IEICE TRANS. INF. & SYST., VOL.E98–D, NO.2 FEBRUARY 2015

PAPER Special Section on Reconfigurable Systems

Performance Modeling of Stencil Computing on a Stream-Based FPGA Accelerator for Efficient Design Space Exploration*

Keisuke DOHI[†], ***Member***, **Koji OKINA**[†], **Rie SOEJIMA**[†], ***Nonmembers***, **Yuichiro SHIBATA**[†a)], ***and*** **Kiyoshi OGURI**[†], ***Members***

SUMMARY In this paper, we discuss performance modeling of 3-D stencil computing on an FPGA accelerator with a high-level synthesis environment, aiming for efficient exploration of user-space design parameters. First, we analyze resource utilization and performance to formulate these relationships as mathematical models. Then, in order to evaluate our proposed models, we implement heat conduction simulations as a benchmark application, by using MaxCompiler, which is a high-level synthesis tool for FPGAs, and MaxGenFD, which is a domain specific framework of the MaxCompiler for finite-difference equation solvers. The experimental results with various settings of architectural design parameters show the best combination of design parameters for pipeline structure can be systematically found by using our models. The effects of changing arithmetic accuracy and using data stream compression are also discussed.

key words: *high-level synthesis, FPGA, stencil computation, heat conduction simulation*

1. Introduction

As a key component of computing accelerators, Field Programmable Gate Arrays (FPGAs) are getting increasing attention especially due to their high energy efficiency [2]. Programming productivity of FPGAs is also progressively improved by emergence of various commercial and non-commercial high-level synthesis tools [3]–[7]. Stream-oriented processing, where input data items are fed into a series of arithmetic operators and processed in a deeply pipelined manner, is one of the most promising programming patterns for FPGA acceleration. A wide range of applications can be described and implemented as the stream-oriented processing. One of them is stencil computation, which is widely used for various scientific applications [8]–[11].

Using the high-level synthesis technologies, users can easily describe desired applications in a programming language without much regard for detailed hardware issues. However, architectural decisions such as a balance between depth and parallelism of pipelines still remain up to users. In order to find the best architectural parameters, time-consuming synthesis, placement and routing processes are required to be carried out over and over, severely restricting the application development productivity.

To cope with this problem, in this paper, we propose performance and resource usage models for stencil computing on a Maxeler Technology FPGA accelerator. Then, we evaluate how the proposed models contribute to finding the best design parameters and optimizing the performance, taking a 3-D heat conduction simulation as an example. The contributions of this paper include:

- Performance and resource usage modeling of 3-D stencil computing on an FPGA accelerator with high-level synthesis, which formulates how two important user-space design parameters affect the execution performance and resource amounts.
- Empirical evaluation of two types of 3-D heat conduction simulations in 32-bit floating-point and fixed-point arithmetic on a Maxeler system, which characterizes the performance and resource usage with exhaustively various design parameters.
- Evaluation and validation of our proposed models based on the empirical evaluation results, which demonstrates that our approach gives reasonable estimation of performance and resource usage, and easily determines the best combination of the design parameters.

In addition to the results of our preliminary work [1], the effects of changing arithmetic accuracy and using data stream compression are also discussed.

The remainder of this paper is organized as follows. Section 2 explains the MaxCompiler and an overview of Maxeler's FPGA accelerator. Section 3 shows MaxGenFD [12], a domain specific framework for 3-D stencil computing on MaxCompiler, and discusses user-space parameters of MaxGenFD. Section 4 proposes resource and performance models for stencil computation. Then, Sect. 5 shows implementation of heat conduction simulations with floating-point and fixed-point arithmetic as benchmark applications. Section 6 compares empirical implementation results with estimation results and discusses how the best design parameters are determined by using our proposed models. Finally, Sect. 7 summarizes the paper.

Manuscript received May 9, 2014.
Manuscript revised August 31, 2014.
Manuscript publicized November 19, 2014.

[†]The authors are with the Graduate School of Enginnering, Nagasaki University, Nagasaki-shi, 852–8521 Japan.

*A preliminary version of this paper appeared in the Proceedings of the International Conference on Reconfigurable Computing and FPGAs, ReConFig 2013, Cancun, Mexico, 9–11 December, 2013 [1].

a) E-mail: shibata@cis.nagasaki-u.ac.jp

DOI: 10.1587/transinf.2014RCP0013

2. MaxCompiler and Data Flow Engines

MaxCompiler is one of high-level synthesis tools developed by Maxeler Technologies. MaxCompiler provides a Java-based development environment with a stream-oriented programming model for high-performance computing on special FPGA accelerators. MaxCompiler generates arithmetic pipelines for FPGAs, PCIe interface, DRAM interface, FPGA-to-FPGA interface and APIs to access the accelerator from host code.

Components generated by MaxCompiler are roughly classified into (a) kernels, (b) managers, (c) streams and (d) host APIs. The kernels are main components for computation with pipelined-architecture consisting of arithmetic units and registers. Stall control mechanisms are also automatically generated by the compiler, so that users can describe desired operations in Java without being aware of latencies of arithmetic units and pipeline stalls. The managers are components which handle I/O configurations of kernels and provide communication among accelerators and the host. Attributes such as a clock frequency for kernels are also managed by the managers. The streams are routes which data items flow along. Communications among the host, kernels and DRAM are established via streams. APIs are generated by the MaxCompiler to control accelerators from applications on the host. The APIs are described in C language.

Since the MaxCompiler mainly intends to achieve high-throughput processing by making the best use of large-scale pipelines, resource reduction techniques such as resource sharing are not implicitly performed. Users have to explicitly describe the code so that resources are shared, when it is needed.

The FPGA accelerators supported by MaxCompiler are called Data Flow Engines (DFEs). The overview of MAX3424A DFE, which is a target of this work, is shown in Fig. 1. This accelerator is equipped with one Virtex-6 SX475T FPGA and six 4 GB DDRIII memories which achieve 38.4 GB/sec peak memory bandwidth in total. The FPGA and the host processor are connected via PCIe gen2 x8 interface. Kernels are mapped on the FPGA and connected with streams based on designation of the manager as shown in Fig. 1.

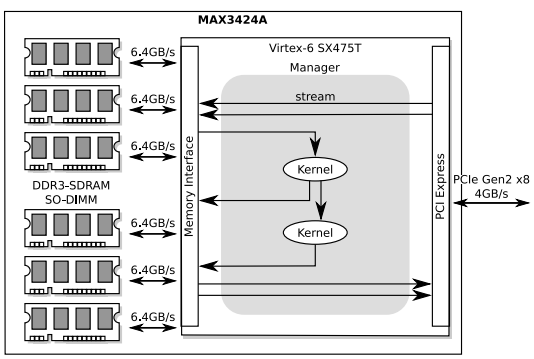


Fig. 1 Architecture overview of MAX3424A.

3. MaxGenFD

MaxGenFD is an application framework for 3-D stencil computations which is built on MaxCompiler [6]. The MaxGenFD provides various libraries and features for helping users to easily develop desired applications taking an advantage of stream-oriented processing. This section describes an architecture overview for 3-D stencil computation which is generated by MaxGenFD and introduces two important design parameters that affect the performance: *multi-pipelines* and *multi-steps*.

3.1 Overview

In MaxGenFD, users can specify the size of a computational space of the simulation target at run time. As shown in Fig. 2, the computational space is decomposed into subregions called blocks. This block decomposition [13], [14] is performed in x and y dimensions, but not in z dimension. That is, when the computational size (X, Y, Z) and the block size (B_W, B_H) are designated, the computation is performed in units of a (B_W, B_H, Z) block.

Accesses to the off-chip DDRIII memory are performed in units of a tile, which is a (T_W, T_H) 2-D array. Tiles fetched from the memory are stored in on-chip Block RAM (BRAM) and necessary data items are sent to the pipelines. Sizes of the block (B_W, B_H) and the tile (T_W, T_H) are specified by users at compile time.

3.2 Multi-Pipelines and Multi-Steps

3-D stencil computation can be accelerated (1) by parallelizing arithmetic pipelines and (2) by applying the update equations more than once per off-chip DRAM access; these methods are called multi-pipelines and multi-steps. Figure 3 shows configuration examples of kernels with the multi-pipelines and the multi-steps.

The multi-pipeline is a method to parallelize arithmetic pipelines in space so that multiple elements are computed at the same time as shown in Fig. 3 (b). This method improves

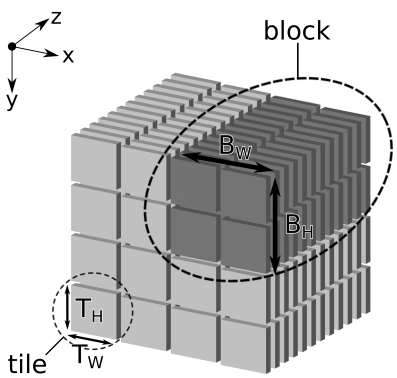


Fig. 2 Computational space decomposed into blocks and tiles.

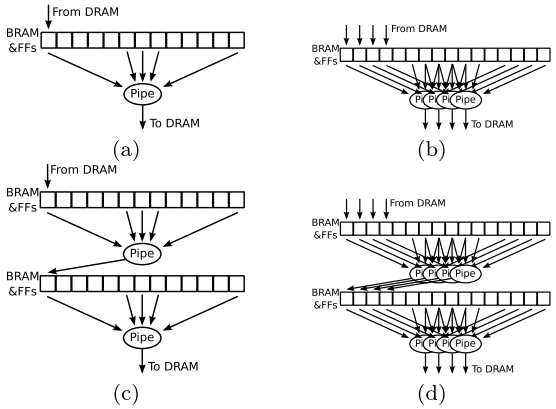


Fig. 3 Kernel configurations with multi-pipelines and multi-steps. (a) Simple kernel. (b) Kernel with 4 multi-pipelines. (c) Kernel with 2 multi-steps. (d) Kernel with 4 multi-pipelines and 2 multi-steps.

computational performance if enough memory bandwidth is available. Since the pipelines can share data caches, the increase in the number of pipelines gives a small impact on the BRAM amount devoted to caches. MaxGenFD requires the number of pipelines to be dividable by the tile width T_W.

In the multi-step method, multiple steps of arithmetic pipelines are cascaded so that the update equation is applied more than once per off-chip DRAM access as shown in Fig. 3 (c). In contrast to the multi-pipeline method, this method can increase computational performance without increasing bandwidth requirements for off-chip DRAM. The number of steps can be changed in units of a step. Thus, the granularity of optimization is finer than that of the multi-pipelines. On the other hand, this method has a large impact on BRAM usage in proportion to the number of steps.

The multi-pipelines and the multi-steps can be used at the same time. The number of pipelines N_p and the number of steps N_s can be independently assigned as shown in Fig. 3 (d). Therefore, determining the optimal combination of the parameters considering available resources on the FPGA and characteristics of the update equations is crucial for bring out the best performance of the accelerator.

Note that configurations shown in Fig. 3 illustrate logical architectures of stencil computation kernels. In the case of 3-D stencil computation, long (longer than $2B_WB_H$) logical shift registers with multiple outputs from intermediate stages are required on the FPGA. In actual design, these logical shift registers are implemented with a combination of BRAM FIFOs, LUTs and flip-flops. Although actual resource mapping is more complicated in MaxGenFD, the logical shift register model provides us a reasonable view for resource estimation.

4. Analysis and Modeling for Resource-Performance Estimation

The number of pipelines N_p and the number of steps N_s for multi-pipelines and multi-steps can be easily changed by just assigning desired numbers to variables in MaxCompiler. By assigning the best values of N_p and N_s, it becomes possible to maximize the FPGA resources that are devoted to valid computation. At the same time, these optimization techniques inevitably increase synthesized circuit size and the time required for FPGA mapping (synthesis, placing and routing). Thus, it is important to narrow down the large architectural parameter space to a realistic size by formulating resource-performance estimation models.

4.1 Resource Estimation Model

We shall first address constraints on the design parameters caused by available amount of FPGA resources. The number of DSP modules is one of the main factors that restrict the degree of the multi-pipelines and the multi-steps. The constraint on N_p and N_s imposed by the number of DSPs is expressed as:

$$N_p \times N_s \times N_{\mathrm{DSP}}^{(1,1)} \leq A_{\mathrm{DSP}} - \alpha \tag{1}$$

where $N_{\mathrm{DSP}}^{(1,1)}$ is the number of DSPs required for the computation when $(N_p, N_s) = (1, 1)$, A_{DSP} is the available number of DSPs on a target FPGA device, and α is the number of DSPs which are utilized except for the arithmetic pipelines. In addition, on the assumption that BRAM utilization per step is linearly increased with N_p, the constraint on the parameters imposed by the number of BRAMs is expressed as:

$$\left(N_{\mathrm{BRAM}}^{(1,1)} + C_p \times N_p\right) \times N_s \leq A_{\mathrm{BRAM}} - \beta \tag{2}$$

where $N_{\mathrm{BRAM}}^{(1,1)}$ is the number of BRAMs required for computation when $(N_p, N_s) = (1, 1)$, C_p is a constant factor for N_p, A_{BRAM} is the available number of BRAMs on the FPGA, and β is the number of BRAMs which are used except for the pipelines. Since FPGAs offer abundance of other resource such as LUTs and FFs compared to DSPs and BRAMs, Eqs. (1) and (2) give us available combinations of N_p and N_s in the FPGA. Note that C_p is expected to be a small value because spatially parallelized pipelines can share large part of BRAMs.

4.2 Performance Estimation Model

The peak performance of a design is limited by the computational ability or the external memory bandwidth, whichever is smaller. Given that an FPGA has infinite off-chip memory bandwidth, the ideal computational performance P_{compute} can be estimated as:

$$P_{\mathrm{compute}} = \eta N_p N_s F_s \text{ [elements/sec]} \tag{3}$$

where η is computational efficiency and F_s is the clock frequency of pipelines. Since stencil computations need values of neighboring elements to update one element, updating all the elements in a block requires values of elements located outside of the block, dubbed halo or ghost zone [15]. Due to the halo region, pipelines require additional clock cycles for

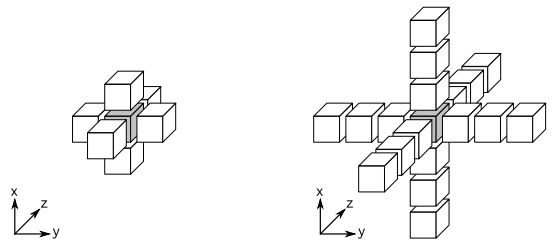


Fig. 4 Examples of stencils which are assumed in this paper. (a) Stencil of $S_d = 1$ (b) Stencil of $S_d = 3$.

loading elements from the halo region to shift registers. The size of the halo region is increased according to N_s. To formulate this effect, the computational efficiency η is defined as a ratio of written elements to read elements per block:

$$\eta = \frac{G_{\text{write}}}{G_{\text{read}}} \tag{4}$$

where G_{write} is the number of elements which are produced and written to the off-chip memory and G_{read} is the number of elements which are loaded from the off-chip memory per block computation. The number of written elements G_{write} is defined in a straightforward way, since it corresponds to the number of elements in a block:

$$G_{\text{write}} = B_W \times B_H \times Z \tag{5}$$

where B_W, B_H and Z are sizes of the block. On the other hand, definition of G_{read} is a bit more complex:

$$G_{\text{read}} = \left(B_W + 2\left\lceil \frac{S_d N_s}{T_W} \right\rceil T_W\right)\left(B_H + 2\left\lceil \frac{S_d N_s}{T_H} \right\rceil T_H\right) Z \tag{6}$$

where T_W and T_H are sizes of a tile and S_d is size of a stencil, which is a distance between an updated element and the farthermost element needed for the calculation. While the size of the stencil S_d depends on applications and the MaxGenFD supports various forms of stencils, we assume that stencils have symmetric forms as shown in Fig. 4. The ceiling functions are needed to reflect the fact that all the memory accesses are made in units of a tile [16].

4.3 Memory-Restricted Performance Model

The performance P_{mem} restricted by the external memory bandwidth is described as:

$$P_{\text{mem}} = G_{\text{write}} \frac{T_{\text{mem}}}{B_{\text{mem}}} N_s \text{ [elements/sec]} \tag{7}$$

where T_{mem} is the peak memory bandwidth of the system and B_{mem} is required amount of data transfer between the FPGA and the memory for both read and write operations. The peak memory bandwidth T_{mem} can be described as:

$$T_{\text{mem}} = 8 \times N_{\text{mem}} \times 2 \times F_{\text{mem}} \text{ [Bytes/sec]} \tag{8}$$

where N_{mem} is the number of memory channels and F_{mem} is the frequency of the memory bus clock. The amount of data transfer B_{mem} strongly depends on each application. Moreover, MaxGenFD provides a data compression mechanism for data streams, which also affects B_{mem}. Thus, we formulate B_{mem} as:

$$B_{\text{mem}} = G_{\text{write}} \sum_{i=1}^{n} \gamma_i W_i \text{ [Bytes]} \tag{9}$$

where γ_i and W_i are the data compression ratio and the data width for the i-th stream, respectively, and n is the number of data streams utilized in the application. Although it is difficult to know the exact values of γ_i, we can use the target compression ratios which are designated in user descriptions for MaxGenFD.

Finally, the peak performance P can be expressed as:

$$P = \min\left(P_{\text{compute}}, P_{\text{mem}}\right). \tag{10}$$

The design space exploration for N_p and N_s can be formulated as a maximization problem of P where constraint conditions are given by Eqs. (1) and (2).

5. Benchmark Application

We implemented heat conduction simulations with various design parameters as benchmark applications of 3-D stencil computation with the MaxCompiler. Two types of arithmetic, 32-bit single-precision floating-point and 32-bit fixed-point were utilized for comparison. The governing equation of heat conduction is expressed as:

$$\frac{\partial T(x,y,z,t)}{\partial t} = \alpha(x,y,z)\,\nabla^2 T(x,y,z,t) \tag{11}$$

where T is a temperature, t is a time variable and α is thermal diffusivity. Equation (11) can be approximated by a finite-difference equation with the explicit scheme as:

$$\begin{aligned} &T_{i,j,k}(t+\Delta t) \\ &= \alpha(x,y,z)\,\Delta t\left(\frac{T_{i+1,j,k}(t) - 2T_{i,j,k}(t) + T_{i-1,j,k}(t)}{\Delta x^2}\right. \\ &\qquad + \frac{T_{i,j+1,k}(t) - 2T_{i,j,k}(t) + T_{i,j-1,k}(t)}{\Delta y^2} \\ &\qquad \left. + \frac{T_{i,j,k+1}(t) - 2T_{i,j,k}(t) + T_{i,j,k-1}(t)}{\Delta z^2}\right) \\ &\qquad + T_{i,j,k}(t). \end{aligned} \tag{12}$$

This can be implemented on the MaxGenFD with the stencil pattern shown in Fig. 4 (a).

Figure 5 shows a part of the main computation kernel of the 3-D heat conduction simulation implemented with MaxGenFD. In this application, three streams are utilized; *in_T* for an input stream of temperature T, *alpha* for an input stream of constants α, and *out_T* for an output stream of temperature T. At line 2, the kernel obtains N_s which

```
1  /* Parameters */
2  int n_step = engine_params.getNumStep();
3  /* Stencil */
4  Stencil s = getStencil();
5  /* Input stream */
6  FDVar curr =
7   io.waveFieldInput("in_T", 1.0,
8                     n_step * (s_size/2));
9  /* Constants stream */
10 FDVar C = io.earthModelInput("alpha",10,0);
11 /* Computations */
12 FDVar s_input = curr;
13 FDVar result = constant.fdvar(0.0);
14 for ( int t= 0; t<n_step; ++t ) {
15  FDVar laplacian =
16   C * convolve(s_input, ConvolveAxes.XYZ, s);
17  result = s_input + laplacian;
18  s_input = result;
19 }
20 /* Output stream */
21 io.waveFieldOutput("out_T", result);
```

Fig. 5 An outline of the computation kernel of the benchmark.

Table 1 Common parameters of target designs.

Parameter	Value
Block size $B_W \times B_H$	(192×120)
Tile size $T_W \times T_H$	(16×12)
Pipeline frequency F_s	150 MHz
Memory bus frequency F_m	300 MHz
Stencil size S_d	1

is given as an external parameter and the multi-step behavior is explicitly described at lines 14–19. N_p is also defined outside the kernel and multiple pipelines are automatically generated by the MaxGenFD according to the value of N_p. Thus, the multi-pipeline structure does not explicitly appear in the kernel code.

6. Evaluation

To validate our proposed models, we implemented and evaluated the performance of the heat conduction simulation for $(512, 512, 512)$ of a computational space with 1,024 iterations on a MAX3424A DFE equipped in a PC with Intel Core i7-2600S 2.8 GHz, DDRIII 16 GB running Cent OS 6.4. The Virtex-6 XC6VSX475T FPGA on the MAX3424A DFE has 297,600 LUTs, 595,200 FFs, 4,788 KBytes of BRAMs and 2,016 DSPs. The evaluated designs were compiled and synthesized using MaxCompiler 2013.3, MaxGenFD 2013.3 and ISE 13.3. Place and route for each design parameter were retried up to 16 times using different cost tables until all the timing constraints were met. Table 1 summarizes common parameters for the designs.

6.1 Floating-Point Designs

Table 2 shows implementation results of floating-point designs with no stream compression. As a result of synthesis for the design of $(N_p, N_s) = (1, 1)$, $N_{\mathrm{DSP}}^{(1,1)}$ and α in Eq. (1) are estimated as 39 and 1, respectively. Then, DSP usage N_{DSP} follows $N_{\mathrm{DSP}} = 39 \times N_p \times N_s + 1$, and Eq. (1) can be transformed to $N_p \times N_s < 51$. The results in Table 2 reveal that the designs with larger N_p need fewer resources when the value of $(N_p \times N_s)$ is the same.

Figure 6 shows limitations on the design parameters

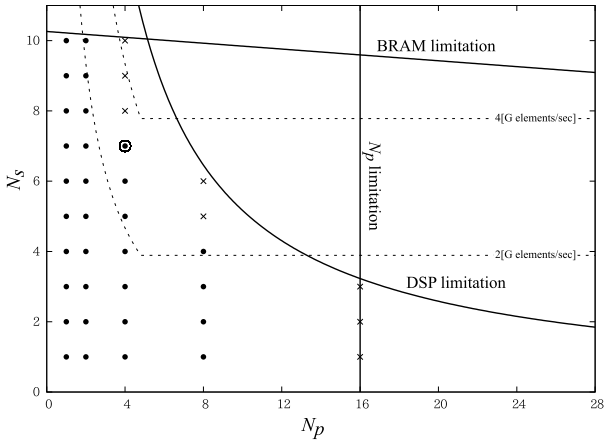


Fig. 6 Estimated available parameters and synthesis results for floating-point designs.

estimated by our proposed model and synthesis results. The three solid lines in Fig. 6 illustrate limitations imposed by the number of DSPs, the number of BRAMs, and the maximum N_p value supported in MaxGenFD, respectively. The contour map of the estimated performance is also demonstrated with dashed lines. Our model estimates that the threshold value of N_p for $P_{\mathrm{mem}} < P_{\mathrm{compute}}$ is 4.8. Here, a dot point indicates that the corresponding design parameter combination was estimated as available and successfully implemented. A cross point indicates that the parameter combination failed to be implemented. The circled dot shows the best parameters that were successfully implemented: $(N_p = 4, N_s = 7)$. Since a few promising parameter combination candidates failed to be implemented due to placing and routing, the best available design was the 5th best one in the estimation model. However, among the actually implemented design parameters, the best parameters were accurately anticipated as the best. While our resource estimation model seems to be slightly optimistic, the performance model gives us a good direction to easily choose parameters without exhaustive parameter space exploration.

Next, we measured the actual execution performances and compared them to the estimated performances as shown in Fig. 7. The plotted marks indicate measured values and dashed lines indicate estimated values by Eq. (10). Table 3 shows estimation errors in terms of the mean absolute percentage error (MAPE). All the configurations in this work have the same computational efficiency $\eta = 0.714$. The evaluation results in Fig. 7 show the performance was well estimated by our model. In total, the MAPE of the estimation was 10.2%.

6.2 Fixed-Point Designs

For comparison, we also evaluated fixed-point designs of the heat conduction simulations with the same condition. Tables 4 and 5 show implementation results. Here, as a result of synthesis for the design of $(N_p, N_s) = (1, 1)$, $N_{\mathrm{DSP}}^{(1,1)}$ and α in Eq. (1) were estimated as 10 and 1, respectively. Chang-

Table 2 Resource utilization and performance for floating-point designs.

Case	N_p	N_s	$N_p \times N_s$	LUTs	FFs	BRAMs	DSPs	Performance [elements/sec]
#1	1	1	1	45,911	65,678	165	40	1.29e+08
#2	1	2	2	50,154	71,186	239	79	2.57e+08
#3	1	3	3	56,729	78,021	334	118	3.86e+08
#4	1	4	4	61,893	84,647	430	157	5.13e+08
#5	1	5	5	67,022	92,033	525	196	6.44e+08
#6	1	6	6	74,808	99,259	621	235	7.75e+08
#7	1	7	7	79,485	105,596	715	274	8.96e+08
#8	1	8	8	85,839	112,111	812	313	1.03e+09
#9	1	9	9	91,444	118,814	907	352	1.16e+09
#10	1	10	10	96,884	125,676	1,003	391	1.28e+09
#11	2	1	2	48,108	69,660	167	79	2.36e+08
#12	2	2	4	56,323	80,038	244	157	4.75e+08
#13	2	3	6	65,540	90,739	340	235	7.03e+08
#14	2	4	8	72,777	100,548	437	313	9.52e+08
#15	2	5	10	81,565	110,304	534	391	1.19e+09
#16	2	6	12	88,773	120,623	631	469	1.41e+09
#17	2	7	14	96,495	130,956	727	547	1.63e+09
#18	2	8	16	105,868	141,143	826	625	1.89e+09
#19	2	9	18	113,188	150,982	924	703	2.13e+09
#20	2	10	20	120,402	161,614	1,021	781	2.40e+09
#21	4	1	4	56,080	79,205	174	157	4.03e+08
#22	4	2	8	70,237	97,614	258	313	8.12e+08
#23	4	3	12	85,344	115,945	360	469	1.21e+09
#24	4	4	16	99,161	134,233	463	625	1.63e+09
#25	4	5	20	113,478	151,871	565	781	2.03e+09
#26	4	6	24	128,643	171,315	669	937	2.44e+09
#27	4	7	28	143,519	189,526	771	1,093	2.86e+09
#28	4	8	32	Failed to place and route				
#29	4	9	36	Failed to place and route				
#30	4	10	40	Failed to place and route				
#31	8	1	8	73,311	99,897	182	313	4.88e+08
#32	8	2	16	102,129	134,467	262	625	9.99e+08
#33	8	3	24	127,970	169,229	360	937	1.49e+09
#34	8	4	32	155,346	204,150	462	1,249	1.98e+09
#35	8	5	40	Failed to place and route				
#36	8	6	48	Failed to place and route				
#37	16	1	16	Failed to place and route				
#38	16	2	32	Failed to place and route				
#39	16	3	48	Failed to place and route				

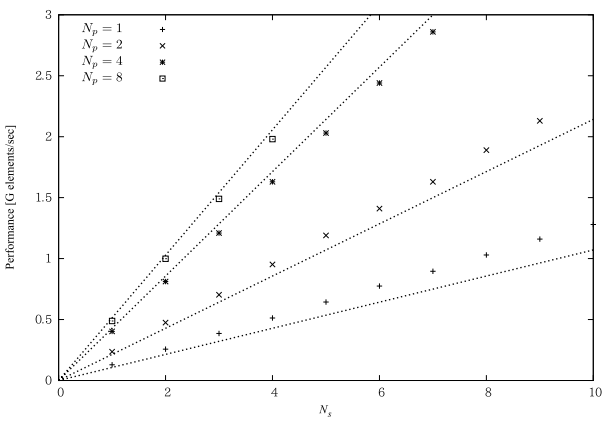


Fig. 7 Estimated and actual performance for floating-point designs.

ing arithmetic from floating-point to fixed-point, resource utilization of DSPs was reduced to about one-quarter. Thus, DSP usage N_{DSP} follows $N_{\mathrm{DSP}} = 10 \times N_p \times N_s + 1$, and Eq. (1)

Table 3 Estimation errors for floating-point designs.

N_p	MAPE [%]
1	16.7
2	9.3
4	5.6
8	3.9
Total	10.2

can be transformed to $N_p \times N_s < 202$, making a much larger design space.

The effect of the alleviation of the resource limitation is clearly shown in Fig. 8; the number of available combinations of N_p and N_s was increased. On the other hand, the performance saturation point in N_p due to the external memory bandwidth remained to be the same, since the width of data streams is 32 bits for both floating-point and fixed-point designs. Thanks to the increase in the number of design alternatives, the performance achieved by the best fixed-point implementation surpassed that of the best floating-point im-

Table 4 Resource utilization and performance for fixed-point designs.

Case	N_p	N_s	$N_p \times N_s$	LUTs	FFs	BRAMs	DSPs	Performance [elements/sec]
#1	1	1	1	40,234	59,979	160	11	1.30e+08
#2	1	2	2	43,172	62,585	203	21	2.59e+08
#3	1	3	3	45,202	65,501	270	31	3.90e+08
#4	1	4	4	44,891	67,301	330	41	5.16e+08
#5	1	5	5	48,204	69,564	391	51	6.49e+08
#6	1	6	6	49,555	71,552	458	61	7.78e+08
#7	1	7	7	51,775	72,637	518	71	9.08e+08
#8	1	8	8	52,411	74,234	572	81	1.03e+09
#9	1	9	9	54,167	76,450	637	91	1.16e+09
#10	1	10	10	56,742	78,860	704	101	1.29e+09
#11	1	11	11	57,836	80,214	765	111	1.42e+09
#12	1	12	12	59,696	82,394	832	121	1.55e+09
#13	1	13	13	61,606	83,013	892	131	1.68e+09
#14	1	14	14	62,151	85,496	951	141	1.81e+09
#15	1	15	15	65,035	86,649	1,018	151	1.93e+09
#16	1	16	16	Over BRAMs utilization (1,078 > 1,064)				
#17	2	1	2	42,262	61,765	162	21	2.39e+08
#18	2	2	4	43,754	64,480	207	41	4.80e+08
#19	2	3	6	44,518	67,177	272	61	7.12e+08
#20	2	4	8	46,914	69,215	330	81	9.61e+08
#21	2	5	10	48,246	71,282	394	101	1.20e+09
#22	2	6	12	50,450	73,861	459	121	1.42e+09
#23	2	7	14	52,450	76,274	520	141	1.67e+09
#24	2	8	16	54,292	78,545	583	161	1.92e+09
#25	2	9	18	55,517	80,731	647	181	2.15e+09
#26	2	10	20	58,008	83,042	707	201	2.38e+09
#27	2	11	22	59,671	85,211	771	221	2.64e+09
#28	2	12	24	60,096	87,518	834	241	2.87e+09
#29	2	13	26	63,160	89,661	895	261	3.09e+09
#30	2	14	28	64,250	92,172	959	281	3.31e+09
#31	2	15	30	64,481	93,682	1,022	301	3.59e+09
#32	2	16	32	Over BRAMs utilization (1,083 > 1,064)				
#33	4	1	4	44,439	64,726	164	41	4.18e+08
#34	4	2	8	45,514	67,835	214	81	8.32e+08
#35	4	3	12	48,699	71,231	278	121	1.22e+09
#36	4	4	16	51,506	74,463	341	161	1.64e+09
#37	4	5	20	53,069	77,506	404	201	2.05e+09
#38	4	6	24	56,718	80,923	469	241	2.50e+09
#39	4	7	28	56,747	84,295	532	281	2.89e+09
#40	4	8	32	60,938	87,429	596	321	3.33e+09
#41	4	9	36	60,378	90,216	669	361	3.66e+09
#42	4	10	40	66,382	93,394	735	401	4.06e+09
#43	4	11	44	67,892	97,363	787	441	4.54e+09
#44	4	12	48	69,895	99,816	864	481	4.99e+09
#45	4	13	52	71,022	104,284	915	521	5.37e+09
#46	4	14	56	73,684	106,682	995	561	5.57e+09
#47	4	15	60	75,205	110,639	1,041	601	6.08e+09
#48	4	16	64	Over BRAMs utilization (1,106 > 1,064)				

plementation. In this case, our model was able to correctly estimate the best available design parameters.

Figure 9 shows the measured and estimated performances for the fixed-point designs and Table 6 summarizes the estimation errors. As is the case with floating-point designs, the performance estimation results agree well with the measured ones, including the performance saturation due to the memory bandwidth that is shown when N_p is 8 and 16. The total MAPE of the estimation was 7.2%.

6.3 Effect of Data Compression

Finally, we evaluated the effect of the stream compression that MaxGenFD offers. We implemented the heat conduct simulations with floating-point arithmetic enabling the stream compression. The target compression ratios were designated as 0.5 for two streams and 0.3125 for the other stream. The designs were compiled using MaxCompiler 2012.1, MaxGenFD 2012.1 and ISE 13.3 with 303 MHz of the memory bus frequency.

Table 7 summarizes the implementation results. Compared to the results in Table 2, the DSP utilization was the same. Also for LUTs, FFs and BRAMs, the utilization was the almost same, suggesting the data compression mechanism of Maxeler has a limited impact on hardware amount.

Figure 10 shows the estimated available design param-

Table 5 Resource utilization and performance for fixed-point designs (continued).

Case	N_p	N_s	$N_p \times N_s$	LUTs	FFs	BRAMs	DSPs	Performance [elements/sec]
#49	8	1	8	50,373	72,957	174	81	5.01e+08
#50	8	2	16	53,330	77,383	220	161	1.00e+09
#51	8	3	24	56,829	82,805	280	241	1.50e+09
#52	8	4	32	59,515	88,145	341	321	2.04e+09
#53	8	5	40	65,388	93,273	399	401	2.49e+09
#54	8	6	48	69,450	99,021	459	481	3.06e+09
#55	8	7	56	70,687	104,444	520	561	3.56e+09
#56	8	8	64	75,110	109,150	578	641	4.06e+09
#57	8	9	72	80,027	115,009	637	721	4.54e+09
#58	8	10	80	81,410	120,397	698	801	5.11e+09
#59	8	11	88	87,223	125,949	757	881	5.63e+09
#60	8	12	96	90,426	131,310	817	961	6.14e+09
#61	8	13	104	94,736	136,564	878	1,041	6.62e+09
#62	8	14	112	99,079	141,886	937	1,121	7.10e+09
#63	8	15	120	101,217	147,359	997	1,201	7.55e+09
#64	8	16	128	105,528	152,450	1,056	1,281	8.18e+09
#65	16	1	16	60,321	86,136	205	161	5.00e+08
#66	16	2	32	66,651	95,125	254	321	1.00e+09
#67	16	3	48	71,962	104,511	315	481	1.53e+09
#68	16	4	64	79,502	113,996	373	641	2.03e+09
#69	16	5	80	85,061	123,278	432	801	2.52e+09
#70	16	6	96	90,142	132,456	495	961	3.04e+09
#71	16	7	112	96,779	142,159	556	1,121	3.56e+09
#72	16	8	128	103,548	151,347	613	1,281	3.97e+09
#73	16	9	144	108,962	160,673	669	1,441	4.60e+09
#74	16	10	160	117,248	170,224	729	1,601	5.11e+09
#75	16	11	176	124,660	180,139	792	1,761	5.43e+09
#76	16	12	192	129,748	189,588	858	1,921	6.11e+09

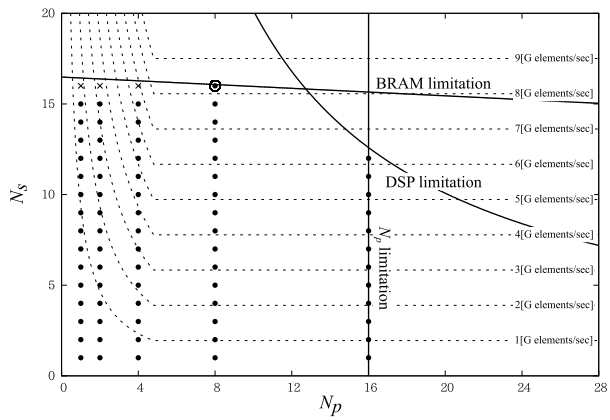


Fig. 8 Estimated available parameters and synthesis results for fixed-point designs.

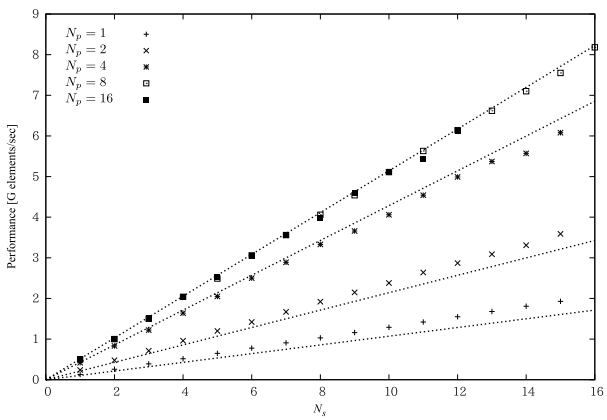


Fig. 9 Estimated and actual performance for fixed-point designs.

eters and their synthesis results. Although the selectable parameters are the same with the case in Fig. 6, B_{mem} in Eq. (7) depends on the compression ratios and thus the estimated performance lines were changed. The threshold value of N_p which makes the performance saturation due to the memory bandwidth was also increased to 44.9. The best performance in the floating-point design with compression was achieved when $(N_p, N_s) = (8, 5)$ and was about 1.4 times faster than the best floating-point design without compression. The number of more promising designs, which were estimated as available but actually failed, was four. This indicates again that our resource estimation models have room to be improved to take into account place and routing issues.

Table 6 Estimation errors for fixed-point designs.

N_p	MAPE [%]
1	17.2
2	10.3
4	4.3
8	1.5
16	1.8
Total	7.2

A comparison between Table 2 and Table 7 reveals that data compression improved the performance only when $N_p \geq 4$, and performance slowdowns were observed for the other cases. One of the negative side effects of the data com-

Table 7 Resource utilization and performance for floating-point designs with data compression.

Case	N_p	N_s	$N_p \times N_s$	LUTs	FFs	BRAMs	DSPs	Performance [elements/sec]
#1	1	1	1	49,105	70,074	175	40	1.07e+08
#2	1	2	2	54,490	77,285	248	79	2.13e+08
#3	1	3	3	58,349	83,804	345	118	3.18e+08
#4	1	4	4	65,907	90,434	440	157	4.23e+08
#5	1	5	5	70,782	96,401	535	196	5.27e+08
#6	1	6	6	75,727	103,328	631	235	6.30e+08
#7	1	7	7	82,076	110,125	727	274	8.38e+08
#8	1	8	8	88,695	117,145	822	313	8.39e+08
#9	1	9	9	93,314	123,686	917	352	9.39e+08
#10	1	10	10	99,420	129,822	1,011	391	1.04e+09
#11	2	1	2	52,603	75,163	178	79	2.12e+08
#12	2	2	4	59,690	85,445	254	157	4.23e+08
#13	2	3	6	67,938	95,837	350	235	6.32e+08
#14	2	4	8	78,077	107,087	447	313	8.41e+08
#15	2	5	10	84,601	115,832	546	391	1.05e+09
#16	2	6	12	93,069	128,687	641	469	1.25e+09
#17	2	7	14	100,655	136,433	739	547	1.45e+09
#18	2	8	16	109,664	150,707	834	625	1.67e+09
#19	2	9	18	117,755	157,225	931	703	1.87e+09
#20	2	10	20	126,181	167,481	1,031	781	2.06e+09
#21	4	1	4	58,949	84,050	183	157	4.21e+08
#22	4	2	8	74,231	102,657	267	313	8.38e+08
#23	4	3	12	89,764	121,454	369	469	1.25e+09
#24	4	4	16	104,261	139,497	471	625	1.66e+09
#25	4	5	20	119,424	157,814	573	781	2.08e+09
#26	4	6	24	131,246	176,207	677	937	2.48e+09
#27	4	7	28	147,648	194,462	780	1,093	2.88e+09
#28	4	8	32	185,949	595,200	884	1249	3.30e+09
#29	4	9	36	Failed to place and route				
#30	4	10	40	Over BRAMs utilization (1,092 > 1,064)				
#31	8	1	8	75,543	105,508	201	313	8.30e+08
#32	8	2	16	104,747	140,537	281	625	1.65e+09
#33	8	3	24	132,534	175,847	381	937	2.45e+09
#34	8	4	32	157,202	211,046	481	1,249	3.26e+09
#35	8	5	40	188,686	247,667	577	1,561	4.06e+09
#36	8	6	48	Failed to place and route				
#37	16	1	16	111,929	150,289	238	625	1.60e+09
#38	16	2	32	165,106	218,620	326	1,249	3.18e+09
#39	16	3	48	Failed to place and route				

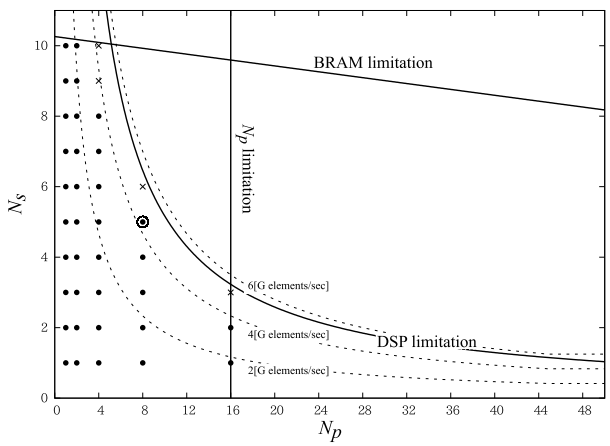


Fig. 10 Estimated available parameters and synthesis results for floating-point designs with data compression.

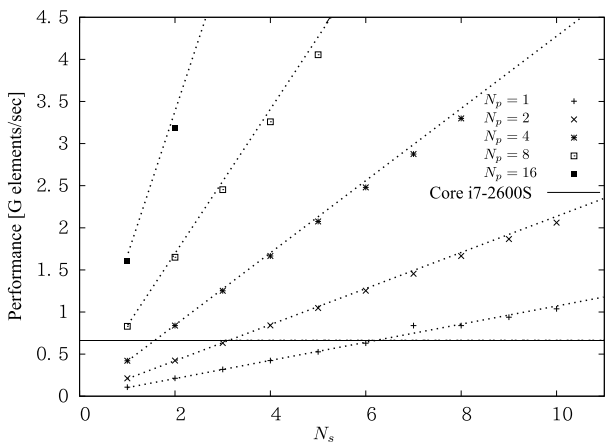


Fig. 11 Estimated and actual performance for floating-point designs with data compression.

Table 8 Estimation errors for floating-point designs with data compression.

N_p	MAPE [%]
1	2.5
2	2.4
4	3.1
8	4.5
16	7.4
Total	3.2

pression is to disable an in-block memory access optimization, which is automatically performed by MaxGenFD. For the designs with small N_p, originally the memory bandwidth was not a severe bottleneck. Therefore, the data compression did not improve the performance and the side effect was actualized. On the other hand, for the designs with large N_p, the data compression effectively alleviated the memory bottleneck, overcoming the side effect.

Figure 11 shows performance evaluation results for the floating-point designs with data compression and their estimation errors are shown in Table 8. The performance was well estimated also in this case, showing a total MAPE of 3.2%. In addition, we compared the DFE designs with multi-threaded CPU implementation with SIMD instructions on the host PC which was used for above evaluations. The CPU implementation achieved the performance of 6.66×10^8 elements/sec, and it is equivalent to the DFE designs which have $(N_p \times N_s) = 6$ in 32-bit single-precision floating-point. The best configuration $(N_p, N_s) = (8, 5)$ in floating-point with data compression was about six times faster than the CPU implementation. As shown in Fig. 11, some of the DFE designs were actually slower than the CPU implementation. This suggests the importance of appropriate selection of design parameters for application acceleration.

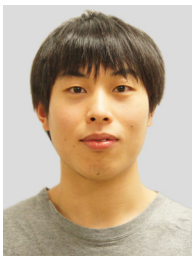
7. Conclusion

In this paper, we have presented the resource and performance estimation models for 3-D stencil computation on the Maxeler Data Flow Engines for efficient design space exploration. We have used heat conduction simulations with 32-bit single-precision floating-point and 32-bit fixed-point arithmetic as benchmark applications and evaluated resource utilization and performance for various parameter configurations. The experiments results have shown that while our resource estimation models were sometimes optimistic due to place and routing issues were not taken in account, the performance estimation model agreed well with measure performance. It has been also shown that the proposed models systematically found the best combination of design parameters of pipeline structure and easily estimated the effect of fixed-point arithmetic and data stream compression. Our future work includes improvement of resource estimation models and introducing of energy consumption models to optimize energy-performance efficiency.

References

[1] K. Dohi, K. Fukumoto, Y. Shibata, and K. Oguri, "Performance modeling and optimization of 3-D stencil computation on a stream-based FPGA accelerator," ReConFig, pp.1–6, 2013.
[2] High-Performance Computing Using FPGAs, ed. W. Vanderbauwhede and K. Benkrid, Springer, New York, 2013.
[3] Impulse Accelerated Technologies, "Impulse C," http://www.impulseaccelerated.com/
[4] Xilinx, "Vivado HSL Design," http://www.xilinx.com/products/design-tools/vivado/integration/esl-design/index.htm
[5] Accellera Systems Inititaive, "System C," http://www.accellera.org/home
[6] Maxeler Technologies, "MaxCompiler," http://www.maxeler.com/
[7] O. Pell, O. Mencer, K. Tsoi, and W. Luk, "Maximum performance computing with dataflow engines," in High-Performance Computing Using FPGAs, ed. W. Vanderbauwhede and K. Benkrid, pp.747–774, Springer, New York, 2013.
[8] Y. Sato, Y. Inoguchi, W. Luk, and T. Nakamura, "Evaluating reconfigurable dataflow computing using the Himeno benchmark," Proc. International Conference on ReConFigurable Computing and FPGAs, pp.1–7, 2012.
[9] H. Giefers, C. Plessl, and J. Förstner, "Accelerating finite difference time domain simulations with reconfigurable dataflow computers," Proc. 4th International Symposium on Highly-Efficient Accelerators and Reconfigurable Technologies, pp.33–38, 2013.
[10] K. Sano, "FPGA-based systolic computational-memory array for scalable stencil computations," in High-Performance Computing Using FPGAs, ed. W. Vanderbauwhede and K. Benkrid, pp.279–303, Springer, New York, 2013.
[11] T. Kobori and T. Maruyama, "A high speed computation system for 3D FCHC lattice gas model with FPGA," in Field Programmable Logic and Application, ed. P. Cheung and G. Constantinides, Lecture Notes in Computer Science, vol.2778, pp.755–765, Springer, Berlin Heidelberg, 2003.
[12] Maxeler Technologies, "MaxGenFD Tutorial Version 2013.3," Dec. 2013.
[13] K. Datta, M. Murphy, V. Volkov, S. Williams, J. Carter, L. Oliker, D. Patterson, J. Shalf, and K. Yelick, "Stencil computation optimization and auto-tuning on state-of-the-art multicore architectures," Proc. 2008 ACM/IEEE Conference on Supercomputing, SC '08, pp.4:1–4:12, Piscataway, NJ, USA, 2008.
[14] A.W. Lim, S.W. Liao, and M.S. Lam, "Blocking and array contraction across arbitrarily nested loops using affine partitioning," SIGPLAN Not., vol.36, no.7, pp.103–112, June 2001.
[15] M. Jiayuan and S. Kevin, "A performance study for iterative stencil loops on GPUs with ghost zone optimizations," International Journal of Parallel Programming, vol.39, no.1, pp.115–142, 2010.
[16] G. Rivera and C.W. Tseng, "Tiling optimizations for 3D scientific computations," Proc. 2000 ACM/IEEE Conference on Supercomputing, SC '00, pp.1–23, Washington, DC, USA, 2000.

Keisuke Dohi received B.E., M.E. and Ph.D. degrees from Nagasaki University, Japan, in 2009, 2011 and 2014, respectively. His research interests include reconfigurable systems and GPGPU computing.



Koji Okina received B.E degree from Nagasaki University, Japan, in 2014. His research interests include reconfigurable systems.

Rie Soejima received B.E degree from Nagasaki University, Japan, in 2014. Her research interests include reconfigurable systems.

Yuichiro Shibata received the B.E. degree in electrical engineering, the M.E. and Ph.D. degrees in computer science from Keio University, Japan, in 1996, 1998 and 2001, respectively. Currently, he is an associate professor at Department of Computer and Information Sciences, Nagasaki University. He was a Visiting Scholar at University of South Carolina in 2006. His research interests include reconfigurable systems and parallel processing. He won the Best Paper Award of IEICE in 2004.

Kiyoshi Oguri received B.S. and M.S. degrees in physics from Kyushu University, Japan, in 1974 and 1976, respectively. He also received the Ph.D. degree in information engineering from the same university in 1997. Since joining NTT in 1976, he have been engaged in the research, design and development of high-end general purpose computer, high-level logic synthesis system and a wired logic-based dynamic computing architecture. Currently, he is a professor of Nagasaki University, Japan. His research interests are in hardware modeling, high-level synthesis, FPGA-related systems, and Plastic Cell Architecture. Prof. Oguri received the Motooka Prize in 1987, the Best Paper Award of IPSJ in 1990, the Okochi Memorial Technology Prize in 1992, the Achievement Award of IEICE in 2000, and the ACM Gordon Bell Prize in 2009.