Received March 17, 2019, accepted April 9, 2019, date of publication April 12, 2019, date of current version May 2, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2910824

Multi-FPGA Accelerator Architecture for Stencil Computation Exploiting Spacial and Temporal Scalability

HASITHA MUTHUMALA WAIDYASOORIYA AND MASANORI HARIYAMA, (Member, IEEE)
Graduate School of Information Sciences, Tohoku University, Sendai 980-8579, Japan

Corresponding author: Hasitha Muthumala Waidyasooriya (hasitha@tohoku.ac.jp)

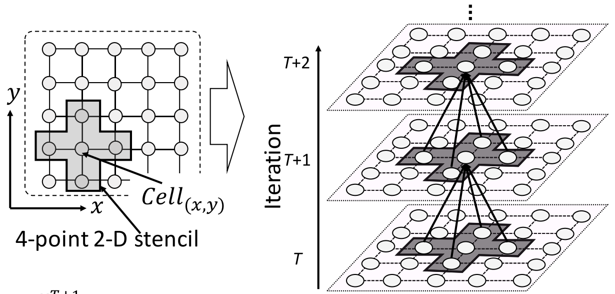
ABSTRACT After the introduction of the OpenCL-based FPGA accelerator design method, FPGAs are getting very popular among high-performance computing. The key to achieving high performance using FPGAs is to design pipelined accelerators. We can increase the pipeline depth beyond the border of one FPGA by connecting multiple FPGAs using high-speed QSFP (quad small form-factor pluggable) connectors. Such a deeply-pipelined accelerator using multiple FPGAs works similar to a single very large FPGA. In this paper, we propose a multi-FPGA accelerator architecture for stencil computation by scaling in spacial and temporal dimensions. According to the experimental results, we achieved performance up to 950 GFLOP/s using one FPGA and nearly doubled the performance using two FPGAs. We achieved a high power-efficiency with competitive performances compared to high-end GPUs.

INDEX TERMS OpenCL for FPGA, high performance computing, stencil computation, multi-FPGA acceleration.

I. INTRODUCTION

Recently, there is a growing trend to use FPGAs in high-performance computing. This is mainly due to the power-efficient computation in FPGAs, and the easy-to-use OpenCL-based programming environment [1]–[3]. In the field of high-performance computing, it is very important to efficiently use multiple FPGAs to increase performance.

FPGAs usually have a small external memory bandwidth compared to GPUs. Therefore, unlike GPUs, FPGAs do not perform efficiently for massive data-parallel operations that require access to a large amount of data in parallel. The key of achieving high performance using FPGAs is to design pipelined accelerators. Such pipelined accelerators even produce competitive performance compared to GPUs for data-intensive applications [4], [5]. We can increase the pipeline depth beyond the border of one FPGA by connecting multiple FPGAs using high-speed QSFP (quad small form-factor pluggable) connectors. It is possible to send data directly from a processing element (PE) of one FPGA to a PE in another FPGA, without using external memory as a temporal storage. Therefore, multiple FPGAs perform similar to one large FPGA, and we can expect near-linear performance by adding more FPGAs to the system. In order to design such an accelerator, we have to consider several factors such as how to partition the application into multiple FPGAs, how to optimize inter-FPGA data transfers, how to deal with different clock frequencies in different FPGAs, etc.



$$Cell^{T+1}_{(x,y)} = P_1 \cdot Cell^{T}_{(x,y-1)} + P_2 \cdot Cell^{T}_{(x-1,y)} + P_3 \cdot Cell^{T}_{(x+1,y)} + P_4 \cdot Cell^{T}_{(x,y+1)}$$

FIGURE 1. **Stencil computation using 4-point 2-D stencil.**

Stencil computation is an iterative method where a grid is updated in each iteration according to a fixed computation pattern [6]. As shown in Fig.1, a stencil is a shape that consists of neighboring grid-points called cells. The typical computation pattern is in the form of a sum of products.

The associate editor coordinating the review of this manuscript and approving it for publication was Junxiu Liu.

Stencil computation is used in many scientific computation applications such as electromagnetic simulations [7], iterative solvers [8], [9], fluid dynamics simulations [10], etc. Since stencil computation is an iterative computation, we can process multiple iterations in parallel on different FPGAs. We can also process multiple portions of a grid in parallel on different FPGAs. We call this, scaling in time and space dimensions. The availability of recent high-speed inter-FPGA data transfers provides an opportunity for us to explore new approaches, and improve the existing approaches such as [11] to scale computation on multiple FPGAs.

In this paper, we propose a multi-FPGA accelerator architecture to scale the stencil computation in both time and space dimensions. We evaluate the performance of several 2-D and 3-D stencil computation benchmarks using "skewed grids" and "non-skewed grids". According to the experimental results, we achieved upto 950 GFLOP/s and 1861 GFLOP/s of performance for skewed grids, using one and two FPGAs respectively. Such performances are very competitive compared to high-end GPUs. However, the internal memory capacity of the FPGA becomes a bottleneck for non-skewed grids. We can reduce the required internal memory size and improve the performance by dividing the grid into multiple partitions called tiles. In addition, we can achieve better power-efficiency using FPGAs compared to state-of-art high-end GPUs. High-power-efficiency is critical for high-performance computing systems with many nodes. The weak point of the proposed FPGA accelerators is the low-performance for double-precision computation. Since FPGAs do not contain double-precision floating-point computation units, the performance drops considerably compared to those of single-precision computations.

The rest of this paper is organized as follows. In section II, we discuss the previous approaches to accelerate stencil computation using single and multiple FPGAs. In section III, we describe the proposed approach to scale stencil computation for multiple FPGAs. In section IV, we evaluate the proposed approach and compare it against previous work that use both FPGAs and GPUs. We give our conclusions in section V.

II. PREVIOUS WORK

Stencil computation has two types of parallel operations as shown in Fig.2. Since the computations of the cells in the same iteration are mutually independent, we can compute multiple cells in parallel as shown in Fig.2a. We call this "cell-parallel" computation. The computations of cells in different iterations have a data dependency, where a value of a cell is computed using its neighboring cells in the previous iteration. Fig.2b. shows two consecutive iterations. The cells in the grey area of iteration 1 are already computed. As a result, we can compute $cell^2_{(1,1)}$ of iteration 2 using the data of its neighboring cells in iteration 1 in the grey area. Simultaneously, we can also compute $cell^1_{(2,2)}$ of iteration 1. As a result, we can compute multiple cells in different iterations in parallel. We call this "iteration-parallel" computation. Detailed

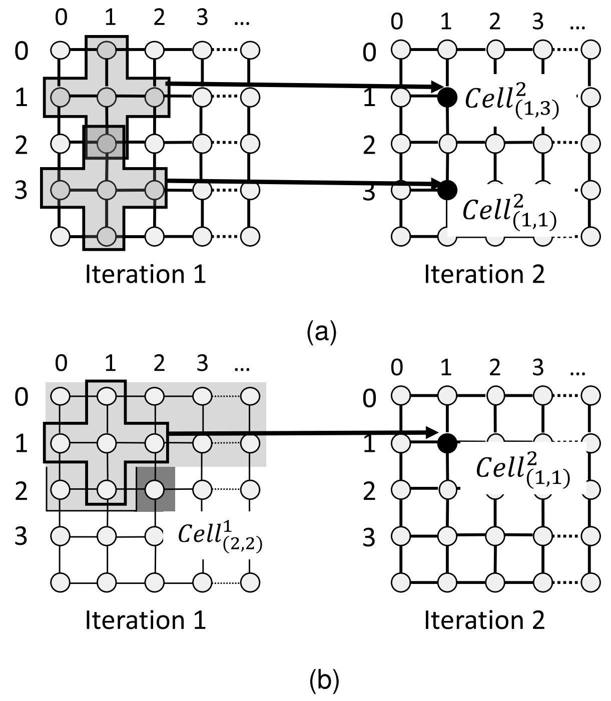


FIGURE 2. Parallel operations in stencil computation. (a) Cell-parallel computation. $cell^1_{(1,1)}$ of iteration 1 and $cell^1_{(1,3)}$ of iteration 1 are processed in parallel. (b) Iteration-parallel computation. $cell^1_{(2,2)}$ of iteration 1 and $cell^2_{(1,1)}$ of iteration 2 are processed in parallel.

explanation about the parallelism in stencil computation is available in the previous work, such as [4], [11].

A. PREVIOUS APPROACHES TO ACCELERATE STENCIL COMPUTATION USING A SINGLE FPGA

In order to implement cell-parallel computation, we have to access data in multiple cells in parallel. Since FPGAs have a relatively small memory bandwidth compared to GPUs, the computation will become memory-bound. Therefore, reducing the external memory access and increasing the operational intensity is the key to improve performance. Therefore, previous work such as [4], [11], [12] uses iteration-parallel computations, while caching the intermediate results between iterations. Fig.3 shows how this method is done. As shown in Fig.3a, the cells are computed from left-to-right and top-to-bottom one after the other. Accessed data are stored in a shift-register array as shown in Fig.3b. In each clock cycle, data of the shift-registers are moved, and a newly accessed data value is written to the first shift-register. When all neighboring data of a cell is available in the shift-register array, we can begin its computation. In the example in Fig.3, we compute $Cell^t_{1,1}$ while storing the data of $Cell^t_{3,2}$. In the next clock cycle, the data of $Cell^t_{0,0}$ is discarded and the data of $Cell^t_{4,2}$ is stored. Note that the data of $Cell^t_{0,0}$ is no longer required for any computation at this stage.

In this method, only a small portion of the grid is cached. To compute a $N \times P$ grid using a 3×3 stencil, we need

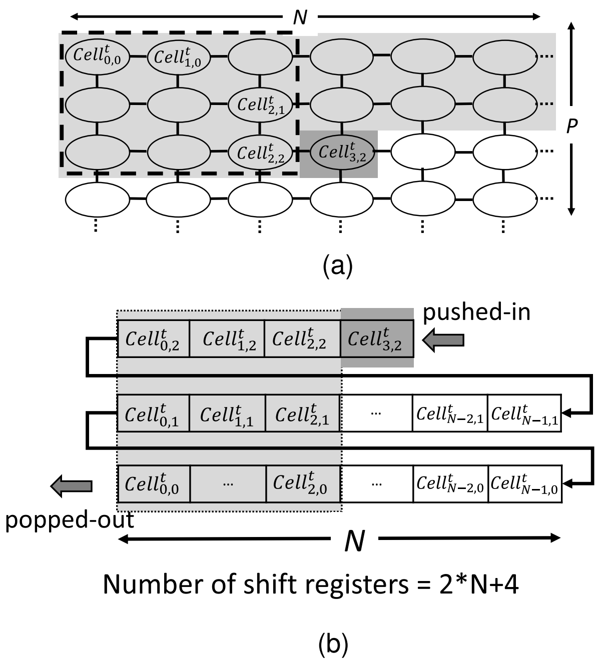

N
P
$Cell^t_{0,0}$
$Cell^t_{1,0}$
$Cell^t_{2,1}$
$Cell^t_{2,2}$
$Cell^t_{3,2}$
(a)
pushed-in
popped-out
$Cell^t_{0,2}$ $Cell^t_{1,2}$ $Cell^t_{2,2}$ $Cell^t_{3,2}$
$Cell^t_{0,1}$ $Cell^t_{1,1}$ $Cell^t_{2,1}$ … $Cell^t_{N-2,1}$ $Cell^t_{N-1,1}$
$Cell^t_{0,0}$ … $Cell^t_{2,0}$ … $Cell^t_{N-2,0}$ $Cell^t_{N-1,0}$
N
Number of shift registers = 2*N+4
(b)

FIGURE 3. The method of caching intermediate data between iterations, which is used by the accelerators proposed in [4], [11], and [12]. (a) $N \times P$ grid and 3×3 stencil. Computations are done by scanning cells from left-to-right and up-to-down. (b) Caching of data in shift-registers.

a shift-register array of length $2 \times N + 4$. The length of the shift-registers only depends on N and not P. Since the computations of the cells of an iteration are mutually independent, we can do the computation after rotating the grid by 90 degrees. In this case, the shift-register size equals to $2 \times P + 4$ and it does not depend on N. If $N \neq P$, we can access data along the dimension with the smallest size to reduce the length of the shift-register array. Therefore, the length of the shift-register array is decided by the smallest row or column for 2-D stencils, and the smallest plane for 3-D stencils. However, if the sizes of all dimensions of a grid are large, we cannot reduce the length of the shift-register size. Therefore, using this method for large grids can be difficult.

Another method is to partition the grid into multiple overlapped tiles. It is a very popular method used in CPUs and GPUs [13], [14] to accelerate stencil computation. Fig.4a shows how to divide a grid into tiles. The tiles are overlapped in order to share the data on their boundaries. This overlapped region is called *halo* region or *ghost zone*. The computations and data access of the *halo* region is redundant. Fig.4b shows how the computation of a tile is done. To compute a $N \times M$ region, we use a $(N + 2) \times (M + 2)$ tile. To compute for t iterations, we need a $(N + 2 \times t) \times (M + 2 \times t)$ tile. Therefore, the *halo* region increases with the number of iterations. This results in a large amount of redundant data access, computations and storage. Overlapped tiling method is used in the previous work such as [15]. In [15], all the intermediate data of a tile is stored in the internal memory of the FPGAs, and computation is done in cell-parallel manner. Although the internal memory in FPGAs is quite large compared to that of a GPU, it is still a scarce resource. When the tile size or the number of iterations increases, more internal memory is required and that limits the performance.

The method proposed in [16] uses overlapped tiling with iteration-parallel computation. It uses cached data of partial tiles to compute the cells in a new iteration while the cells in the previous iteration are also been computed. As a result, the cache requirement is significantly reduced. To implement this method, CPUs and GPUs require barrier synchronization of threads, which could limit the performance. This approach is used for FPGAs in [17]. It uses a shift-register array similar to that in [4] to cache the data in tiles. When there are enough data in the shift-register array, the cells in the next iteration are computed in parallel, similar to that in Fig.2b. The cached data are replaced by new once when the old data are no longer been required for further computations. It needs to cache only a fraction of a data of a tile unlike [15] that caches all the data of a tile. Moreover, the tile size is much smaller than the whole grid so that the shift-register length is much smaller compared to that in [4]. For an added advantage, this method does not need any synchronization since all the computations can be precisely scheduled in every time step using single-work-item kernels in OpenCL [3]. The disadvantages of this method are very similar to those in the original overlapped tiling approach. To process more iterations in parallel, a larger tile size is required and that leads to a large *halo* region. That results in many redundant memory accesses, computations and also storage.

B. PREVIOUS APPROACHES TO ACCELERATE STENCIL COMPUTATION USING MULTIPLE FPGAS

The research in [18] is one of the earliest work that proposed a multi-FPGA accelerator. It uses tile-based implementation. It processes tiles independently on different FPGAs and merge the computation results. The research in [11] uses HDL-based design with older generation Stratix III FPGAs. Multiple iterations are processed in parallel using different FPGAs. Multiple FPGAs are connected using 1GB/s HSTC connectors (high speed Terasic connectors) through the HSMC (high speed mezzanine card) interface [19]. The results of one FPGA are transferred to another using HSMC interface. Since the data transfer speed is small, it could often become the bottleneck. It is also extremely difficult to design this accelerator in HDL due to different clock frequencies in multiple FPGAs. The work in [20] also proposes a similar approach but uses high-speed data transfers between FPGAs. However, the accelerator in [20] can be used only for a specific fluid-dynamics simulation application and it does not consider different applications. Both [11] and [20] have not explored the feasibility of overlapped tiling, which could be very important for large grids.

C. PARTITIONING OF STENCIL COMPUTATION AND STRUCTURE OF FPGA ACCELERATORS

Stencil computation can be partitioned in both space and time dimensions. Dividing a large grid into multiple partitions and

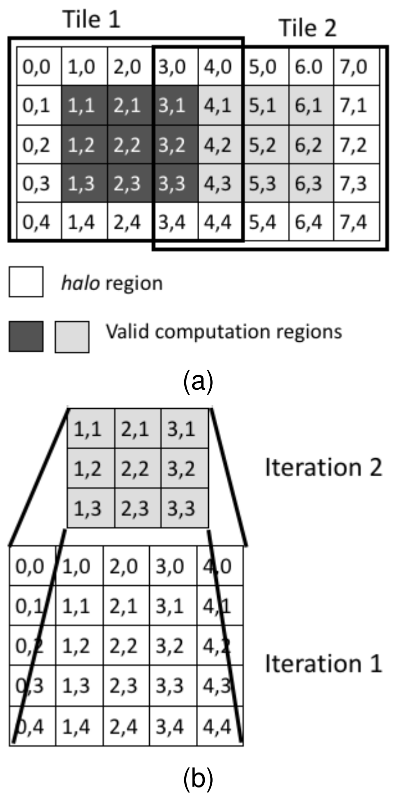


FIGURE 4. Stencil computation based on overlapped tiling. (a) Overlapped tiles. (b) Computation of a tile. To compute a 3 × 3 region in iteration 2, we need the data of a 5 × 5 tile in iteration 1.

process those partitions in parallel using multiple FPGAs is called "scaling in space dimension". Processing multiple iterations of the same grid using multiple FPGAs in parallel is called "scaling in time dimension". FPGA accelerators can be organized in several ways. One way is to connect all FPGAs to one host CPU. Another way is to connect one FPGA to another through parallel or serial connection. All those methods and different grid types should be evaluated to obtain a proper conclusion.

III. SCALING STENCIL COMPUTATION USING MULTIPLE FPGAS

We can scale stencil computation in both space and time dimensions. Such scaling is required to increase the processing speed using the computation capability of multiple FPGAs, or to process a large-size problem that can not be done on a single FPGA due to the lack of resources.

A. SCALING IN TIME DIMENSION: DEEPLY-PIPELINED FPGA-ARRAY

In this approach, we use iteration-parallel computation on multiple FPGAs to increase the processing speed. The accelerator architecture for this approach is based on our earlier work [4]. Fig.5 shows the stencil computation architecture using two FPGAs. It contains a series of pipelined computation modules (PCMs) connected in an array. The computation of an iteration is done in a PCM. The results of one PCM is stored in shift-registers and accessed by the next PCM. Multiple PCMs process multiple iterations in parallel as explained in section II. The number of PCMs represents the degree of iteration parallelism P_{iter}. The number of PEs per PCM represents the degree of cell parallelism P_{cell}. The total degree of parallelism is given by $P_{iter} \times P_{cell}$.

To increase the processing speed, we have to increase either P_{iter} or P_{cell}. If we increase P_{cell}, we have to access the input data of all the cells that are processed in parallel. That increases the required memory bandwidth. Therefore, the external memory bandwidth of the FPGA limits the maximum degree of cell parallelism. We can increase P_{iter} by using multiple FPGAs to increase the processing speed, without affecting the required memory bandwidth. For this purpose, we have to send the intermediate results of one FPGA to another directly.

1) DATA TRANSFERS AMONG MULTIPLE FPGAS USING I/O CHANNELS

Data transfers between FPGAs can be done in several ways. As shown in Fig.6, FPGAs and a host CPU are connected through the PCIe bus. Multiple FPGAs can also be connected to each other directly using QSFP connectors. To transfer the computation results between two FPGAs, we can copy the results of one FPGA to the host memory and then copy those to another. However, this approach requires data copying to the host, which is an additional overhead. Instead, we can use QSFP connectors. There are two different methods to use QSFP connectors to transfer data. One method is to use ethernet protocol as discussed in [21]. This method is applicable for both inter-FPGA data transfers and also CPU-FPGA data transfers. However, the data are sent as packets where a packet contains additional overhead such as I/P address, flags, etc. As a result, the effective data rate can be reduced. The other method is to use a serial connection between two FPGAs. The data are send from a PE in one FPGA directly to a PE in another FPGA. The connection is a dedicated one between two PEs in two FPGAs. As a result, the additional overheads such as I/P addresses are not required.

In Intel FPGA SDK for OpenCL [22], channels are used to transfer data between two kernels. A channel is a hand-shake based data transfer method between a source and a destination kernels. The source kernel writes to the channel, and waits until the data are accessed by the destination kernel. The destination kernel waits until data are available in the channel. This is called the "blocking behavior" of the channels. To connect two kernels in two FPGAs, we use I/O channels. I/O channels behave very similar to normal channels, and the source and the destination kernels are belonging to two different FPGAs. I/O channels are implemented using QSFP serial connections between two FPGAs. QSFP connectors of multiple FPGAs are directly connected using fiber optic cables. One major advantage of the I/O channel based FPGA-to-FPGA data transfer is the blocking behavior. The source kernel in one FPGA stalls if the data in the channel is not read by

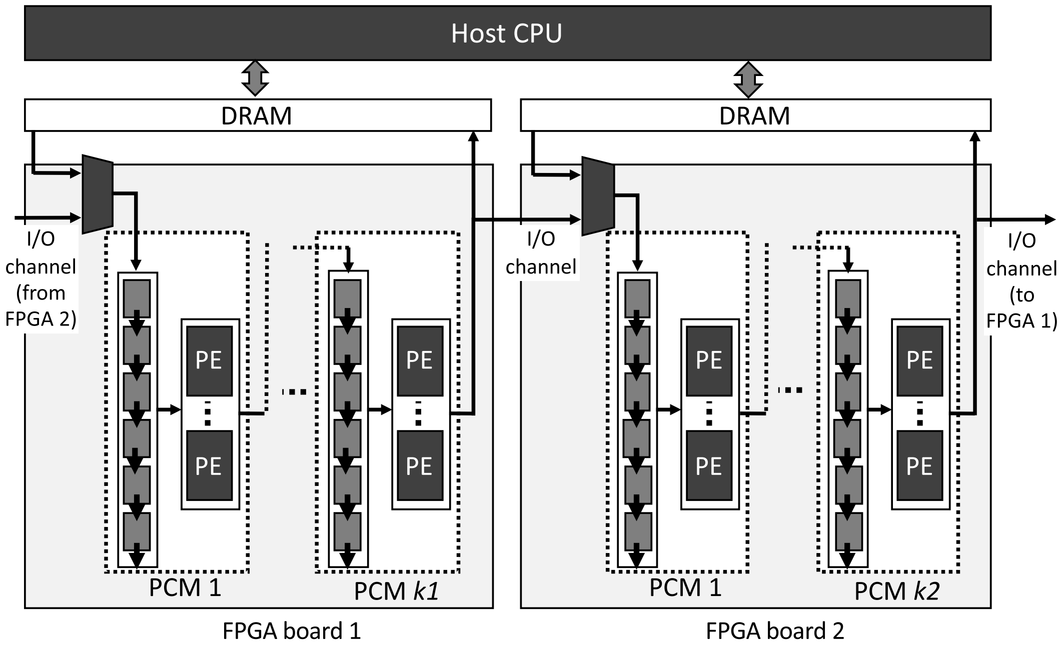


FIGURE 5. Stencil computation architecture using two FPGAs.

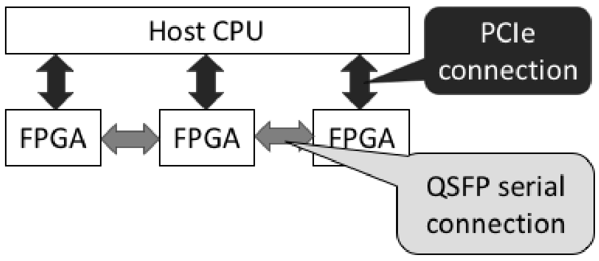


FIGURE 6. Connecting multiple FPGAs.

the destination kernel in the other FPGA. We do not get packet losses such as those in ethernet-based data transfers [21]. Therefore, we can transfer data between two FPGAs, even their clock frequencies are different. In addition, we can achieve more than 99% of the theoretical throughput of an I/O channel for both read and write accesses simultaneously. Using multiple I/O channels in parallel, we can increase the throughput by a factor of number of channels. Also note that, the next generation Stratix 10 FPGA boards have four QSFP28 connectors [23] that provide a throughput of upto 100Gbps per channel.

Fig.7 shows the stencil computation architecture using multiple FPGAs. Fig.7a shows the inter-FPGA data transfer between n FPGAs that are connected through I/O channels. Each FPGA can access either its external memory data, or channel data from the other FPGA. The datapath has two phases. In phase 1, PCM 1 of FPGA 1 accesses the data from the external memory, and the output of PCM_{k1} of FPGA 1 is written to PCM_1 of FPGA 2 through the I/O channel. Similarly, the outputs of PCM_{k1} of each FPGA are transferred to the inputs of PCM_{k1} of the next FPGA. The output of PCM_{k2} of FPGA n is written to the external memory of FPGA n. In phase 2, PCM_1 of FPGA n accesses the data from the external memory of FPGA n, and the output of PCM_{k2} of FPGA n is written to PCM_1 of FPGA $n-1$ through the I/O channel. Similarly, the rest of the outputs are transferred to the previous FPGAs through the I/O channels. The output of PCM_{k1} of FPGA 1 is written to the external memory of FPGA 1. The phases 1 and 2 are executed one after the other by the host. In each phase, the input is selected (external memory or channel) according to a control signal sent by the host. Similarly, the same control signal is used to decide whether to write the outputs to the I/O channel or to the external memory. Intel offline compiler generates complex datapath to conditionally change the data stream, resulting more resource usage and less performance.

We can simplify the datapath by designing the accelerator architecture as shown in Fig.7b. In this architecture, the output of PCM_{k-1} in FPGA 1 is transferred to PCM_1 in FPGA 2. Similar data transfers are done among all the other FPGAs. The output of PCM_{k2} of FPGA n is transferred back to FPGA 1 through an I/O channel. All the data transfers among FPGAs are done simultaneously. In this architecture, the last FPGA transfers the output to the first FPGA and then it is written to the external memory. Therefore, the external memory read and write accesses are done by PCM_1 and PCM_k of FPGA 1 respectively. All the other FPGAs do not access the external memory. The datapath among FPGAs is simplified and we do not need a control signal from the host to change the input source. If there are no data transfer bottlenecks, we can obtain near-liner performance by adding more FPGAs to the system.

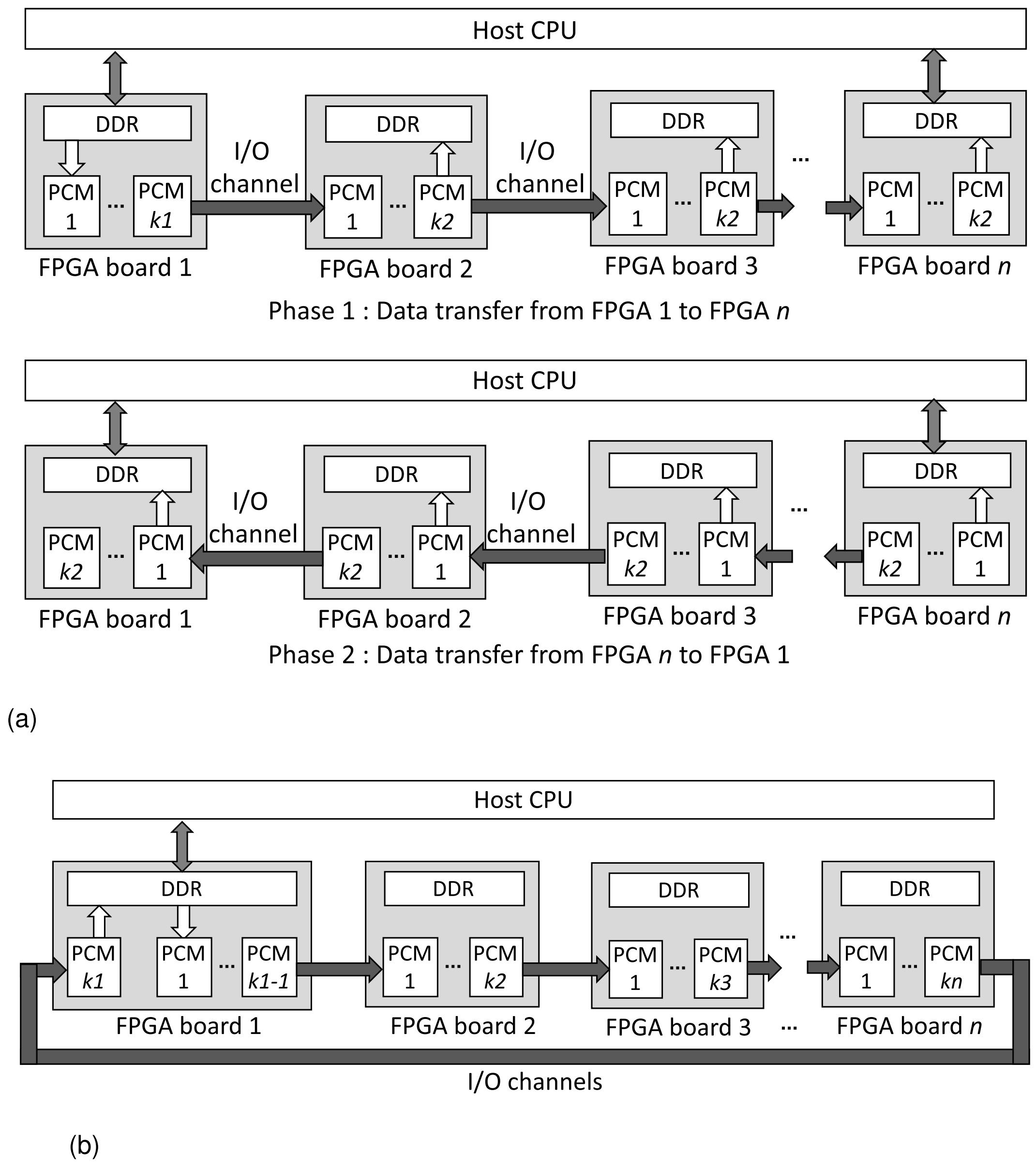


FIGURE 7. **Stencil computation accelerator using multiple FPGAs. (a) Inter-FPGA data transfer based on a two-phased datapath. The phases 1 and 2 are executed one after the other by the host. (b) Improved stencil computation accelerator architecture using a fixed datapath.**

2) BOTTLENECKS AND LIMITATIONS OF A DEEPLY-PIPELINED FPGA ARRAY

Processing speed of this approach is decided by the value of P_{iter}. To increase P_{iter}, we have to allocate more PCMs. Since a PCM contains a shift-register array, we have to implement it using resources such as RAM blocks and registers. The required shift-registers for 2-D 3×3 stencil is given by Eq.(1).

$$\textit{Shift register size} = \left\{2 \times \left\lceil \frac{d_{size}}{P_{cell}} + 4 \right\rceil \times sizeof(float) \times P_{cell}\right\} \times P_{iter} \quad (1)$$

The size of the smallest dimension of the grid is denoted by d_{size}. When d_{size} increases, the RAM block utilization can be a bottleneck. Since the total degree of parallelism is given by $P_{cell} \times P_{iter}$, we can reduce the shift-register size without affecting the performance by increasing P_{cell} and decreasing P_{iter}. Note that the total degree of parallelism does not change if we increase the degree of cell parallelism to $P_{cell} \times 2^k$ and decrease the degree of iteration parallelism to $P_{iter}/2^k$, where k is an integer. However, increasing P_{cell} also increases the required global memory bandwidth and the required channel throughput given by Eqs.(2) and (3), respectively. The kernel frequency is denoted by f.

$$\textit{Memory bandwidth} = 2 \times sizeof(float) \times P_{cell} \times f \quad (2)$$

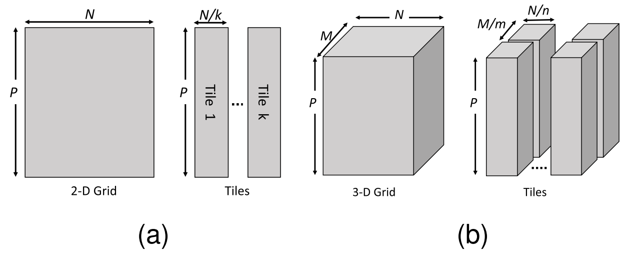


FIGURE 8. **Overlapped tiling implementations of FPGA ($P \geq M, N$). (a) Tiles of a 2-D grid. (b) Tiles of a 3-D grid.**

$$Channel\ throughput = sizeof(float) \times P_{cell} \times f \tag{3}$$

Therefore, we can trade RAM blocks with memory bandwidth and channel throughput to improve the performance. We discuss this trade-off in section IV. In addition, the maximum grid size is restricted by the global memory size of one FPGA, and we cannot increase the grid size using multiple FPGAs.

B. SCALING IN SPACE DIMENSION: SKEWED OVERLAPPED TILING

In this approach, we divide a large grid into multiple overlapped-tiles. The computation of a tile is done using both cell-parallel and iteration-parallel computations as explained in section II. The computations in a single FPGA is done similar to the previous work [17]. However, we improve the tile partition method to reduce the *halo* region. As explained in section II-A, the shift-register size is decided by the size of only a single dimension for 2-D stencils, and the sizes of only two dimensions for 3-D stencils. Therefore, instead of choosing square tiles for 2-D stencils and cubic tiles for 3-D stencils, as done in previous work, we use skewed-tiles as shown in Fig.8. Tile partitioning of 2-D and 3-D grids are shown in Figs.8a and 8b respectively. Since we do not divide the tiles along the largest dimension, we can remove the *halo* regions of it, as shown in Fig.9. Note that, how to obtain the optimal number of tiles and the optimal tile size is a difficult problem, and it is not in the scope of this paper. In the evaluation, we consider many different tile sizes and use the best results.

We can process tiles in parallel on multiple FPGAs. The architecture for this implementation is based on the one in [3]. The host computer (CPU) divides the grid into tiles and assign tiles to different FPGAs. When all the tiles are processed for P_{iter} iterations, the host read and transfer the overlapped (or shared) data among multiple FPGAs. Since we can decide the size of each dimension of a tile, we can use a small tile size to reduce the shift-register size. The data of the whole grid is shared by multiple FPGAs. Therefore, only a portion of the grid is stored in the external memory of an FPGA. As a result, we can reduce the external memory usage and process larger grids using more FPGAs.

In this method, the host have to update the data of the overlapped data regions among multiple FPGAs.

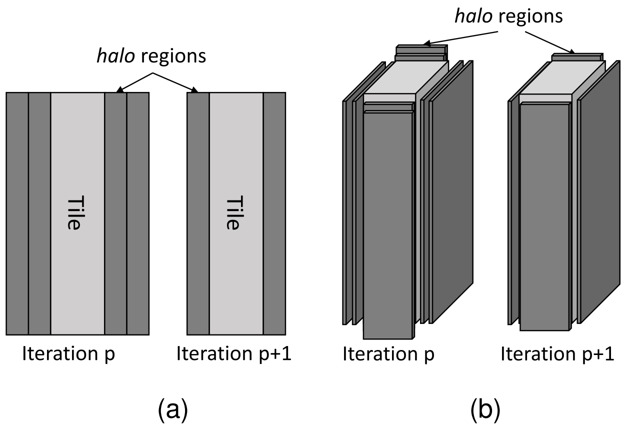


FIGURE 9. ***halo* regions of 2-D and 3-D tiles. (a) *halo* regions for 2-D tiles. (b) *halo* regions for 3-D tiles.**

The processing time required for this data transfers is usually small compared to the processing time in FPGAs. We further reduce this data access time by using inter-FPGA data transfers. After processing all tiles in all FPGAs for P_{iter} iterations, we run a small kernel that transfer only the overlapped data among FPGAs. Therefore, the inter-FPGA data transfer time is negligible compared to the computation time.

If a grid is fit into the FPGA and the RAM block usage is not a bottleneck, we must give the priority for the first approach and scale in time dimension. Since scaling in time dimension does not require redundant computations, redundant data access or redundant storage, it is very efficient and provides the best performance. However, if a grid is too large or the RAM block usage is a bottleneck, we have to evaluate both approaches to design the best accelerator. If we have many FPGAs, we can combine both approaches and scale in both space and time dimensions together.

IV. EVALUATION

We use four 2-D stencil benchmarks and two 3-D stencil benchmarks as shown in Table 1 for the evaluation. For each benchmark, we evaluate kernels with different P_{iter} and P_{cell} values. Then we chose the kernel with the smallest processing time. We use two "Nallatech 385A accelerator boards" [24] that contain Intel Arria 10 10AX115N3F40E2SG FPGAs. Compilation of the OpenCL codes is done using Quartus 17.1 with Intel FPGA SDK for OpenCL 17.1 [22]. The grid sizes we used are similar to the recent work [17], but much larger compared to the previous work, such as [25], [26].

A. EVALUATION USING 2-D AND 3-D STENCIL COMPUTATION BENCHMARKS

Table 2 shows the performance comparison of one-FPGA and 2-FPGA implementations of 2-D and 3-D stencil computation benchmarks using single-precision floating-point computations. In this evaluation, we use skewed grids where the size of one dimension of the grid is significantly smaller than the others. Such skewed grids are available

TABLE 1. **Stencil computation benchmarks.**

Benchmark	Computation
Laplace eq. 2-D	$0.25(V^t_{i,j-1} + V^t_{i-1,j} + V^t_{i+1,j} + V^t_{i,j+1})$
Diffusion 2-D	$k_1 \cdot V^t_{i,j-1} + k_2 \cdot V^t_{i-1,j} + k_3 \cdot V^t_{i,j} + k_4 \cdot V^t_{i,j} + k_5 \cdot V^t_{i,j+1}$
Hotspot 2-D	$V^t_{i,j} + k \times (power_{i,j} + (V^t_{i+1,j} + V^t_{i-1,j} - 2.0V^t_{i,j}) \times R_x$ $(V^t_{i,j+1} + V^t_{i,j-1} - 2.0V^t_{i,j}) \times R_y + (TEMP_{AMB} - V^t_{i,j}) \times R_z)$
Jacobi 9-pt. 2-D	$k_1.V^t_{i-1,j-1} + k_2.V^t_{i,j-1} + k_3.V^t_{i+1,j-1} + k_4.V^t_{i-1,j} + k_5.V^t_{i,j} + k_6.V^t_{i+1,j} +$ $k_7.V^t_{i-1,j+1} + k_8.V^t_{i,j+1} + k_9.V^t_{i+1,j+1}$
Diffusion 3-D	$k_1.V^t_{i,j-1,k} + k_2.V^t_{i-1,j,k} + k_3.V^t_{i,j,k-1} + k_4.V^t_{i,j,k} + k_5.V^t_{i+1,j,k} + k_6.V^t_{i,j+1,k}$
Hotspot 3-D	$k_1 \cdot V^t_{i,j,k} + k_2 \cdot V^t_{i,j-1,k} + k_3 \cdot V^t_{i,j+1,k} + k_4 \cdot V^t_{i+1,j,k} + k_5 \cdot V^t_{i-1,j,k} + k_6 \cdot V^t_{i,j,k-1} +$ $k_7 \cdot V^t_{i,j,k+1} + k_8 \cdot power_{i,j,k} + k_9 \cdot TEMP_{AMB}$

TABLE 2. **Comparison of 1-FPGA and 2-FPGA implementations of the stencil computation benchmarks with skewed grids.**

Benchmark	Grid size	1-FPGA implementation			2-FPGA implementation			Speed -up
		Performance		Frequency	Performance		Frequency	
		GFLOP/s	Cells/s	(MHz)	GFLOP/s	Cells/s	(MHz)	
Laplace eq. 2-D	4096×65536	659	164.1	352	1210	302.5	315	1.84
Diffusion 2-D	4096×65536	763	84.8	356	1491	165.7	348	1.95
Hotspot 2-D	4096×65536	740	49.3	309	1450	96.7	302	1.96
Jacobi 9-pt. 2-D	4096×65536	950	55.9	369	1861	109.5	352	1.96
Diffusion 3-D	$128 \times 128 \times 8192$	628	48.3	276	1281	98.5	275	2.04
Hotspot 3-D	$128 \times 128 \times 8192$	630	37.1	240	1148	67.6	267	1.82

TABLE 3. **Resource utilization of 1-FPGA implementations of skewed grids. Grayed areas indicate the bottleneck resources.**

Benchmark	Logic resources	DSP blocks	Memory (MB)	RAM blocks	Required memory bandwidth
Laplace eq. 2-D	161,634 (38%)	1440 (95%)	2.2 (34%)	1142 (42%)	66.3 %
Diffusion 2-D	116,365 (27%)	1440 (95%)	1.3(19%)	737 (27%)	66.9%
Hotspot 2-D	134,723 (32%)	1440 (95%)	3.0 (45%)	1627 (60%)	58.1%
Jacobi 9-pt. 2-D	114,113 (27%)	1406 (93%)	1.5(23%)	832 (31%)	34.7%
Diffusion 3-D	135,197 (32%)	1408 (93%)	3.1 (47%)	1449 (53%)	51.9%
Hotspot 3-D	147,345 (34%)	1440 (95%)	5.4 (81%)	2396 (88%)	90.1%

in many real world applications. For example, *Hotspot 3-D* benchmark [27] is often used to simulate the temperature distribution of 3-D LSIs [28]. Another example is FDTD simulation of fiber-optic cables [29] where the cross-section of a cable is extremely small compared to its length. Therefore, the size of dimensions corresponds to the cross section is small compared to the size of the dimension corresponds to the length. As shown in Table 2, the processing speeds of 2-FPGA implementations are two times larger compared to those of 1-FPGA implementations. The processing speed depends on the clock frequency and the degree of parallelism. The degree of parallelism is doubled by processing more iterations in parallel in two FPGAs, while the clock frequency is nearly the same. Therefore, we can obtain near-linear speed-up using multiple FPGAs. Due to the blocking behavior of the I/O channels as explained in section III-A.1, we can transfer data between multiple FPGAs even they have different clock frequencies. However, the performance of the whole system is decided by the smallest clock frequency among all FPGAs.

Tables 3 and 4 show the resource utilization and required global memory bandwidth of 1-FPGA and 2-FPGA implementations respectively. Table 4 also shows the required channel throughput for 2-FPGA implementations. The grayed areas indicate the bottleneck resources. In both tables, performances are restricted by the DSP utilization. According to our experience, it is possible to obtain more than 90% of the external memory bandwidth for sequential external memory access. We can also obtain more than 99% of the theoretical channel throughput. For all benchmarks, the required channel throughput and memory bandwidth can be achieved. Even though stencil computation is usually memory-bound in CPUs and GPUs, neither the external memory access nor the inter-FPGA data transfers are bottlenecks for any stencil computation benchmark that use skewed grids. As a result, we can increase the performance by adding more FPGAs to the system.

Table 5 shows the power consumption of one and two FPGA implementations. The power consumption of the FPGA boards are measured using MMD library functions of the Nallatech BSP 17.1. It provides access to the power sensors on the FPGA board. The power consumption is measured for the whole board including the FPGA, two memory modules, two QSFP connections and all other on-board components. The power consumption per FPGA is around 45W∼58W for all benchmarks, which is smaller compared to the typical power consumption of GPUs.

Table 6 shows the performance comparison of 1-FPGA and 2-FPGA implementations for non-skewed grids. The number of cells and the computation amount are the same as those of the skewed grids. The performances of *Laplace eq. 2-D* and all 3-D benchmarks have reduced significantly compared to their performances using skewed-grids in Table 2. In addition, the performance does not scale linearly for some benchmarks.

Table 7 shows the possible bottlenecks of stencil computations using non-skewed grids. The processing speed is mainly restricted by the global memory bandwidth, RAM block usage, DSP usage or channel throughput. Non-skewed grids require a large amount of shift-registers to store the intermediate results among iterations. Therefore, the demand for RAMs blocks is large. As explained in section III-A.2, we can trade global memory bandwidth for RAM blocks and increase the degree of cell-parallelism. This results in a shorter lifetime of the intermediate data, thus reducing the amount of shift registers, as shown by Eq.(1). However, this also require more data to be accessed in parallel from the global memory or from the other FPGAs and leads to data access bottlenecks. As a result, either the global memory bandwidth, channel throughput or the RAM block usage is a bottleneck for all 3-D benchmarks and also for *Laplace eq. 2-D* benchmark. The performance of these benchmarks have severely reduced despite having a lot of unused DSP blocks.

Table 8 shows the trade-off between the RAM block usage and the data access for *Laplace eq. 2-D* benchmark. When the degree of cell-parallelism is 4 ($P_{cell} = 4$), the RAM block usage is a bottleneck. As a result, the degree of total parallelism ($P_{cell} \times P_{iter}$) is small. When $P_{cell} = 8$, we can increase the degree of parallelism by nearly 50% compared to that of $P_{cell} = 4$, using a similar amount of RAM blocks. However, the required memory bandwidth and the required channel throughput are also increased by nearly 50%. Since the memory bandwidth is not a bottleneck and the required channel throughput is still close to the theoretical maximum, we can see a large performance improvement. When $P_{cell} = 16$, the degree of parallelism is increased further compared to that of $P_{cell} = 8$, and the RAM block usage is no longer a bottleneck. However, the processing speed is reduced significantly. When $P_{cell} = 16$, the required channel throughput is 182%, and the memory bandwidth is also a bottleneck. Due to these bottlenecks, the processing speed is decreased. This

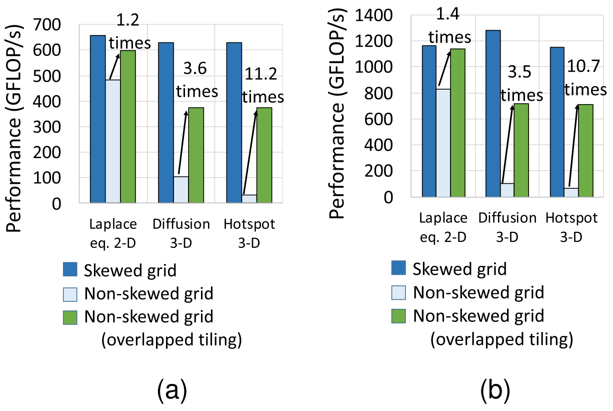


FIGURE 10. Performance comparison of "skewed grids", "non-skewed grids (no tiling)" and non-skewed grids with overlapped tiling". (a) 1-FPGA implementation. (b) 2-FPGA implementation.

shows that we have to consider both, the RAM block usage and the data access bottlenecks to improve the performance. Note that, despite having RAM block usage limitations or data access bottlenecks, we can still improve the performance by adding more FPGAs to the array. However, each FPGA would give sub-optimal performance.

We can increase the performance of the stencil computations of non-skewed grids by dividing the grid into "skewed overlapped tiles". Fig.10 shows the performance comparisons using "skewed grids", "non-skewed grids and "non-skewed grids divided into tiles". We use the three benchmarks that have a very high RAM block utilization compared to DSPs for this evaluation. The performances of 1-FPGA and 2-FPGA implementations of non-skewed grids are improved significantly using overlapped tiles, as shown in Figs.10a and 10b. However, the performances of the non-skewed grids are still smaller than those of the skewed grids, despite both grid types contain the same number of cells and require the same amount of computations. The reason for this is the redundant computations and memory accesses of the *halo* regions of the overlapped tiles.

Fig.11 shows the speed-up of 2-FPGA implementations using non-skewed grids by scaling in either time or space dimension. We used the skewed-overlapped–tiling implementation for all 3-D stencil benchmarks and *Laplace eq. 2D* benchmark, since the RAM block usage is a bottleneck, and scale the computation in space dimension using two FPGAs. We compute the whole grid without dividing it into tile for all the other benchmarks, and scale the computation in time dimension. The performances are nearly doubled when using two FPGAs for all the benchmarks. This results are very similar to the ones with skewed grids in Table 2. This shows, we can nearly double the performance using two FPGAs, for any grid type. Theoretically, it is possible to achieve near-linear speed-up using multiple FPGAs, since we can scale the computation in either time or space dimension depending on the bottlenecks.

TABLE 4. Resource utilization of 2-FPGA implementations. Grayed areas indicate the bottleneck resources.

Benchmark		Logic resources	DSP blocks	Memory (MB)	RAM blocks	Required memory bandwidth	Required channel throughput
Laplace eq. 2-D	FPGA 1	175,009 (41%)	1440 (95%)	4.1 (62%)	1983 (73%)	29.7%	50.4%
	FPGA 2	160,203 (38%)	1440 (95%)	4.1 (62%)	1970 (73%)		
Diffusion 2-D	FPGA 1	118,336 (28%)	1416 (93%)	2.2 (33%)	1117 (41%)	32.7%	55.7%
	FPGA 2	111,396 (26%)	1440 (95%)	2.2 (34%)	1118 (41%)		
Hotspot 2-D	FPGA 1	135,648 (32%)	1440 (95%)	3.0 (45%)	1678 (62%)	56.9%	48.4%
	FPGA 2	135,648 (32%)	1440 (95%)	3.0 (45%)	1678 (62%)		
Jacobi 9-pt. 2-D	FPGA 1	118,663 (28%)	1443 (95%)	1.6 (24%)	843 (31%)	33.1%	56.2%
	FPGA 2	109,454 (26%)	1443 (95%)	1.6 (24%)	818 (30%)		
Diffusion 3-D	FPGA 1	135,197 (32%)	1408 (93%)	3.1 (47%)	1449 (53%)	51.8%	88.1%
	FPGA 2	135,197 (32%)	1408 (93%)	3.1 (47%)	1449 (53%)		
Hotspot 3-D	FPGA 1	153,392 (36%)	1440 (95%)	4.4 (66%)	2466 (91%)	75.3%	85.3%
	FPGA 2	153,392 (36%)	1440 (95%)	4.4 (66%)	2466 (91%)		

TABLE 5. Power consumption of the FPGA accelerator board for stencil computations.

Benchmark	1-FPGA implementation		2-FPGA implementation	
	Power (W)	Performance per Watt $(\frac{GLOP/s}{W})$	Power (W)	Performance per Watt $(\frac{GLOP/s}{W})$
Laplace eq. 2-D	49.1	13.1	100.4	12.0
Diffusion 2-D	50.5	15.1	99.2	15.0
Hotspot 2-D	48.8	15.1	94.0	15.4
Jacobi 9-pt. 2-D	52.9	17.9	107.6	17.3
Diffusion 3-D	57.2	11.0	117.7	10.9
Hotspot 3-D	45.6	13.8	91.0	12.6

TABLE 6. Comparison of 1-FPGA and 2-FPGA implementations of the stencil computation benchmarks with non-skewed grids. (overlapped tiling is not used).

Benchmark	Grid size	1-FPGA implementation			2-FPGA implementation			Speed -up
		Performance		Frequency	Performance		Frequency	
		GFLOP/s	Cells/s	(MHz)	GFLOP/s	Cells/s	(MHz)	
Laplace eq. 2-D	16384×16384	485	121.2	299	828	206.9	327	1.71
Diffusion 2-D	16384×16384	740	82.2	345	1333	148.1	320	1.80
Hotspot 2-D	16384×16384	611	40.7	296	1204	80.3	300	1.97
Jacobi 9-pt. 2-D	16384×16384	874	51.4	340	1769	104.0	340	2.02
Diffusion 3-D	$512 \times 512 \times 512$	105	8.1	267	100	15.5	271	0.95
Hotspot 3-D	$512 \times 512 \times 512$	33	2.0	221	66.7	3.9	242	2.00

Table 9 shows the 1-FPGA performance of double-precision floating-point computation using skewed grids. Processing speeds of the double-precision computations are significantly smaller compared to those of the single-precision computations shown in Table 2. FPGAs only contain single-precision computation units, and do not contain double-precision computation units. Therefore, double-precision multiplication is done using logic resources, registers and DSPs, while double-precision addition is done using logic resources and registers. Since *Laplace eq. 2-D* benchmark can be done using additions only, we can see 0% DSP usage. (Multiplication by 0.25 can be implemented using bit-shift operations). The performances are limited mainly by the logic resource utilization. Usually, it is very difficult to fit a kernel that has more than 80% logic resource utilization. For double-precision computations, a CPU or a double-precision capable GPU such as P100 or V100 is more suitable than FPGAs.

B. COMPARISON AGAINST OTHER FPGA-BASED ACCELERATORS

Fig.12 shows the comparison against the previous work [17] and [30]. This work and [17] use the same type of FPGA boards, the same benchmarks and the same grid sizes.

TABLE 7. Bottleneck factor for the stencil computations of non-skewed grids. (Overlapped tiling is not used. Hardware resources are shown for each FPGA separately in 2-FPGA implementation.).

Benchmark	1-FPGA implementation			2-FPGA implementation			
	DSP blocks	RAM blocks	Required bandwidth	DSP blocks	RAM blocks	Required bandwidth	Req. channel throughput
Laplace eq. 2-D	1440 (95%)	1876 (69%)	112%	1008 (66%) 1008 (66%)	2506 (92%) 2506 (92%)	61%	105%
Diffusion 2-D	1440 (95%)	1872 (69%)	65%	1440 (95%) 1440 (95%)	1903 (70%) 1903 (70%)	60%	102%
Hotspot 2-D	1440 (95%)	2496 (92%)	106%	1440 (95%) 1440 (95%)	2546 (94%) 2546 (94%)	113%	96.0%
Jacobi 9-pt. 2-D	1387 (91%)	1279 (47%)	64%	1406 (92%) 1443 (95%)	2273 (84%) 2300 (85%)	32%	54%
Diffusion 3-D	256 (17%)	2364 (87%)	100%	256 (17%) 256 (17%)	2364 (87%) 2364 (87%)	102%	173%
Hotspot 3-D	144 (9%)	2429 (90%	125%	144 (9%) 144 (9%)	2390 (88%) 2390 (88%)	91%	155%

TABLE 8. Trade-off between RAM block usage and data access for 2-FPGA implementation of *Laplace eq. 2-D* benchmark.

Parallelism		Resources		Data access		Performance
P_{cell}	P_{iter}	DSP blocks	RAM blocks	Required bandwidth	Required channel throughput	GFLOP/s
4	44	528 (35%) 528 (35%)	2579 (95%) 2565 (94%)	33%	56%	487
8	42	1008 (66%) 1008 (66%)	2506 (92%) 2506 (92%)	61%	105%	828
16	30	1440 (95%) 1440 (95%)	1890 (70%) 1890 (70%)	107%	182%	471

TABLE 9. 1-FPGA implementations of double-precision floating-point stencil computation benchmarks. Grayed areas indicate the bottleneck resources.

Benchmark	Performance (GFLOP/s)	Clock Frequency (MHz)	Resource utilization				
			Logic resources	Registers	Memory (MB)	RAM blocks	DSP blocks
Laplace eq. 2-D	115	273	362074 (85%)	716,611	2.0 (30%)	1056 (39%)	0 (0%)
Diffusion 2-D	104	294	357,436 (84%)	748,742	1.6 (24%)	853 (31%)	720 (47%)
Hotspot 2-D	101	263	357,749 (84%)	739,379	3.7 (56%)	2051 (76%)	416 (27%)
Jacobi 9-pt. 2-D	93	282	353,403 (83%)	430,118	0.7 (10%)	467 (17%)	720 (47%)
Diffusion 3-D	74	260	323,812 (76%)	639,492	3.1 (47%)	1889 (70%)	616 (41%)
Hotspot 3-D	52	221	270,302 (63%)	536,729	3.9 (59%)	2225 (82%)	448 (30%)

The work in [30] uses a fully automated approach to implement stencil computations on AlphaData ADM-PCIE-KU3 FPGA board that contains Xilinx XCKU060 FPGA. All grids are non-skewed ones. When using a single FPGA, the processing speeds of the proposed method are larger than those of [30]. One major reason for the better performance in this work is the larger memory bandwidth and internal memory capacity of our Nallatech 385A board compared to the ADM-PCIE-KU3 board used in [30]. We also achieved similar or slightly better processing speed compared to those of [17]. Since we used skewed tiles, the *halo* reghion is much smaller compared to the square or cubic tiles used in [17]. This could be the reason for the better performance in this work. Moreover, we can improve the performance more than 1.8 times by using two FPGAs, and can expect near-liner performance by using more FPGAs.

There are some other works such as [11], [12], [20], [26] that shows the performances of stencil computation accelerators using FPGAs. The work in [11] shows 260 GFLOP/s using 9 Stratix III DE3 FPGA boards [31]. The fluid simulation accelerator proposed in [12] shows 519 GFLOP/s using one FPGA, and a similar accelerator proposed in [20] shows

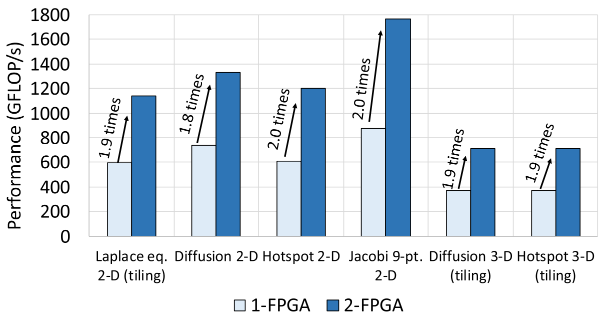


FIGURE 11. Performance comparison of 1-FPGA and 2-FPGA implementations using non-skewed grids. Overlapped tiling is used for the benchmarks that the RAM block usage is a bottleneck.

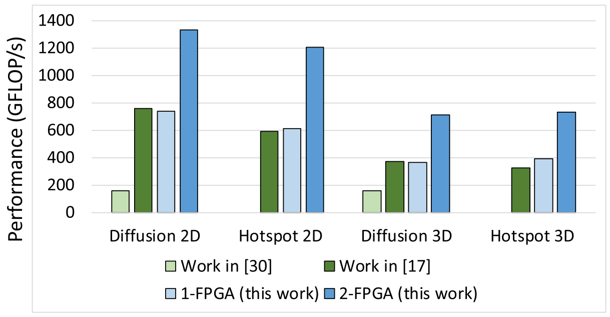


FIGURE 12. Performance comparison against [17] and [30].

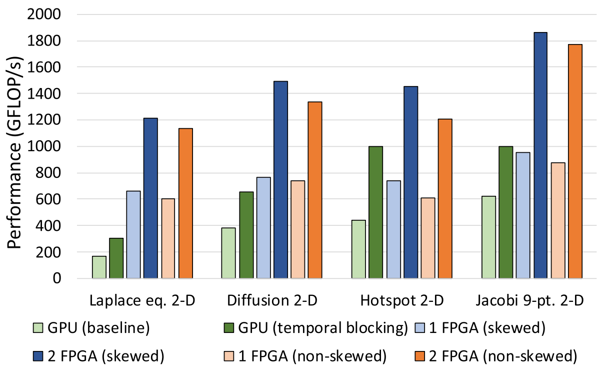


FIGURE 13. Performance comparison against GPU-implementations (Titan X Pascal) for 2-D stencil computations. The performance for both skewed and non-skewed grids are evaluated.

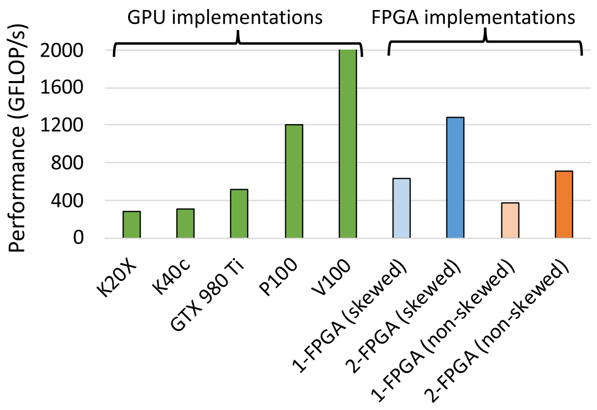


FIGURE 14. Performance comparison against GPU-implementations of *diffusion 3-D* benchmark. The performance of GPUs are obtained from [17] and [34].

4,808 GFLOP/s using 16 FPGAs. The work in [26] shows less than 12 GFLOP/s processing speed using low-end Xilinx Zinq FPGAs. However, a direct and a fair comparison with these works is difficult due to the differences in FPGA boards, unknown algorithms an unknown grid sizes, etc.

C. COMPARISON AGAINST GPU-BASED ACCELERATORS

To compare the performance with GPU accelerators, we implement 2-D stencil computation benchmarks on Titan XP (Pascal) GPU [32]. We use two types of GPU implementations. The baseline GPU performance is achieved without using any cache blocking method. We also evaluate the GPU performance using temporal blocking. Temporal blocking [13], [14], [16] is a technique that uses the cache or shared memory to re-use the data between consecutive iterations without accessing the external memory. The baseline performances of GPUs are 76% to 91% of the maximum performances predicted by the roofline model [33], and we achieved 1.5 to 2.3 times speed-up using temporal blocking. We achieved higher performances for *Laplace eq. 2-D* and *Diffusion 2-D* benchmarks, while lower performances for *Hotspot 2-D* and *Jacobi 9-pt. 2-D* benchmarks using a single FPGA. The 2-FPGA implementation provides the highest performances for all benchmarks. The comparison results against GPU implementations are similar for both skewed and non-skewed grids of 2-D stencils, despite the small performance reduction for non-skewed ones.

Fig.14 shows the comparison with the GPU using *diffusion 3-D* benchmark. The performance of K20X GPU is obtained from the work in [34], and the performances of the other GPUs are obtained from [17]. Both [17] and [34] use temporal blocking to increase the performance. The performances of 1-FPGA and 2-FPGA implementations of skewed-grids are slightly better than the performances of "GTX 980 Ti" and "P100" GPUs respectively. We observed a 40% performance reduction for non-skewed grids compared to skewed grids. The latest high-end V100 GPU gives the largest processing speed among all implementations. We need four FPGAs to exceed the performance of a single V100 GPU.

Fig.15 shows power-efficiency comparison of GPU and FPGA implementations using *Diffusion 3-D* benchmark. Power consumption data of FPGAs are measured using MMD library API of the Nallatech BSP 17.1, while the power consumption data of GPUs are taken from [17]. Work in [17] measured the power consumption using NVDIA NVML library [35] that can access the on-board power sensors of the GPU boards. The power-efficiency is obtained by dividing the performance from the average power consumption. The average power consumption is calculated by measuring the power consumption over hundred times. According to the results,

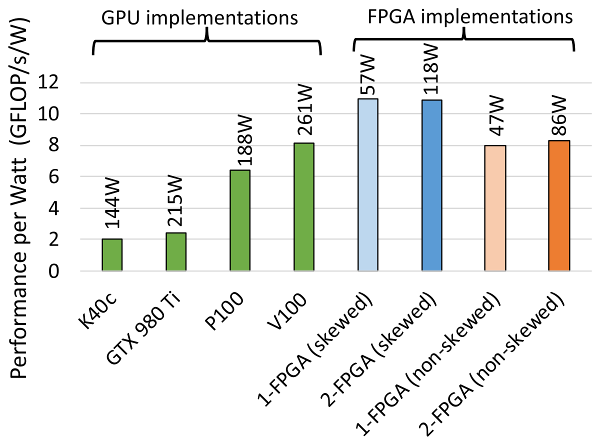


FIGURE 15. Comparison of the power-efficiency of GPU and FPGA implementations using *diffusion 3-D* benchmark. The power-efficiency data of GPUs are taken from [17].

the power-efficiency of FPGA implementations are similar or better than all GPU implementations. We achieved a better power-efficiency using FPGAs, especially for the skewed-grids. Note that the *Diffusion 3-D* benchmark has the worst power-efficiency among all FPGA implementations as shown in Table 5. Therefore, we could expect even better results by comparing the power-efficiency of the other benchmarks. However, it is very difficult to do a fair comparison due to the unavailability of power consumption data in the previous work of optimized GPU implementations. In addition, the power consumption of the FPGA implementations are significantly smaller compared to those of GPU implementations. This observation is very important when designing supercomputers. If the power consumption is large, we need large power supply units, better cooling systems and also have to deal with a large peak current. Since all of these adds-up to the total cost, we can see a huge potential in FPGAs in the field of high-performance computing. The next generation Stratix 10 FPGAs [23] are expected to provide much higher performance compared to middle-range Arria 10 FPGAs we used in this work.

V. CONCLUSION

In this paper, we proposed a scalable accelerator for stencil computations using multiple FPGAs. We can scale stencil computation in both time and space dimensions. Scaling in time dimension does not require redundant computations or memory accesses. However, this method is suitable for stencil computations of skewed grids or some 2-D stencils where the RAM block usage is not a bottleneck. Usually, the RAM block usage is large for 3-D stencils with non-skewed grids, and scaling in space dimension is required to accelerate these computations. We can nearly double the performance of all benchmarks with any grid-type using two FPGAs, by scaling on either time or space dimension. Theoretically, we can expect near-linear performance using more FPGAs. We have shown that our 2-FPGA accelerator produces over 1 TFLOP/s of processing speed for all 2-D and 3-D stencil computations. The performance are comparable to high-end GPUs at a significantly low power consumption. The power-efficiency is better than many high-end GPUs. As a result, we can see a huge potential to use FPGAs in the field of high-performance computing.

Obtaining the optimal tile size for multi-FPGA accelerator is an important area that should be considered in future work. Previous work such as [17] and [30] have formulated models to predict performance of single FPGA implementations. We have to include inter-FPGA communications also in performance models of multi-FPGA accelerators to find the optimal tile size.

REFERENCES

[1] (2015). *The Open Standard for Parallel Programming of Heterogeneous Systems*. [Online]. Available: https://www.khronos.org/opencl/
[2] T. S. Czajkowski *et al.*, "OpenCL for FPGAs: Prototyping a compiler," in *Proc. Int. Conf. Eng. Reconfigurable Syst. Algorithms (ERSA)*, Feb. 2012, p. 1.
[3] H. M. Waidyasooriya, M. Hariyama, and K. Uchiyama, *Design FPGA-Based Computing Systems With OpenCL*. New York, NY, USA: Springer, 2018.
[4] H. M. Waidyasooriya, Y. Takei, S. Tatsumi, and M. Hariyama, "OpenCL-based FPGA-platform for stencil computation and its optimization methodology," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 5, pp. 1390–1402, May 2017.
[5] S. Tatsumi, M. Hariyama, K. Ito, and T. Aoki, "An FPGA accelerator for PatchMatch multi-view stereo using OpenCL," *J. Real-Time Image Process.*, vol. 5 pp. 1–13, Mar. 2018.
[6] R. G. Brickner, K. Kennedy, J. Mellor-Crummey, and G. H. Roth, "Compiling stencils in high performance Fortran," in *Proc. ACM/IEEE Conf. Supercomput.*, Aug. 1997, pp. 1–20.
[7] K. Yee, "Numerical solution of initial boundary value problems involving Maxwell's equations in isotropic media," *IEEE Trans. Antennas Propag.*, vol. 14, no. 3, pp. 302–307, May 1966.
[8] W. M. Kahan, "Gauss-Seidel methods of solving large systems of linear equations," Ph.D. dissertation, Univ. Toronto, Toronto, Ontario, Canada, 1958.
[9] L. A. Hageman and D. M. Young, *Iterative Methods and Applications*. Chelmsford, MA, USA: Courier Corporation, 2012.
[10] G. Karniadakis and S. Sherwin, *Spectral/hp Element Methods for Computational Fluid Dynamics: Second Edition*. New York, NY, USA: Oxford Univ. Press, 2013.
[11] K. Sano, Y. Hatsuda, and S. Yamamoto, "Multi-FPGA accelerator for scalable stencil computation with constant memory bandwidth," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 3, pp. 695–705, Mar. 2014.
[12] K. Sano and S. Yamamoto, "FPGA-based scalable and power-efficient fluid simulation using floating-point DSP blocks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 10, pp. 2823–2837, Mar. 2017.
[13] J. Guo, G. Bikshandi, B. B. Fraguela, and D. Padua, "Writing productive stencil codes with overlapped tiling," *Concurrency Comput. Pract. Exper.*, vol. 21, no. 1, pp. 25–39, Mar. 2009.
[14] J. Meng and K. Skadron, "A performance study for iterative stencil loops on GPUS with ghost zone optimizations," *Int. J. Parallel Program.*, vol. 39, no. 1, pp. 115–142, Jun. 2011.
[15] Y. Takei, H. M. Waidyasooriya, M. Hariyama, and M. Kameyama, "FPGA-oriented design of an FDTD accelerator based on overlapped tiling," in *Proc. Int. Conf. Parallel Distrib. Process. Techn. Appl. (PDPTA)*, Jun. 2015, p. 72.
[16] A. Nguyen, N. Satish, J. Chhugani, C. Kim, and P. Dubey, "3.5-D blocking optimization for stencil computations on modern CPUs and GPUs," in *Proc. ACM/IEEE Int. Conf. High Perform. Comput., Netw., Storage Anal.*, May 2010, pp. 1–13.
[17] H. R. Zohouri, A. Podobas, and S. Matsuoka, "Combined spatial and temporal blocking for high-performance stencil computation on FPGAs using OpenCL," in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays*, Feb. 2018, pp. 153–162.

[18] L.-N. Pouchet, P. Zhang, P. Sadayappan, and J. Cong, "Polyhedral-based data reuse optimization for configurable computing," in *Proc. ACM/SIGDA Int. Symp. Field Program. Gate Arrays*, May 2013, pp. 29–38.

[19] (2009). *High Speed Mezzanine Card (HSMC) Specification*. [Online]. Available: https://www.altera.com/en_US/pdfs/literature/ds/hsmc_spec.pdf

[20] A. Mondigo, K. Sano, and H. Takizawa, "Performance estimation of deeply pipelined fluid simulation on multiple FPGAs with high-speed communication subsystem," in *Proc. IEEE 29th Int. Conf. Appl.-Specific Syst., Architectures Processors (ASAP)*, May 2018, pp. 1–4.

[21] R. Kobayashi, Y. Oobata, N. Fujita, Y. Yamaguchi, and T. Boku, "OpenCL-ready high speed FPGA network for reconfigurable high performance computing," in *Proc. Int. Conf. High Perform. Comput. Asia-Pacific Region*, May May 2018, pp. 192–201.

[22] (2018). *Intel FPGA SDK for OpenCL*. [Online]. Available: https://www.altera.com/products/design-software/embedded-software-developers/opencl/overview.html

[23] (2017). *Stratix 10 GX/SX Device Overview*. [Online]. Available: https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/hb/stratix-10/s10-overview.pdf

[24] (2018). *Nallatech 385A Accelerator Card*. [Online]. Available: http://www.nallatech.com/store/fpga-accelerated-computing/pcie-accelerator-cards/nallatech-385a-arria10-1150-fpga/

[25] J. Cong, P. Li, B. Xiao, and P. Zhang, "An optimal microarchitecture for stencil computation acceleration based on nonuniform partitioning of data reuse buffers," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 35, no. 3, pp. 407–418, Jun. 2016.

[26] G. Deest, T. Yuki, S. Rajopadhye, and S. Derrien, "One size does not fit all: Implementation trade-offs for iterative stencil computations on FPGAs," in *Proc. 27th Int. Conf. Field Program. Logic Appl. (FPL)*, Sep. 2017, pp. 1–8.

[27] S. Che *et al.*, "Rodinia: A benchmark suite for heterogeneous computing," in *Proc. IEEE Int. Symp. Workload Characterization*, Mar. 2009, pp. 44–54.

[28] C.-Y. Liu, Y.-J. Chen, and M. Hariyama, "Thermal-aware architectural design automation method for multiprocessor system-on-chips with 3D-stacked hybrid memories," in *Proc. Tohoku U-NTU Symp.*, May 2018, p. 26.

[29] H. M. Waidyasooriya, T. Endo, M. Hariyama, and Y. Ohtera, "OpenCL-based FPGA accelerator for 3D FDTD with periodic and absorbing boundary conditions," *Int. J. Reconfigurable Comput.*, vol. 2017, Aug. 2017, Art. no. 6817674.

[30] Y. Chi, J. Cong, P. Wei, and P. Zhou, "SODA: Stencil with optimized dataflow architecture," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, Feb. 2018, pp. 1–8.

[31] (2018). *Altera DE3 Development System*. [Online]. Available: http://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=39&No=260

[32] (2018). *GeForce GTX TITAN X Specifications*. [Online]. Available: https://www.geforce.com/hardware/desktop-gpus/geforce-gtx-titan-x/specifications.

[33] S. Williams, A. Waterman, and D. Patterson, "Roofline: An insightful visual performance model for multicore architectures," *Commun. ACM*, vol. 52, no. 4, pp. 65–76, 2009.

[34] N. Maruyama and T. Aoki, "Optimizing stencil computations for NVIDIA Kepler GPUs," in *Proc. 1st Int. Workshop High-Perform. Stencil Comput., Vienna*, Aug. 2014, pp. 89–95.

[35] (2015). *NVML API Reference Guide*. [Online]. Available: http://docs.nvidia.com/deploy/pdf/NVML_API_Reference_Guide.pdf

HASITHA MUTHUMALA WAIDYASOORIYA received the B.E. degree in information engineering, the M.S. degree in information sciences, and the Ph.D. degree in information sciences from Tohoku University, Japan, in 2006, 2008, and 2010, respectively, where he is currently an Assistant Professor with the Graduate School of Information Sciences. His research interests include reconfigurable computing, processor architectures for big-data processing, and high-level design methodology for VLSIs.

MASANORI HARIYAMA received the B.E. degree in electronic engineering, the M.S. degree in information sciences, and the Ph.D. degree in information sciences from Tohoku University, Sendai, Japan, in 1992, 1994, and 1997, respectively, where he is currently a Professor with the Graduate School of Information Sciences. His research interests include real-world applications such as robotics and medical applications, big data applications such as bio-informatics, high-performance computing, VLSI computing for real-world application, high-level design methodology for VLSIs, and reconfigurable computing.