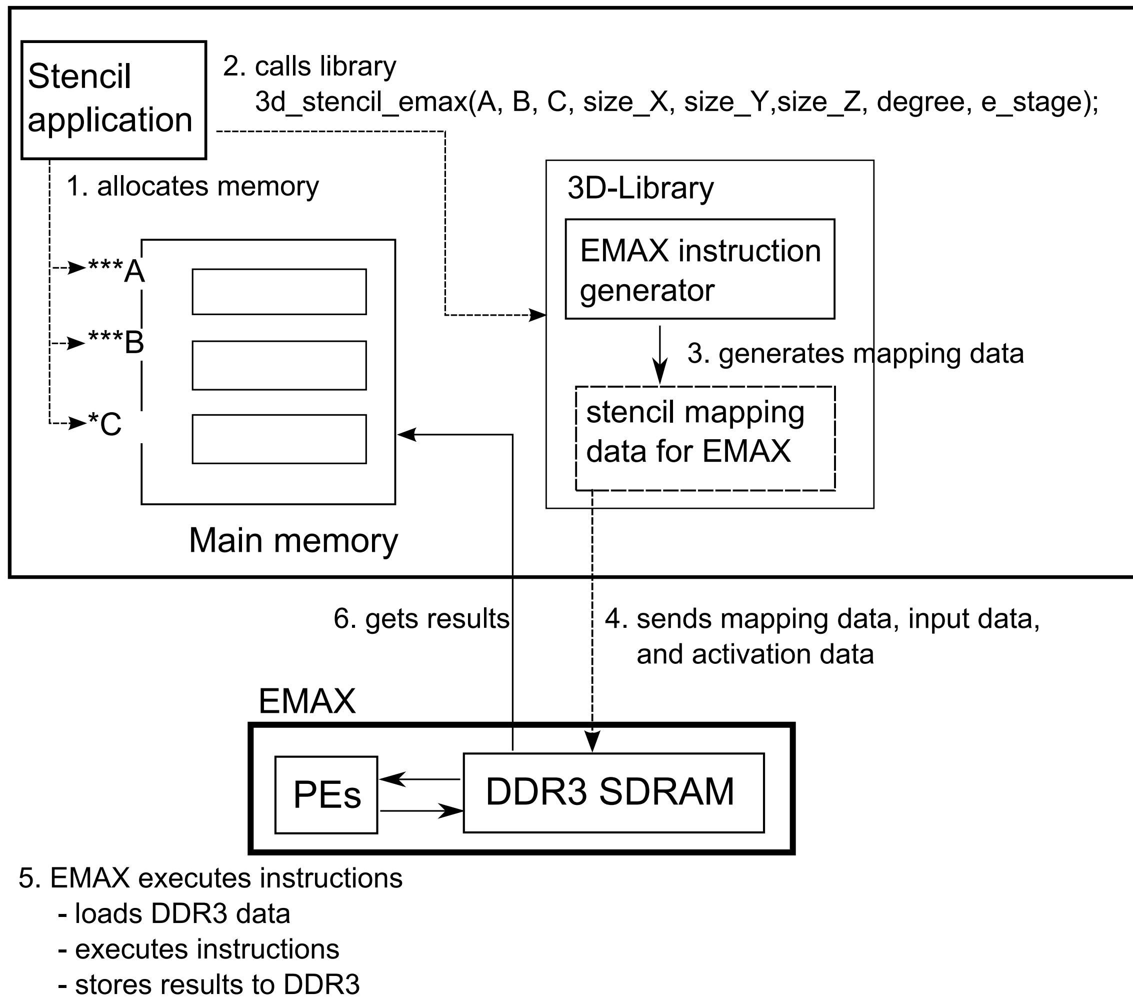


Figure 11. Structure of 3D-Stencil Library

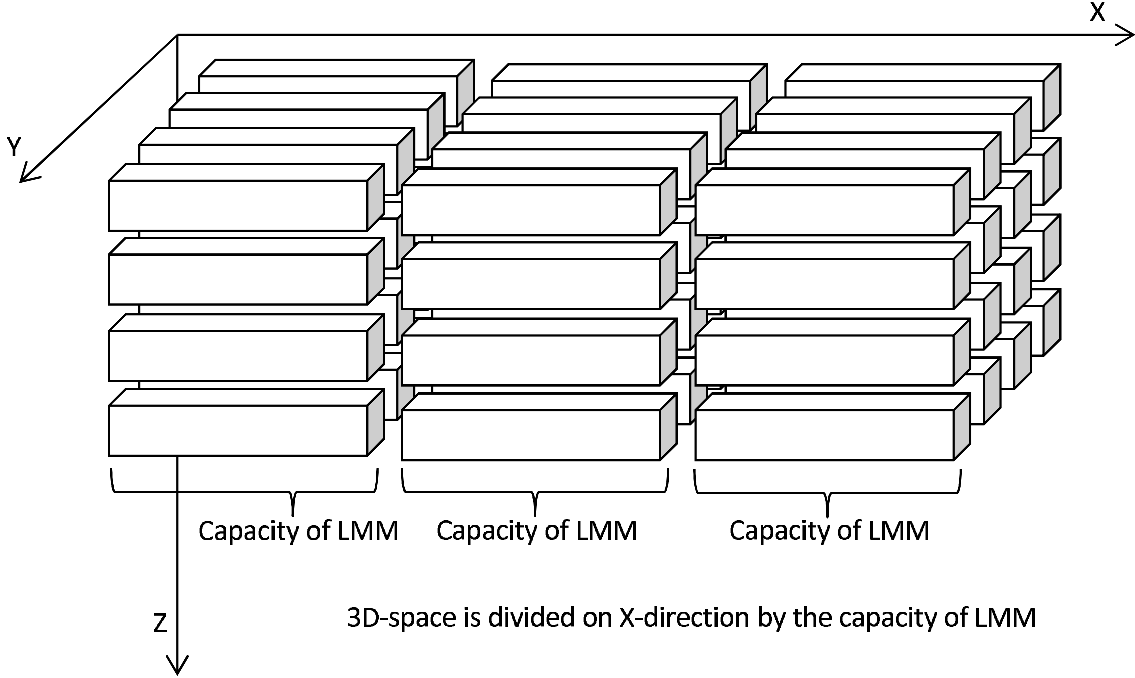


Figure 12. Divided 3D-Stencil space

5.3 Basic Instruction Mapping

As mentioned above, the amount of data, which is transmitted between LMMs and the main memory (DDR3), must be reduced to increase EMAX's execution efficiency. The stencil computation has spatial (horizontal) locality in the X-direction and temporal (vertical) locality in the Y-direction. Horizontal locality can be maximally utilized by employing FIFOs in the same row of the EMAX. The FIFOs are filled with the data from the LMM in the same row and can hold several neighbors in the X-direction. In contrast, vertical locality can be maximally utilized by reusing LMMs that hold different streams in the Y-directions. Fig. 13 describes the basic memory mapping of stencil computation when the "degree" equals one. Fig. 13(1) shows the load instructions for a stencil kernel expressed by EMAX mnemonics. First, seven load instructions are mapped on corresponding EAGs (Fig. 13(2)). Then for utilizing FIFOs to load the neighbor data in the X-direction, the load instructions from $A[z][y][x-1]$, $A[z][y][x]$, and $A[z][y][x+1]$

must be mapped in the same row, and the load instructions from $A[z][y-1][x]$, $A[z][y][x-1]$, and $A[z][y+1][x]$ must be mapped in the same column to reuse the data in the LMMs. By setting each *dist* field of the instructions to 1, subsequent processing can reuse the LMMs of $A[z][y][[x]$ and $A[z][y+1][x-1]$ (Fig. 13(3)).

5.4 Packing of Instructions

After basic mapping is completed, the packing of instructions should be considered by unrolling in the Z-direction. As mentioned above, the X-direction is mapped on the sequential access with FIFOs, and the Y-direction is mapped on the reusing of LMMs. Moreover, we have another chance to parallelize the Z-direction by unrolling so that two contiguous stencil computations can be performed simultaneously. An example of the program and an arrangement for unrolling is shown in Fig. 14. The load instructions from $A[z][y][x-1]$, $A[z][y][x]$, $A[z][y][x+1]$, $A[z][y-1][x]$, and $A[z][y+1][x]$ are defined as a fixed pattern in the 3D-Stencil Library due to utilizing FIFOs and reusing LLMs. The neighbor stencil on the Z-direction is mapped on empty PEs in a mirrored fashion (Fig. 14(2)). By having unrolled codes share two load instruction ($A[z][y][x]$, $A[z+1][y][x]$), two load instructions can be eliminated. Therefore, we expect that the packing of instructions reduces the total amount of data to be transmitted between LMMs and the main memory (DDR3).

5.5 Evaluation of Packing

If we assume that the target EMAX has an infinite number of rows, the number of parallel mappings in the Z-direction can theoretically be increased infinitely. To compute the optimal degree of parallelizing in the Z-direction based on the number of EMAX rows, the amount of data to be transmitted in each case is evaluated. The flow of prefetching data to LMM in the case of stencil computation is shown in Fig. 15, where each block corresponds to a stream in the X-direction and is stored in each LMM. The two dimensions of Y and Z in Fig. 15 correspond to the Y- and X-directions. The amount of data to be transmitted to LMM can be estimated by the following formulas: p: number of parallel mappings, d: number of stencil degrees, y: size of Y-direction, and z: size of Z-direction.

(1) First required bit of data: $(4d+1)+p(2d+1)$

(2) Subsequent required data: $(2d+1)+(p-1)=2d+p$

(3) Total data in xy-iteration: $y(2d+p)$

(4) Total data in xyz-iteration: $z((1)+(3))/p$

(5) Total data (approximate value): $yz(2d+p)/p$

Formula (1) is the amount of data to be transmitted to LMM at the first iteration of the Y-direction, and formula (2) is the amount of data required by the subsequent incremental stencil in the Y-direction. The total amount of data with the incremental to the Y-direction is expressed by formula (3) (i.e., $(2)*y$). Since the number of iterations in the Z-direction (z) is divided by the number of parallel mappings (p), formula (4) becomes the total amount of data to be transmitted to LMM. Since the stencil calculation has many dimensions ($p < y$), the value of (1) can be ignored, and the amount of required data can be approximated as the value of formula (5). Therefore, the more p increases, the more the required data are reduced. In the case of $degree = 1$, the ratio of without parallelization ($p = 1$) to with parallelization becomes $3 : (2+p)/p$. When p is assumed to be infinite, the ratio becomes $3 : 1$. Therefore, by parallelization, the amount of data to be transmitted is reduced to a maximum of 1/3 of that without parallelization, and for $degree = 2$ and $degree = 3$, it is reduced to a maximum of 1/5 and 1/7, respectively.

5.6 Parallel Mapping in 3D-Stencil Library

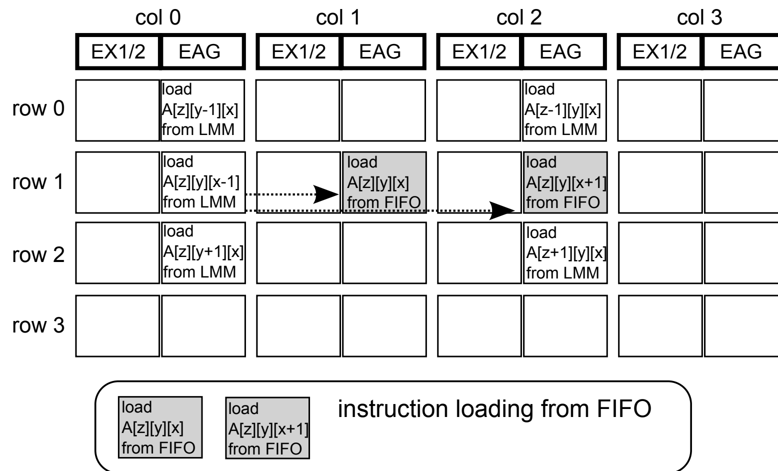
It has become obvious that parallelization can reduce the amount of data transmission based on the above evaluation. Therefore, the 3D-Stencil Library must automatically generate instructions to increase the number of parallel mappings at least to the maximum number of EMAX rows. First, the 3D-Stencil Library maps the instructions according to the basic pattern. When $degree = 1$, the basic pattern is copied in seven rows (Fig. 16(1)). If the number of EMAX columns is seven or less, the 3D-Stencil Library maps the instructions in the form of Fig. 16(1) (i.e., $p = 1$). If the number of EMAX columns is eight or more, for

computing the neighbor stencil in the Z-direction, the basic pattern occupies the flipped horizontal location next to the original pattern (Fig. 16(2) (i.e., $p = 2$)). When the number of sets for parallel mapping is 3 or 4, the basic pattern is copied and put in the lower location. Then the added number of the EMAX rows becomes four because the empty units are used (Fig. 16(3)). For example, when the number of EMAX columns is 11 or 12, the number of parallel mappings becomes three or four, respectively. On the other hand, if $degree = 3$, since the PEs in the same row are occupied by the basic pattern using eleven rows, all of the basic patterns should be located in subsequent rows. The number of rows, generated by the 3D-Stencil Library according to the degree of the stencils and the number of parallel mappings, can be defined as the following formulas (p: number of parallel mappings):

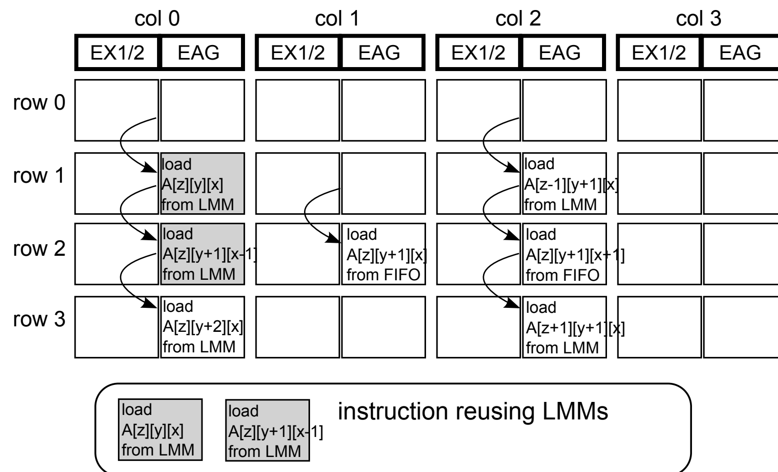
- $degree = 1 : \; 8 + 4((p/2) + (p \bmod 2) - 1) - (p \bmod 2)$
- $degree = 3 : \; 11p$

```
@0,0,1 [ 320 ] & ld ( ri+=, 8 ), r0 rgi[A._Y-1] LMM[ .A_Y-1]
@0,2,1 [ 320 ] & ld ( ri+=, 8 ), r1 rgi[A._Z-1] LMM[ .A_Z-1]
@1,0,1 [ 320 ] & ld ( ri+=, 8 ), r2 rgi[A._X-1] LMM[ .A_X-1]
@1,1,1 [ 320 ] & ld ( ri+=, 8 ), r3 rgi[A._X+0]
@1,2,1 [ 320 ] & ld ( ri+=, 8 ), r4 rgi[A._X+1]
@2,0,1 [ 320 ] & ld ( ri+=, 8 ), r5 rgi[A._Y+1] LMM[ .A_Y+1]
@2,2,1 [ 320 ] & ld ( ri+=, 8 ), r6 rgi[A._Z+1] LMM[ .A_Z+1]
```

(1) Load instructions for stencil kernel (degree=1)



(2) First processing of EMAX at Y=y-1,y,y+1.



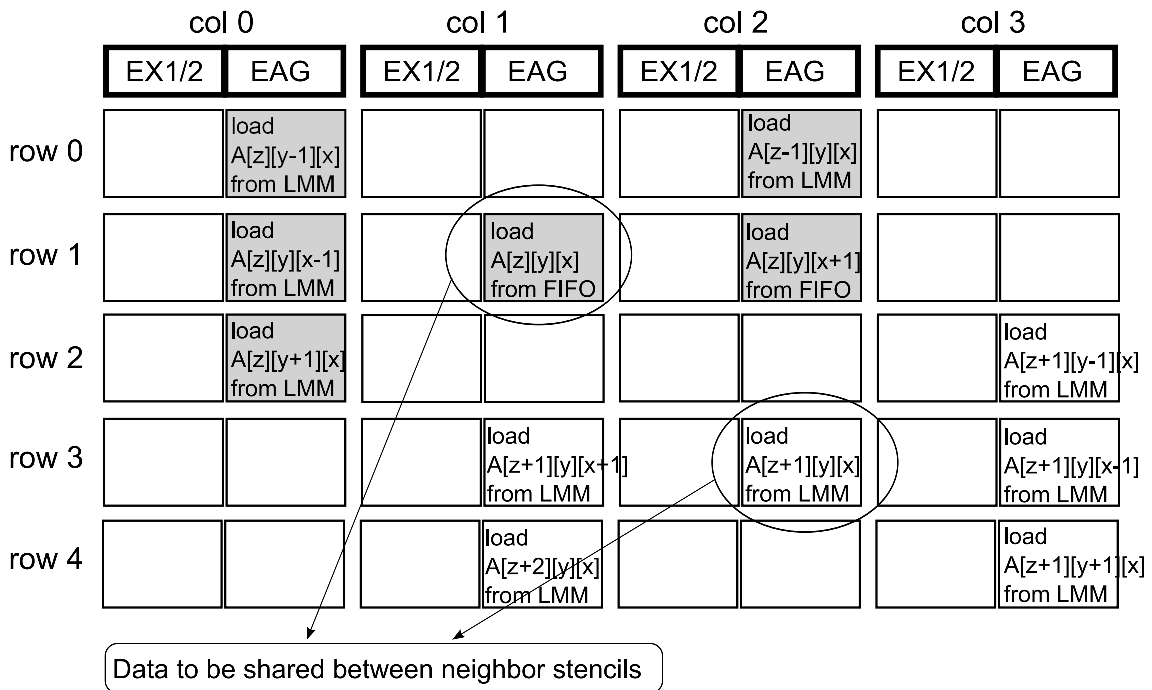
(3) Next processing of EMAX at Y=y,y+1,y+2

Figure 13. Basic instruction mapping

```
for( z=1; z< (z_size-1)/2 ; z+=2 ) {
    for( y=1; y<y_size;-1; y++ )    {
        for( x=1; x<x_size-1); x++ )    {
            B[z][y][x]      = C0 * ( A[z][y][x] ) +
                              C1 * ( A[z+1][y][x] + A[z][y+1][x] + A[z][y][x+1] +
                                     A[z-1][y][x] + A[z][y-1][x] + A[z][y][x-1] );

            B[z+1][y][x]  = C0 * ( A[z+1][y][x] ) +
                            C1 * ( A[z+2][y][x] + A[z+1][y+1][x] + A[z+1][y][x+1] +
                                   A[z][y][x] + A[z+1][y-1][x] + A[z+1][y][x-1] ); }}}
```

(1) Unrolled code in Z-direction



(2) Parallel mapping of two contiguous stencils

Figure 14. Parallel mapping

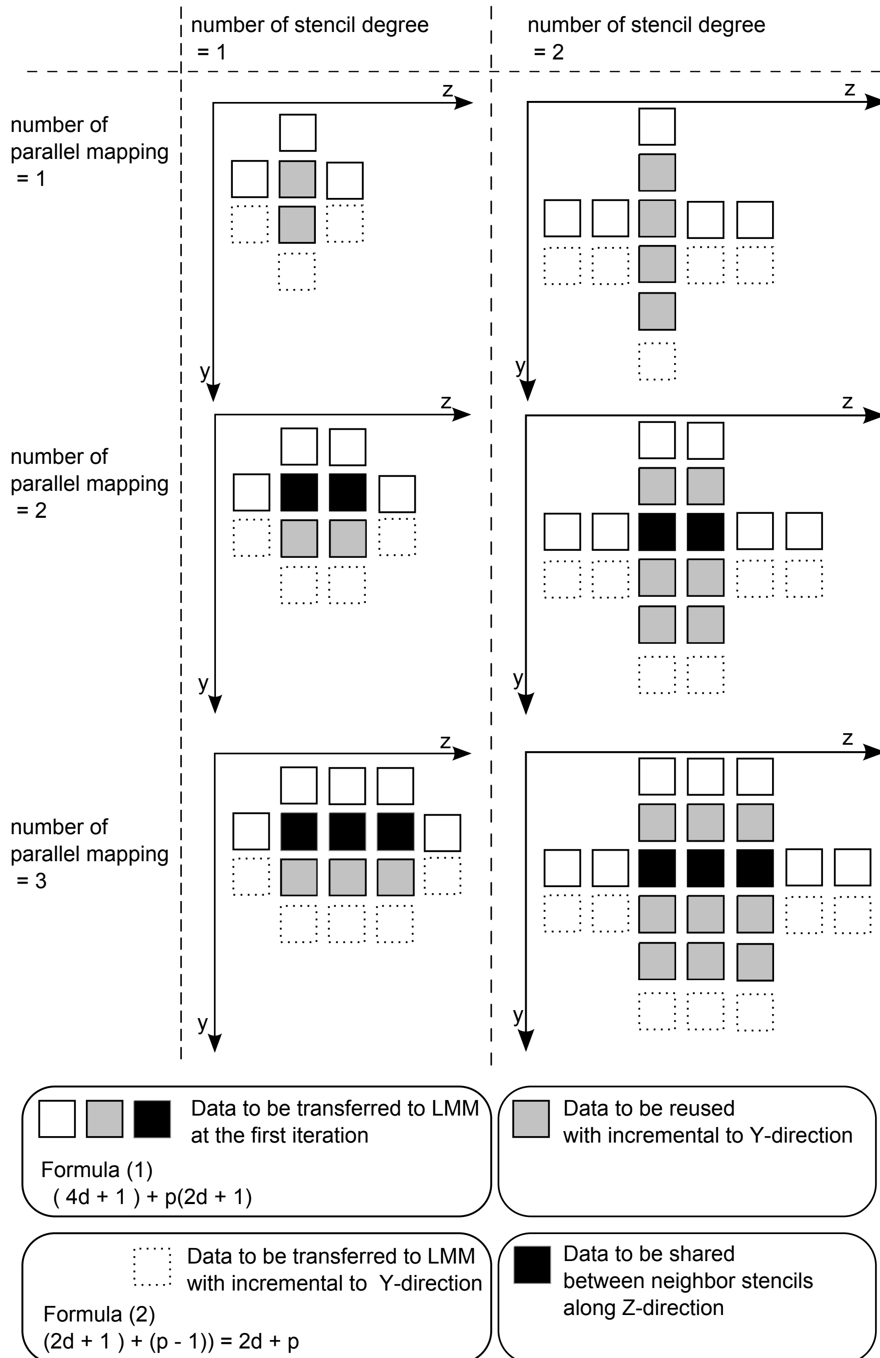


Figure 15. Data transmission to LMM on stencil computing

	col 0		col 1		col 2		col 3	
	EX1/2	EAG	EX1/2	EAG	EX1/2	EAG	EX1/2	EAG
row 0		A[z][y-1][x]				A[z-1][y][x]		
row 1	fmul	A[z][y][x-1]	fmul	A[z][y][x]		A[z][y][x+1]		
row 2	fma3	A[z][y+1][x]	fma3			A[z+1][y][x]		
row 3	fma3		fma3					
row 4	fma3							
row 5	fma3							
row 6		store						

(1) Basic mapping of stencil [degree=1]

	col 0 EX1/2	col 0 EAG	col 1 EX1/2	col 1 EAG	col 2 EX1/2	col 2 EAG	col 3 EX1/2	col 3 EAG
row 0		A[z][y-1][x]				A[z-1][y][x]		
row 1	fmul	A[z][y][x-1]	fmul	A[z][y][x]		A[z][y][x+1]		A[z+1][y-1][x]
row 2	fma3	A[z][y+1][x]	fma3	A[z+1][y][x+1]	fmul	A[z+1][y][x]	fmul	A[z+1][y][x-1]
row 3	fma3		fma3	A[z+2][y][x]	fma3		fma3	A[z+1][y+1][x]
row 4	fma3				fma3		fma3	
row 5	fma3						fma3	
row 6		store					fma3	
row 7								store

(2) Parallel mapping (p=2)]

	col 0 EX1/2	col 0 EAG	col 1 EX1/2	col 1 EAG	col 2 EX1/2	col 2 EAG	col 3 EX1/2	col 3 EAG
row 0		A[z][y-1][x]				A[z-1][y][x]		
row 1	fmul	A[z][y][x-1]	fmul	A[z][y][x]		A[z][y][x+1]		A[z+1][y-1][x]
row 2	fma3	A[z][y+1][x]	fma3	A[z+1][y][x+1]	fmul	A[z+1][y][x]	fmul	A[z+1][y][x-1]
row 3	fma3	A[z+2][y-1][x]	fma3		fma3		fma3	A[z+1][y+1][x]
row 4	fma3	A[z+2][y][x-1]	fmul	A[z+2][y][x]	fma3	A[z+2][y][x+1]	fma3	
row 5	fma3	A[z+2][y+1][x]	fma3			A[z+3][y][x]	fma3	
row 6		store	fma3				fma3	
row 7	fmul		fma3					store
row 8	fma3		fma3				fmul	
row 9	fma3						fmul	
row A		store					fmul	

(3) Parallel mapping (p=3)]

Figure 16. Instruction mapping of stencil [$degree = 1$]

6. Results and Analysis

In this section, the execution time of the 3D-Stencil Library is estimated by a clock-accurate EMAX simulator. The results are compared with general-purpose processors.

6.1 Simulation Model for Performance Measurement

For an accurate estimation of the execution time in EMAX, we developed a clock-accurate simulator that represents the activities of the memories and the registers in EMAX. The assumptions of the frequency in each component, the memory bandwidth, and the host bandwidth are shown in Table 1. After performing circuit composition by CAD with 28-nm technology using design data that became LSI by Rohm 0.18 (it operates at 52.6 MHz), we learned that EMAX operates at an internal clock speed of 252 MHz. Therefore, in the simulator condition, the EMAX frequency is assumed to be 200 MHz.

Table 1. Simulation parameters

EMAX frequency	200 MHz
HOST-DDR3 bandwidth (USB 3.0)	400 Mbyte/sec
DDR3-LocalMEM bandwidth	800 Mbyte/sec
DDR3 SDRAM capacity	256 Mbyte
Local MEM capacity	8 Kbyte

When the 3D-Stencil Library executes 3D stencil computation with a dimension size of $320 \times 320 \times 320$, the 3D-Stencil Library activates EMAX 320×320 times (when not using parallel mapping). For $degree = 1$ in the first execution, five data streams along the X axis are transmitted to DDR3 to get the results of 320-points data with the same X axis. This execution pattern resembles the processing in Fig. 5(1), and the unit of the execution corresponds to a *stream* in Fig. 17. To obtain all the stencil computing results, the 3D-Stencil Library activates EMAX in increments with the Y and Z axes, and the total activated time

becomes 320×320. Fig. 17 describes the timing chart of EMAX. Each execution is composed of five states.

- State 1

  Data transmission is executed between the main memory of the host PC and the DDR3 in EMAX.

- State 2

  Data prefetching is executed from DDR3 to LMM.

- State 3

  The instructions mapped on the PEs are executed simultaneously on the data stream.

- State 4

  The result stored in LMM is transmitted to DDR3.

- State 5

  Data transmission is executed from DDR3 to the main memory of the host PC.

- [Other conditions]

  - Each state can be started after the previous state when identical execution is completed.
  - States 1 and 5 and States 2 and 3 use the same data path. Therefore each state exclusively uses the data path.
  - Each state can be overlapped by another state.
  - Since the data size of the control information including the operation of the EMAX activation is much smaller than the 3D-array, it is ignored.

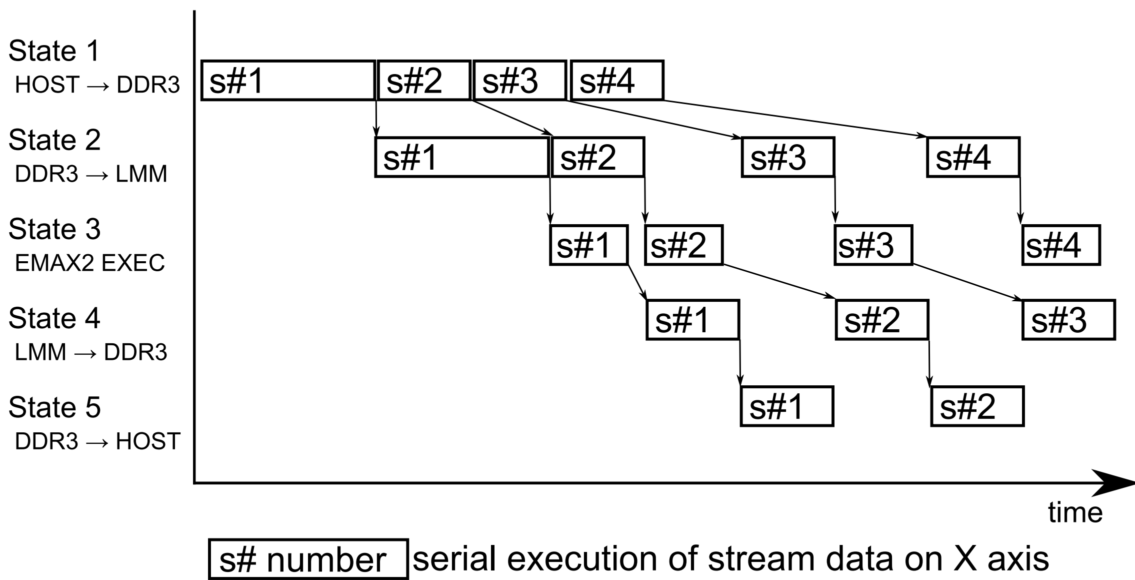


Figure 17. Execution sequence of EMAX

6.2 Evaluation of 3D-Stencil Library

We measured the execution time of the 3D-Stencil Library using the simulator. Fig. 18 presents the execution time for running stencil kernels with $degree = 1$. The results with $degree = 3$ are shown in Fig. 19. The size of the X, Y, and Z-directions in the 3D space is fixed to 320, and each point has a double precision floating point value. The "stage count" shows the number of EMAX rows. In Fig. 18, "stage count=7" corresponds to a hand-tuned code (the number of parallel mappings = 1) and the others correspond to the 3D-Stencil Library. In the same manner, in Fig. 19, "stage count=11" corresponds to a hand-tuned code, and the others correspond to the 3D-Stencil Library. The results in each case show that, as the degree of parallelization increases, the 3D-Stencil Library can reduce the execution time by more than 90% compared with a hand-tuned code. Our result shows that the evaluation in Section 5.5 is correct, and the execution time can be reduced by more than the ratio of the amount of data transmission. However, this is an over-estimation; when the size of the data increases, many conflicts occur on the communication path between DDR3 and the main memory of the host PC. Such a model is not included in the current simulator yet. However, in general, commercial accelerators have many banks to increase the memory throughput. EMAX can also increase the performance in the same manner.

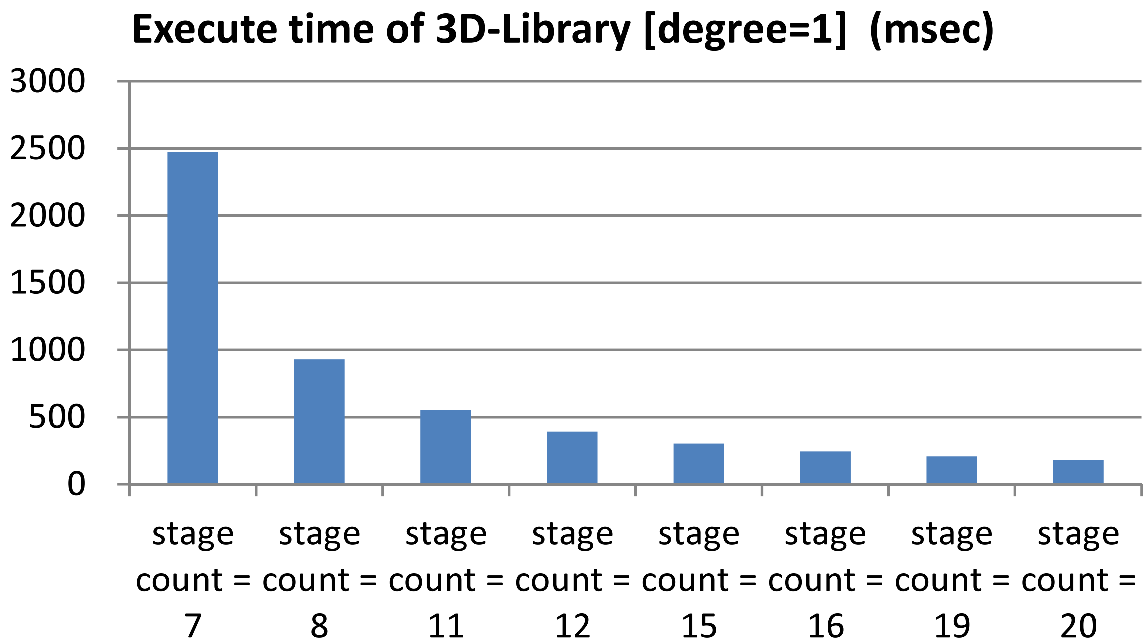


Figure 18. Execution time of 3D-Library [degree=1]

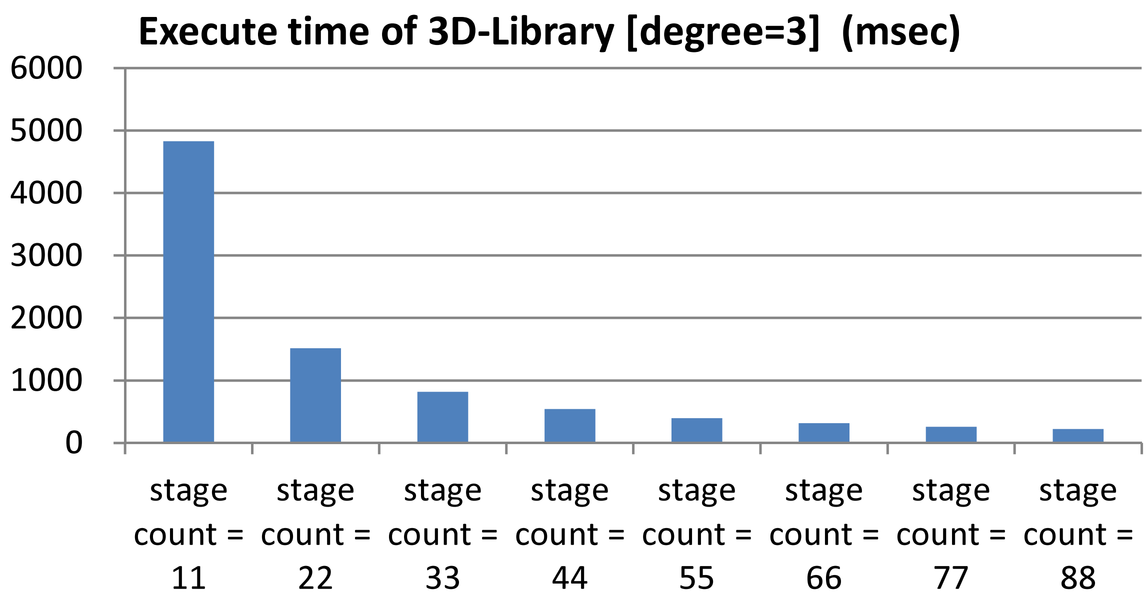


Figure 19. Execution time of 3D-Library [degree=3]

6.3 Comparison with general-purpose processors

The execution times of the stencil computation of EMAX are compared with general-purpose CPUs. The processors, the version of compilers, and the compiling options are shown in Table 2. We also use the same stencil kernels with the same size (the size of the X, Y, and Z-directions is 320), where $degree = 1$ and $degree = 3$. The results are shown in Fig. 20. In the case of $degree = 1$, Haswell is faster than the other processors including EMAX. However, when $degree = 3$, EMAX is faster. For reducing memory traffic, fitting many stencil points into FIFOs and LMMs works better than a normal shared cache system.

Table 2. General-purpose processors for comparison

CPU	Compiler	Compile option
Intel (R) Xeon (R) CPU E5405 2.00 GHz	gcc 4.1.2	-O3 -msse2 -ffast -math
Intel (R) Core (TM) i5-4670 CPU@3.40 GHz (Haswell)	gcc 4.6.3	-O3 -msse2 -ffast -math

6.4 Comparison with GPGPU

Finally, the execution times of the stencil computation of EMAX are compared with GPGPU. The specifications of GPGPU and the version of CUDA are shown in Table 3. We also use the same stencil kernels with the same size (the size of the X, Y, and Z-directions is 320), where $degree = 1$ and $degree = 3$. When we execute the stencil computation with GPGPU, it is necessary to divide the range of the calculation of each core. In the evaluation, Z-axis is divided by the number of thread block, and the space of X and Y-axis are divided by two dimension threads in the thread block as shown in Fig. 21. The measurement result of the execution time of each division case is shown in Fig. 22 and Fig. 23. Execution times were shortened most in the number of thread block 8 and the number of threads 64×16 in both measurement results.

The comparative result of the execution time of GPGPU and EMAX is

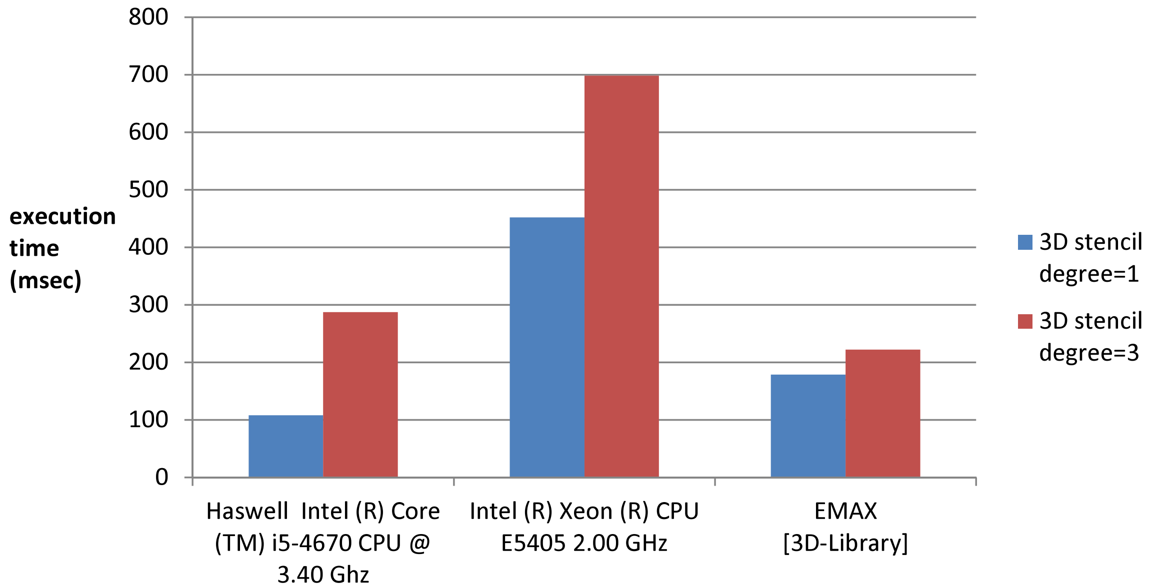


Figure 20. Comparison of execution between CPUs and EMAX

shown in Fig. 24. In both cases with degree=1 and degree=3, GPGPU is faster than EMAX. This results are the same compared with general purpose CPUs. However, when the number of cores is evaluated as the same (GPGPU:512, EMAX:352), EMAX is the twice as fast as GPGPU in the case of degree=3. It is shown that EMAX has the performance without inferiority compared with GPGPU. Moreover, there is a big difference in the data transfer performance in EMAX and GPGPU. The theoretical performance value of the EMAX simulator is 400Mbyte/sec, and measurement of the performance of GPGPU is 12Gbyte/sec. The improvement of the data transfer performance between HOST and EMAX is indispensable for the performance improvement of EMAX. The performance improvement is expected further in changing the interface between HOST and EMAX.

Table 3. Specification of GPGPU

GPU	Freq	Num of core	Interface	CUDA ver
GeForce GTX 980	1126-1216(boost)	2048	PCI-e3.0 x16	7.0

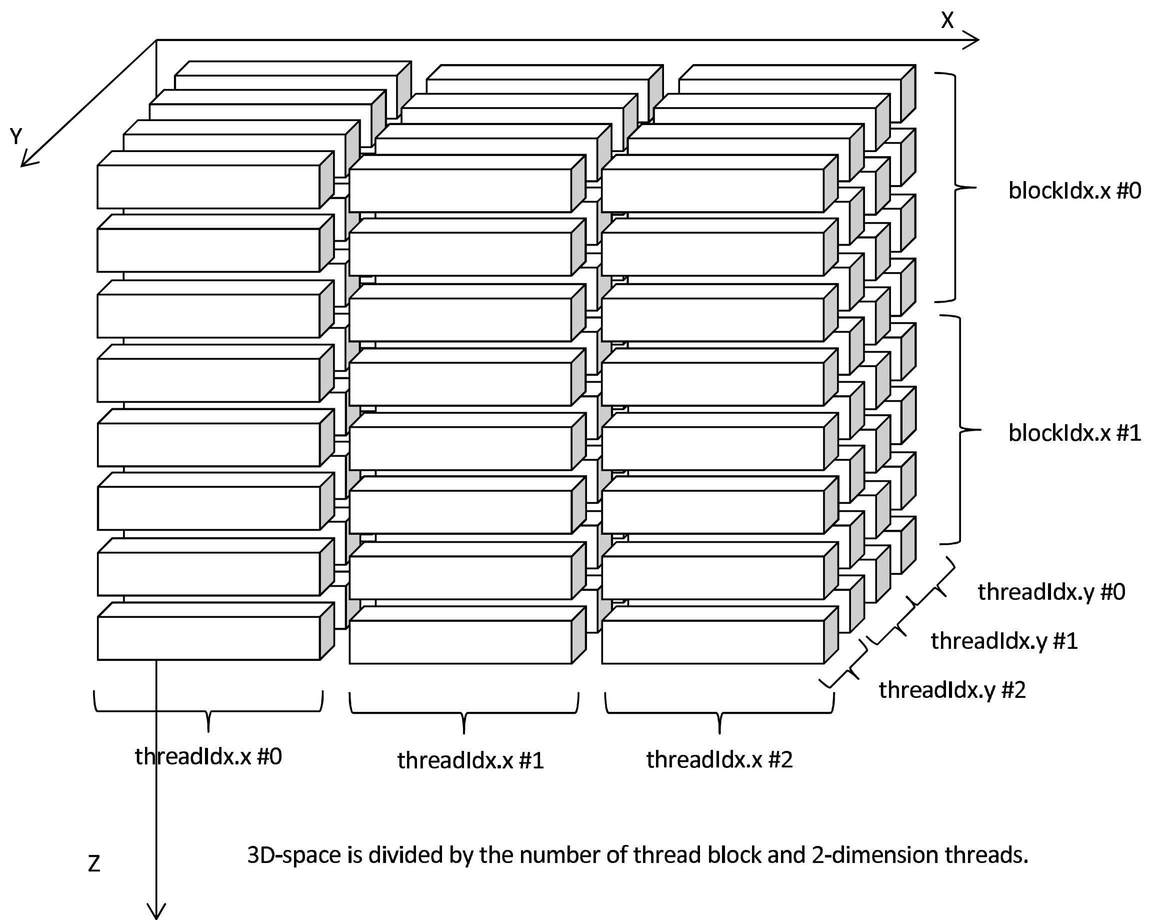


Figure 21. 3D stencil space divided according to number of threads

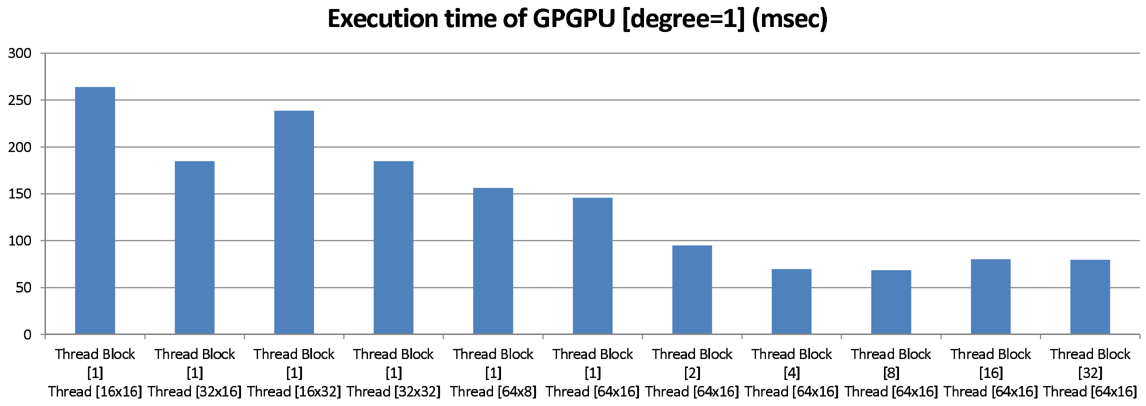


Figure 22. Execution time of GPGPU [degree=1]

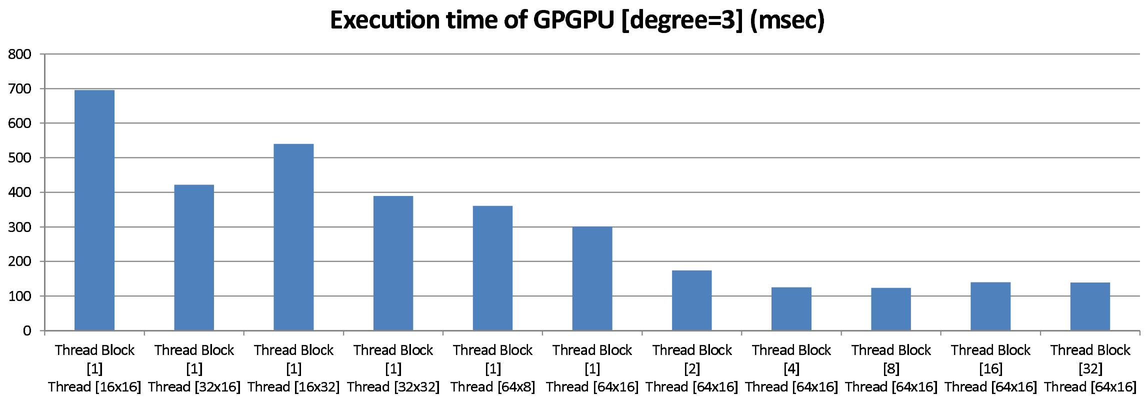


Figure 23. Execution time of GPGPU [degree=3]

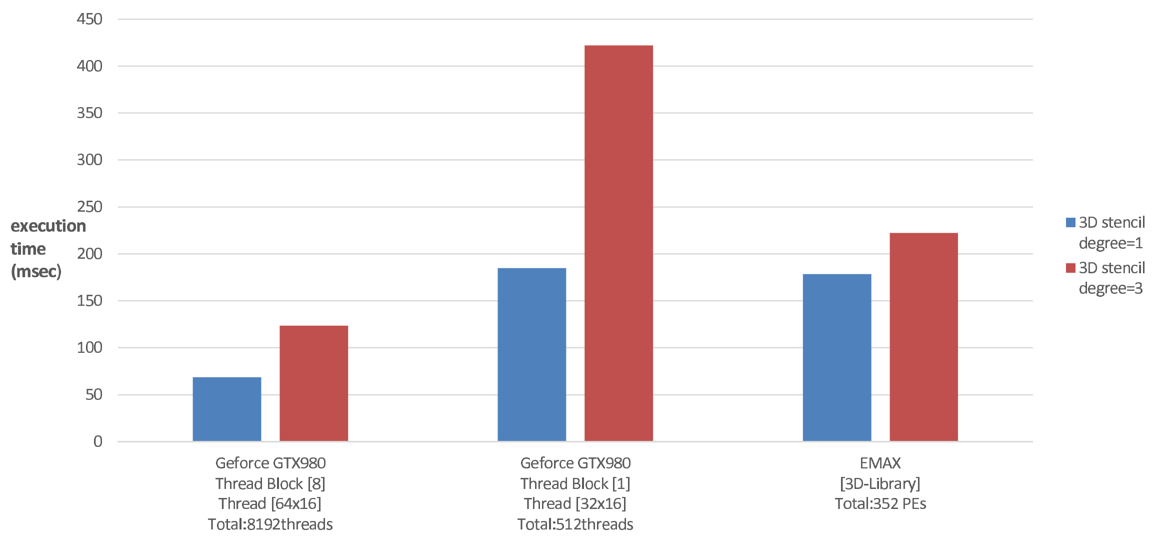


Figure 24. Comparison of execution between GPGPU and EMAX

7. Conclusion

In this dissertation, we proposed a 3D-Stencil Library that automatically generates an instruction sequence that efficiently employs EMAXs. In section 1, as a study background, we showed high performance computing architecture and the optimization schemes for stencil computing. In section 2, we showed the previous works those are either used in, or directly related to this dissertation, and LAPP we have developed for boosting performance under a given power budget. Section 3 describes the structure of stencil computing that is supported by the 3D-Stencil Library that we proposed. In section 4, we showed the features, mnemonics and configuration of EMAX that 3D-Stencil Library supports. In section 5, we showed the user interface of the 3D-Stencil Library and a technique for generating instructions for the best use of EMAX, and evaluated legitimacy of the proposed technique. In section 6, we showed simulation model of EMAX and result of performance evaluation. First, with optimization of 3D-Stencil Library, we showed execution time of 3D-Stencil kernel can be reduced more than 90% compared with a hand-tuned code. Using a 3D-Stencil Library, application developers can easily connect stencil computation on EMAX with C programs. Moreover, EMAX's execution time can be significantly reduced by eliminating the data transmission by parallelizing instruction mapping. In addition, with a performance simulator, the 3D-Stencil Library reduced the execution time 23% more than a general-purpose processor, and it was shown that EMAX and the 3D-Stencil Library have the superior performance compared with GPGPU. The practicality of EMAX and the 3D-Stencil Library is in sight. We plan to enhance the library features to cope with many-point stencils, such as 27-point stencils.

Acknowledgements

I would like to gratefully acknowledge Professor Yasuhiko Nakashima of Nara Institute of Science and Technology (NAIST) for supervising this dissertation, continuous support, suggestions and encouragement.

I wish to express my gratitude to former president and CEO Nobuyuki Kikuchi and corporate officer Takeshi Ibusuki of Fujitsu Computer Technologies Limited (FCT) who gave the opportunity to enter a school of higher grade in doctor course.

I also would like to express my appreciation to Jun Yao (HUAWEI), Assistant Professor Shinya Yamazaki-Takamaeda (NAIST) and Professor Michiko Inoue (NAIST) for their valuable opinions and support.

I also would like to thank the members of NAIST Nakashima Laboratory and the member of TMP Depts in FCT for fruitful discussions and support.

Finally, I also would like to express my special thanks to my wife Kanako Inagaki, son Towa, and daughter Yua from the bottom of my heart for continuous support and encouragement.

References

[1] Standard Performance Evaluation Corporation
http://www.spec.org/

[2] NAS Parallel Benchmarks
http://www.nas.nasa.gov/publications/npb.html

[3] Intel Instruction Set Architecture Extensions
https://software.intel.com/en-us/intel-isa-extensions

[4] NEON - ARM
http://www.arm.com/products/processors/technologies/neon.php

[5] 3D Finite Differences on Multi-core Processors
https://software.intel.com/en-us/articles/3d-finite-differences-on-multi-core-processors/

[6] High Performance Computing (HPC)-Supercomputing with NVIDIA Tesla GPU
http://www.nvidia.com/object/tesla_computing_solutions.html

[7] T. Maruyama, T. Yoshida, R. Kan, I. Yamazaki, S. Yamamura, N. Takahashi, H. Mikio, and H. Okano, "SPARC64 VIIIfx: A New-Generation Octocore Processor for Petascale Computing," IEEE Micro, Vol. 30, No. 2, pp. 30-40, 2010.

[8] X. Liao, L. Xiao, C. Yang, and Y. Lu, "MilkyWay-2 supercomputer: system and application," in Frontiers of Computer Science, vol. 8, no. 3, pp. 345-356, 2014.

[9] T. Endo and S. Matsuoka, "Massive supercomputing coping with heterogeneity of modern accelerators," in IEEE International Parallel & Distributed Processing Symposium, pp. 1-10, 2008.

[10] N. Sedaghati, R. Thomas, L. Pouchet, R. Teodorescu, and P. Sadayappan, "StVEC: A Vector Instruction Extension for High Performance Stencil Computation," in PACT, pp. 276-287, 2011.

[11] J. Cabezas, M. Araya-Plo, I. Gelado, N. Navarro, E. Morancho, and J. Cela, "High-Performance Reverse Time Migration on GPU," in International Conference of the Chilean Computer Science Society, pp. 77-86, 2009.

[12] T. Grosser, A. Cohen, S. Verdoolaege, P. Sadayappan, and J. Holewinski, "Hybrid hexagonal/classical tiling for GPUs," In: Proceedings of Annual IEEE/ACM International Symposium on Code Generation and Optimization. ACM, pp. 66, 2014.

[13] T. Henretty, R. Veras, F. Franchetti, L. Pouchet, and J. Ramanujam, "A Stencil Compiler for Short-Vector SIMD Architectures," In International Conference on Supercomputing (ICS), pp. 13-24, 2013.

[14] M. Christen, O. Schenk, and Y. Cui, "PATUS for Convenient High-Performance Stencils: Evaluation in Earthquake Simulations," In SC, pp. 11, 2012.

[15] P. Barrio, C. Carreras, R. Sierra, T. Kenter, and C. Plessl, "Turning Control Flow Graphs into Function Calls: Code Generation for Heterogeneous Architectures," In High Performance Computing and Simulation (HPCS), pp. 559-565, 2012.

[16] T. Endo and G. Jin, "Software Technologies Coping with Memory Hierarchy of GPGPU Clusters for Stencil Computations," in Cluster Computing (CLUSTER), pp. 132-139, 2014.

[17] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S. Lee, and K. Skadron, "Rodinia: A Benchmark Suite for Heterogeneous Computing," In IISWC, pp. 44-54, 2009.

[18] G. Misra, N. Kurkure, A. Das, M. Valmiki, S. Das, and A. Gupta, "Evaluation of Rodinia Codes on Intel Xeon Phi," In Intelligent Systems Modelling & Simulation (ISMS), pp. 415-419, 2013.

[19] P. Di, D. Ye, Y. Su, U. Sui, and J. Xue, "Automatic Parallelization of Tiled Loop Nests with Enhanced Fine-Grained Parallelism on GPUs," In Parallel Processing (ICPP), pp. 350-359, 2012.

[20] J. Yao, M. Saito, S. Okada, K. Kobayashi, and Y Nakashima, “EReLA: a Low-Power Reliable Coarse-Grained Reconfigurable Architecture Processor and Its Irradiation Tests,” in IEEE Nuclear and Space Radiation Effects Conference, 2014.

[21] J. Yao, Y. Nakashima, M. Saito, Y. Hazama, and R. Yamanaka, “A Flexibly Fault-Tolerant FU Array Processor and its Self-Tuning Scheme to Locate Permanently Defective Unit,” in IEEE Symposium on Low-Power and High-Speed Chips, 2014.

[22] S. Kurebayashi, J. Yao, and Y. Nakashima, “A Pipelined Newton-Raphson Method for Floating Point Division and Square Root on Distributed Memory CGRAs,” in IEEE Symposium on Low-Power and High-Speed Chips (poster), 2014.

[23] R. Shimizu, Takamaeda-Yamazaki, J. Yao, and Y. Nakashima, “High Performance Graph Processing with a Memory Intensive Array Accelerator,” in CPSY2014-11, pp. 7-12, 2014.

[24] R. Shimizu, M. M. Tanomoto, S. Takamaeda-Yamazaki, J. Yao, and Y. Nakashima, “Implementation and Evaluation of An Accelerator based on Manymemory Network,” in CPSY2014-81, pp. 51-56, 2014.

[25] M. Tanomoto, S. Takamaeda-Yamazaki, J. Yao, and Y. Nakashima, “Convolutional Neural Network Processing on An Accelerator based on Manymemory Network,” in CPSY2014-82, pp. 57-62, 2014.

[26] S. Kurebayashi, Takamaeda-Yamazaki, J. Yao, and Y. Nakashima, “Parallelization of Shortest Path Search on Various Platforms and Its Evaluation,” in CPSY2014-74, pp. 13-18, 2014.

[27] Kazuhiro YOSHIMURA, Takuya IWAKAMI, Takashi NAKADA, Jun YAO, Hajime SHIMADA and Yasuhiko NAKASHIMA, “An Instruction Mapping Scheme for FU Array Accelerator,” IEICE Trans. on Information and Systems, Vol.E94-D, No.2, pp.286-297, Feb, 2011

[28] Wei Wang, Jun Yao, Youhui Zhang, Wei Xue, Yasuhiko Nakashima, and Weimin Zheng, "HW/SW Approaches to Accelerate GRAPES in an FU Array," IEEE Symposium on Low-Power and High-Speed Chips 2013, Apr, 2013

[29] Naveen Devisetti, Takuya Iwakami, Kazuhiro Yoshimura, Takashi Nakada, Jun Yao, Yasuhiko Nakashima, "LAPP: A Low Power Array Accelerator with Binary Compatibility," HPPAC2011, pp.849-857, May. 2011

[30] N. Bell and M. Garland, "Efficient Sparse Matrix-Vector Multiplication on CUDA," NVIDIA Corporation, NVIDIA Technical Report, Dec. 2008.

[31] J. D. Davis and E. S. Chung, "SpMV: A Memory-Bound Application on the GPU Stuck Between a Rock and a Hard Place," Microsoft Research, Microsoft Technical Report, Sep. 2012.

[32] M. Gebhart, B. A. Maher, K. E. Coons, J. Diamond, P. Gratz, M. Marino, N. Ranganathan, B. Robatmili, A. Smith, J. Burrill, S. W. Keckler, D. Burger, and K. S. McKinley, "An Evaluation of the TRIPS Computer System," in Proceedings of the 14th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS XIV), Mar. 2009, pp. 112.

[33] K. Sankaralingam, R. Nagarajan, H. Liu, C. Kim, J. Huh, D. Burger, S. W. Keckler, and C. R. Moore, "Exploiting ILP, TLP, and DLP with the Polymorphous TRIPS Architecture," in Proceedings of the 30th Annual International Symposium on Computer Architecture (ISCA'03), May 2003, pp. 422433.

[34] Y. Kim, I. Park, K. Choi, and Y. Paek, "Power-Conscious Configuration Cache Structure and Code Mapping for Coarse-Grained Reconfigurable Architecture," in Proceedings of the 2006 International Symposium on Low Power Electronics and Design (ISLPED ' 06), 2006, pp.310315.

[35] Yongjun Park, Hyunchul Park, Scott Mahlke, "CGRA Express: Accelerating Execution using Dynamic Operation Fusion," CASE'09 Proceedings of the

2009 international conference on Compilers, architecture, and synthesis for embedded systems, Pages 271-280. 2009

[36] J. Lee, K. Choi, and N. Dutt. "Compilation approach for coarse-grained reconfigurable architectures, "IEEE D&T, 20:26-33, January/February. 2003

[37] B. Mei, S. Vernalde, D. Verkest, H. Man, and R. Lauwereins, "ADRES: An Architecture with Tightly Coupled VLIW Processor and Coarse-Grained Reconfigurable Matrix," in Field Programmable Logic and Application, Sep. 2003, pp. 6170.

[38] F. Bouwens, M. Berekovic, A. Kanstein, and G. Gaydadjiev, "Architectural Exploration of the ADRES Coarse-grained Reconfigurable array," Reconfigurable Computing: Architectures, Tools and Applications, vol. 44191, pp.1-13. 2007.

[39] K. Datta, M. Murphy, V. Volkov, S. Williams, J. Carter, L. Oliker, D. Patterson, J. Shalf, and K. Yelick, "Stencil Computation Optimization and Autotuning on State-of-the-art Multicore Architectures," in Proceedings of the 2008 ACM/IEEE Conference on Supercomputing (SC'08), Nov. 2008, pp. 4:14:12.

[40] L. Seiler, D. Carmean, E. Sprangle, T. Forsyth, P. Dubey, S. Junkins, A. Lake, R. Cavin, R. Espasa, E. Grochowski, T. Juan, M. Abrash, J. Sugerman, and P. Hanrahan, "Larrabee: A Many-Core x86 Architecture for Visual Computing," IEEE Micro, vol. 29, no. 1, pp. 1021, 2009.

[41] A. Heinecke, K. Vaidyanathan, M. Smelyanskiy, A. Kobotov, R. Dubtsov, G. Henry, A. Shet, G. Chrysos, and P. Dubey, "Design and Implementation of the Linpack Benchmark for Single and Multi-node Systems Based on Intel R Xeon Phi Coprocessor,“ in 2013 IEEE 27th International Symposium on Parallel Distributed Processing (IPDPS'13), May 2013, pp. 126137.

[42] J. Park, G. Bikshandi, K. Vaidyanathan, P. T. P. Tang, P. Dubey, and D. Kim, "Tera-scale 1D FFT with Low-communication Algorithm and Intel R

Xeon PhiTM Coprocessors," in Proceedings of 2013 International Conference for High Performance Computing, Networking, Storage and Analysis (SC'13), Nov. 2013, pp. 34:134:12.

[43] X.-J. Yang, X.-K. Liao, K. Lu, Q.-F. Hu, J.-Q. Song, and J.-S. Su, "The TianHe-1A Supercomputer: Its Hardware and Software," Journal of Computer Science and Technology, vol. 26, no. 3, pp. 344351, 2011.

[44] A. Heinecke, M. Klemm, and H.-J. Bungartz, "From GPGPU to ManyCore: Nvidia Fermi and Intel Many Integrated Core Architecture," Computing in Science Engineering, vol. 14, no. 2, pp. 7883, 2012.

[45] P. Micikevicius, "3D Finite Difference Computation on GPUs Using CUDA," in Proceedings of 2nd Workshop on General Purpose Processing on Graphics Processing Units (GPGPU-2), Mar. 2009, pp. 7984.

[46] Y. Zhang and F. Mueller, "Auto-generation and auto-tuning of 3d stencil codes on gpu clusters," in Proceedings of the Tenth International Symposium on Code Generation and Optimization (CGO ’12), 2012, pp. 155164.

[47] Ho, R (Dept. of Computer. Sci., Stanford Univ., CA, USA), Mai, K.W, Horowitz, M.A. "The future of wires," PROCEEDINGS OF THE IEEE, VOL89, NO.4, APRIL, 2001.

Publications

Journal Papers

1. Yoshikazu INAGAKI, Shinya, Takamaeda-Yamazaki, Jun Yao, and Yasuhiko NAKASHIMA, “Performance Evaluation of a 3D-Stencil Library for Distributed Memory Array Accelerators”, IEICE Transactions on Information and Systems, Vol.E98-D, No.12, (to appear), Dec. 2015.

Conference and Workshops (Referred)

1. Yoshikazu INAGAKI, Shinya, Takamaeda-Yamazaki, Jun Yao, and Yasuhiko NAKASHIMA, “‘Performance Evaluation of a 3D-Stencil Library for Distributed Memory Array Accelerators”, Proc. 2nd Int’l Workshop on Computer Systems and Architectures (CSA’14), held in conjunction with CANDAR’14, Shizuoka, Japan, Dec. 2014.

Conference and Workshops (Not Referred)

1. 稲垣慶和, 原祐子, 姚駿, 中島康彦, “リング型アレイアクセラレータ向け演算ライブラリの実装と性能評価”, In Proceeding of SWOPP 2013, 研究方向計算機アーキテクチャ(ARC), 2013-ARC-206, No.1, pp.1-6, Jul. 2013.