FPGA-Based Systolic Computational-Memory Array for Scalable Stencil Computations

Kentaro Sano

Abstract Stencil computation is one of the typical kernels of numerical simulations, which requires acceleration for high-performance computing (HPC). However, the low operational-intensity of stencil computation makes it difficult to fully exploit the peak performance of recent multi-core CPUs and accelerators such as GPUs. Building custom-computing machines using programmable-logic devices, such as FPGAs, has recently been considered as a way to efficiently accelerate numerical simulations. Given of the many logic elements and embedded coarse-grained modules, state-of-the-art FPGAs are nowadays expected to efficiently perform floating-point operations with sustained performance comparable to or higher than that given by CPUs and GPUs. This chapter describes a case study of an FPGA-based custom computing machine (CCM) for high-performance stencil computations: *a systolic computational-memory array (SCM array) implemented on multiple FPGAs*.

1 Introduction

Numerical simulation is now an important and indispensable tool in science and engineering, which analyzes complex phenomena that are difficult to subject to experimentation. For a large-scale simulation with sufficient resolution, high-performance computing (HPC) is required. Stencil computation [4] is one of the typical kernels of high-performance numerical simulations, which include computational fluid dynamics (CFD), electromagnetic simulation based on the finite-difference time-domain (FDTD) method, and iterative solvers of a linear equation system.

K. Sano (✉)
Tohoku University, 6-6-01 Aramaki Aza Aoba, Sendai, Miyagi 980-8579, Japan
e-mail: kentah@caero.mech.tohoku.ac.jp

Since most of their computing time is occupied by the computing kernels, acceleration techniques for stencil computation have been required. However, the low operational-intensity of stencil computation makes it difficult to fully exploit the peak performance of recent accelerators such as GPUs. Operational intensity is the number of floating-point operations per data size read from an external DRAM for cache misses [30]. In general, stencil computation requires relatively many data accesses per unit operation, and therefore its operational-intensity is low. Although the recent CPUs and GPUs have been getting higher peak-performance for arithmetic operations by integrating more cores on a chip, their off-chip bandwidth has not been sufficiently increased in comparison with the peak-performance. As a result, stencil computation can enjoy only a fraction of the peak performance because cores are idle for many cycles where required data are not ready. To make matters worse, a large-scale parallel system with many CPUs and/or GPUs usually has the problem of parallel processing overhead, which limits speedup especially for computations with a low operational-intensity due to inefficient bandwidth and latency of the interconnection network. These inefficiencies of the processor-level and system-level execution cause the performance per power of supercomputers to decline. We need to address this efficiency problem because the performance per power is also a big issue in high-performance computation.

Custom computing machines (CCMs) constructed with programmable-logic devices, such as FPGAs, are expected to be another way to efficiently accelerate stencil computation because of their flexibility in building data-paths and memory systems dedicated to each individual algorithm. Especially, state-of-the-art FPGAs have become very attractive for HPC with floating-point operations due to their advancement with a lot of logic elements and embedded modules, such as DSP blocks, block RAMs, DDR memory-controllers, PCI-Express interfaces, and high-speed transceivers/receivers. High-end FPGAs are now capable of performing floating-point operations with sustained performance comparable to or higher than that achieved by CPUs and GPUs.

This chapter presents *a systolic computational-memory array* (SCM array) [20, 22, 23] to be implemented on multiple FPGAs, which is a programmable custom-computing machine for high-performance stencil computations. The SCM array is based on the SCM architecture that combines the systolic array [11, 12] and the computational memory approach [7, 17, 18, 29] to scale both computing performance and aggregate memory-bandwidth with the array size. Processing elements of the array perform 3×3 star-stencil computations in parallel with their local memories. Since this architecture is especially designed to achieve performance scalability on a multi-FPGA system, two techniques are introduced for the bandwidth and synchronization problems of inter-FPGA data-transfer: *a peak-bandwidth reduction mechanism* (BRM) and *a local-and-global stall mechanism* (LGSM).

We describe a target computation, an architecture and a design for the SCM array to be implemented on a multi-FPGA system. For three benchmark problems, we evaluate resource consumption, performance and scalability of prototype implementation with three FPGAs. We also discuss feasibility and performance for implementation with a 2D array of high-end FPGAs. We show that a single state-of-the-art FPGA can achieve a sustained performance higher than 400 GFlop/s and a multi-FPGA system scales the performance in proportion to the number of FPGAs.

2 Related Work

So far, various trials have been conducted to design FPGA-based CCMs for stencil computations in the numerical simulations based on finite difference methods, which include an initial investigation of an FPGA-based flow solver[10], an overview of an FPGA-based accelerator for CFD applications[26], a design of FPGA-based arithmetic pipelines with a memory hierarchy customized for a part of CFD subroutines[16], and proposals for FPGA-based acceleration of FDTD method [3, 6, 25]. Most of them rely on pipelining data-flow graphs on an FPGA. However, their design and discussion lack in scalability of performance and memory bandwidth which is necessary to achieve HPC. Furthermore, discussion and evaluation of the system scalability, particularly for multiple-FPGA implementation, are indispensable for HPC with a large-scale system.

Recently, several systems with a lot of tightly coupled FPGAs have been developed: BEE3, Maxwell, Cube, Novo-G, and SSA. BEE3, the Berkeley Emulation Engine 3, is designed for faster, larger and higher fidelity computer architecture research [5]. It is composed of modules with four tightly coupled Virtex-5 FPGAs connected by ring interconnection. The modules can be further connected to each other to construct a large FPGA computer. Maxwell is a high-performance computer developed by the FPGA high performance computing alliance (FHPCA), which has a total of 64 FPGAs on 32 blade servers [2]. Each blade has an Intel Xeon CPU and two Xilinx Virtex-4 FPGAs. While the CPUs are connected by an interconnection network, the FPGAs are also connected directly by their dedicated 2D torus network. The Cube is a massively parallel FPGA cluster consisting of 512 Xilinx Spartan 3 FPGAs on 64 boards [15]. The FPGAs are connected in a chain, so that they are suited to pipeline and systolic architecture.

The Novo-G is an experimental research testbed built by the NSF CHREC Center, for various research projects on scalable reconfigurable computing [8]. Novo-G is composed of 24 compute nodes, each of which is a Linux server with an Intel quad-core Xeon processor and boards of ALTERA Stratix IV FPGAs. Data transfer can be made between adjacent FPGAs, through a wide and bidirectional bus. The SSA, the scalable streaming-array, is a linear array of ALTERA Stratix III FPGAs for scalable stencil computation with a constant memory bandwidth [24]. The FPGAs are connected by a 1D ring network to flow data through a lot of computing stages on the FPGAs. By deeply pipelining iterative stencil computations with the stages, the SSA achieves scalable performance according to the size of the system.

These systems provide not only a peak computing performance scalable to the system size but also low-latency and wide-bandwidth communication among

FPGAs. Particularly the inter-FPGA communication achieved by direct connection of FPGAs is very attractive and promising for parallel computing with less overhead. Present typical accelerators, such as GPUs, suffer from the latency and bandwidth limitation of their connection via host nodes and a system interconnection network.

However, it is also challenging to efficiently use reconfigurable resources with flexibility of inter-FPGA connection for large-scale custom computing. We have to find an answer to the question of how we should use multiple FPGAs, or what architecture and designs are suitable and feasible for scalable and efficient stencil-computation. This chapter presents *the SCM array* as an answer to this question. The SCM array is a programmable custom-computing machine for high-performance stencil computations. The SCM array is based on the SCM architecture that combines the systolic array [11,12] and the computational memory approach [7,17,18,29] to scale both computing performance and aggregate memory-bandwidth with the array size. The SCM array behaves as a computing memory, which does not only store data but also perform computations with them. The processing elements of the array perform 3×3 star-stencil computations in parallel with their local memories. To achieve both flexibility and dedication, we give the SCM array a hardware layer and a software layer to execute programs with various stencil computation. The hardware layer has a simple instruction-set for only a minimum of programmability.

Since this architecture is especially designed for scalable computation on a multi-FPGA system with different clock-domains, two techniques are introduced for the bandwidth and synchronization problems of inter-FPGA data-transfer: the peak-bandwidth reduction mechanism (BRM) and the LGSM. To evaluate resource consumption, performance, and scalability for three benchmark problems, we demonstrate that the SCM array prototyped with three FPGAs achieves performance scalable to the number of devices.

The FPGA-based programmable active memory (PAM) [29] is an approach that is similar to our SCM array in terms of *the PAM* concept. PAM is a 2D array of FPGAs with an external local-memory, which behaves as a memory for a host machine while processing the stored data. Extensibility is also given by allowing PAM to be connected with I/O modules or other PAMs. On the other hand, our SCM array and its concept differ in the following ways from PAM. First, PAM is not specialized for floating-point computation. Second, the constructive unit of our SCM array is different from that of PAM. The constructive unit of PAM is PAM itself, which is an FPGA array where custom circuits are configured over multiple FPGAs. In our SCM array, each FPGA is a basic unit and has the same hardware design as a module. The array of FPGAs forms a scalable SCM array, and therefore we can easily extend the system by adding FPGAs, such as is implemented on "stackable mini-FPGA-boards."

Although the peak performance cannot be exploited due to the low operational-intensity of stencil computation, efforts have also been made to optimize the computation on GPUs [19]. Williams et al. reported the performance of 3D stencil computation with a GPU [4]. A single NVidia GTX280 GPU achieves 3D stencil computation of 36 GFlop/s, which is a much higher performance than that of a multicore processor due to the GPU's high memory bandwidth of 142 GByte/s. However, the efficiency is not high, being 46% of the double-precision peak performance of 78 GFlop/s. It is important to note here that the performance is measured without the data-transfer to/from a host PC. The inefficiency results in a low performance per power, 0.15 GFlop/sW, even though they measured the power consumption of only the GPU board.

Phillips et al. reported parallel stencil-computation on a GPU cluster [19]. By applying optimization techniques, they achieved 51.2 GFlop/s with a single NVidia Tesla C1060 similar to GTX280, which corresponds to 66% of the peak performance. However, in the case where 16 GPUs were utilized, only a $\times 10.3$ speedup was achieved for a $256^2 \times 512$ grid because of the communication/synchronization overhead among GPUs. As a result, the efficiency is reduced to 42% from 66%. These results show that while each GPU has a high peak-performance, it is difficult to obtain high sustained-performance for stencil computation, especially with multiple GPUs. Scalability is significantly limited in a large parallel system with many accelerators, and most of the entire performance is easily spoiled. We need a solution for efficiently scaling performance according to the size of the system.

3 Target Computation and Architecture

3.1 General Form of Stencil Computations

The SCM array targets iterative stencil-computation, which is described by the pseudo-code of Fig. 1. In scientific simulations, such as CFD, the partial differential equations (PDE) that govern the physical phenomena to be simulated are often numerically solved by the finite difference method. The computational kernels for the method are of stencil computations, which are given by discretizing values defined at grid points and numerically approximating the derivatives. Here we explain a 2D case for simplicity. Let $v(n,i,j)$ be values defined on a 2D computational grid, where n, i and j denote a time step, x- and y-positions, respectively. The nested inner-loops iterate the stencil computation for each grid-point (i,j) over the grid. At each grid-point, $v(n,i,j)$ is updated from time-step n to $(n+1)$ by computing $F()$ with only the values in the neighboring grid-points. We refer to the region

```
for(n=0; n<N ; n++) {  // for iterations

  for(j=0; j<y; j++)   // for grid traverse
    for(i=0; i<x ; i++) {
      // Update grid value from n to n+1
      v(n+1,i,j) := F( v(n,i,j) in S(i,j) )
    }
}
```

Fig. 1 Pseudo code of iterative stencil computation

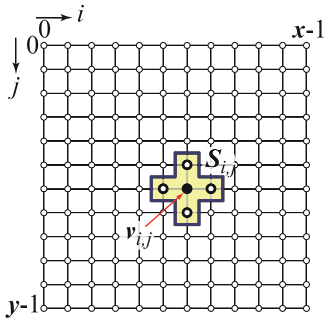


Fig. 2 Simple 3×3 star stencil

of these neighboring grid-points as *a stencil* and denote it with $S(i, j)$. Figure 2 illustrates a 2D simple example of a 3×3 star-stencil containing the five local grid-points. The outer loop of the pseudo code repeats the grid updates for time-marching computation along successive time-steps.

As reported in [20–22], $F()$ of typical stencil computation is simply a weighted sum of the neighboring values, which is written as:

$$v_{i,j}^{\text{new}} = c_0 v_{i,j} + c_1 v_{i-1,j} + c_2 v_{i+1,j} + c_3 v_{i,j-1} + c_4 v_{i,j+1} \tag{1}$$

for a 3×3 star-stencil, where c_0–c_4 are constants. We refer to (1) as *a 2D neighboring accumulation*. Similarly, a 3D neighboring accumulation requires two more terms for a $3 \times 3 \times 3$ star-stencil in 3D. Although wide stencils are used for higher-order differential schemes, they can be decomposed into 2D or 3D multiple neighboring-accumulations. Therefore we consider that neighboring accumulation for 3×3 or $3 \times 3 \times 3$ star-stencils is a general form of 2D or 3D stencil computation, respectively, which should be accelerated. The stencil computation has parallelism, which allows neighboring accumulations of grid-points to be independently performed. Furthermore, the computation of each grid-point has locality because the necessary values are localized around the point. We designed the SCM array to accelerate the neighboring accumulations by exploiting their parallelism and locality.

3.2 SCM Architecture

The SCM array is based on the SCM architecture [20,22,23], which is a combination of the programmable systolic array [11,12] and the computational memory approach [7, 17, 18, 29]. The systolic array is a regular arrangement of many processing elements (PEs) in an array, where data are processed and synchronously transmitted between neighbors across the array. Such an array provides scalable performance according to the array size by pipelining and spatially parallel processing with input data passing through the array. However, the external memory access can also be a

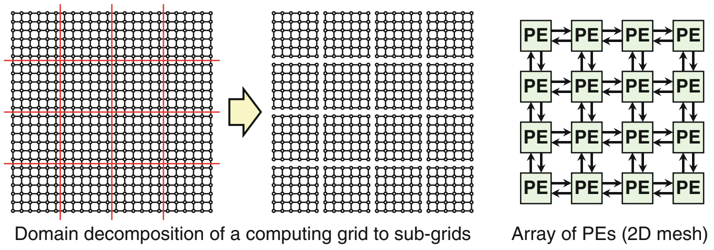


Fig. 3 Grid decomposition into sub-grids and their assignment to a PE array

bottleneck of the performance improvement if the memory bandwidth is insufficient in comparison with the bandwidth required by the array.

To avoid this limitation of the external memory bandwidth, we adopted the computational memory approach. This approach is based on the "processing in memory" concept, where computing logic and memory are arranged very close to each other [7, 17, 18, 29]. In the SCM architecture, the entire array behaves as *a computing memory*, which not only stores data but also performs computations with them. The memory of the array is formed by the local memories distributed to the PEs. Since PEs can concurrently access their local memories and perform computations, the SCM architecture has a computing performance scalable to the array size without any bottlenecks related to external memories.

The SCM architecture is actually a 2D systolic array of PEs connected by a mesh network. To exploit the parallelism and locality of stencil computation, we decompose the entire grid into sub-grids, and assign them to the PEs, as shown in Fig. 3. Since all the grid data are loaded onto the local memories, the PEs can perform stencil computations of their assigned sub-grids in parallel, exchanging the boundary data between adjacent PEs.

4 Design of SCM Array

4.1 2D Array of Processing Elements

Figure 4 shows an overview of the designed SCM array, which is a 2D array of PEs with local memories. The PEs are connected by a 2D mesh network via communication FIFOs. Each PE has a floating-point multiply-and-accumulate (FMAC) unit for the neighboring accumulation with data read from the local memory or the FIFOs. The result can be written to the memory and/or sent to the FIFOs of the adjacent PEs. The four communication FIFOs, $\{S,N,E,W\}$*-FIFOs*, hold the data transferred from the four adjacent PEs until they are read.

Figure 5 shows the data-path of a PE, which is pipelined with the following eight stages: *the instruction sequencing* (IS), *the memory read* (MR), *the five execution*

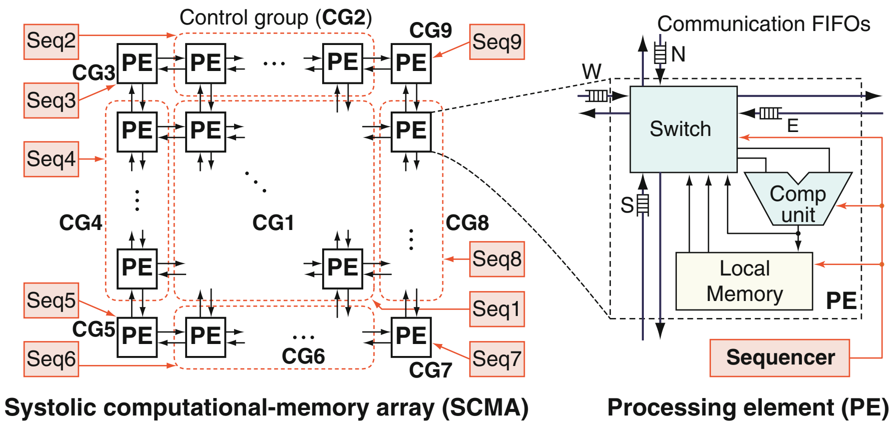


Fig. 4 SCM array of PEs and sequencers for control groups

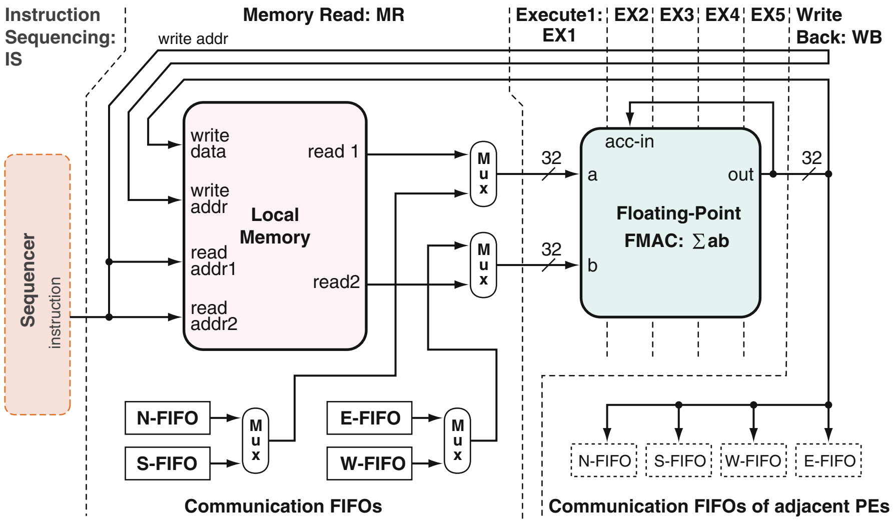


Fig. 5 Pipelined data-path of a PE

stages of the FMAC unit (EX1 to EX5), and *the memory write-back* (WB) [22]. In the MR stage, two values are selected from the read data of the local memory, and the data outputted by the communication FIFOs. The selected values are inputted to the FMAC unit. The FMAC unit can sequentially accumulate an arbitrary number of the products of inputs, which are IEEE754 single-precision floating-point numbers. Since the FMAC unit has the forwarding path from the EX5 stage to the EX2 stage for accumulation, it can sum up the products of inputs fed every three cycles. This means that three sets of (1) are required to fully utilize the FMAC unit.

4.2 Sequencers and Instruction Set

As shown in Fig. 4, the SCM array has sequencers, which give programmability to the array. In scientific simulation with a computational grid, the PEs taking charge of the inner grid-points regularly perform the same computation while the PEs for the boundary grid-points have to execute different computations. In order to achieve such diversity of control, we partition the PEs into several control groups (CGs), and assign a sequencer to each of them. Figure 4 shows a typical example of CGs partitioning. Here the array has the nine CGs: CG1, CG2, ..., CG9, which are given the PEs of the top, left, bottom, and right sides; the four corners; and the inner grid-points. The sequencer of each CG reads instructions from its own sequence memory and sends control signals to all the PEs of the CG based on the instructions. That is, the PEs of each CG are controlled in single instruction-stream and multiple data-stream (SIMD) fashion.

Table 1 shows an instruction set of the SCM array. Instructions are classified into two major groups: *control instructions* and *execution instructions*. Each control instruction supports the nested loop-control, no operation (nop), and halt. Nested-loops are executed with the lset and bne operations. The lset initiates the next-level loop by setting the number of iterations and the starting address of the loop body. After the lset is executed, the loop counter is decremented and the branch to the starting address is taken if it is not zero when the bne is executed. Each execution instruction has an opcode to select operation of the FMAC, two operands to specify the sources of the FMAC inputs, and two operands to specify the destinations of the FMAC output for the local memory and the communication FIFOs. Please note that an execution instruction can be merged with a bne instruction.

The code of Fig. 6 is an example of a sequence for three sets of (1) for grid-points (0,0), (1,0), and (2,0), where all the constants are 0.25. The grid is decomposed into 3×2 sub-grids as shown in Fig. 7. The PE computing the grid points communicates with the east, west, and north PEs through the E-, W-, and N-FIFOs, respectively. Because of the three-cycle forwarding of the FMAC unit, we concurrently perform the three sets of accumulations every three instructions. The lset instruction is used to repeat the computation for 1,600 times. Note that the branch is actually performed after the next instruction to the bne is executed. The code written in the assembly language is assembled and converted to the sequence binary for sequencers.

4.3 Techniques for Multiple-FPGA Implementation

For performance scalability, we design the SCM array to be scalably implemented over multiple FPGAs, by partitioning the array into sub-arrays. We can build a larger SCM array with more FPGAs. Figure 8 shows sub-arrays implemented over FPGAs A, B, and C.

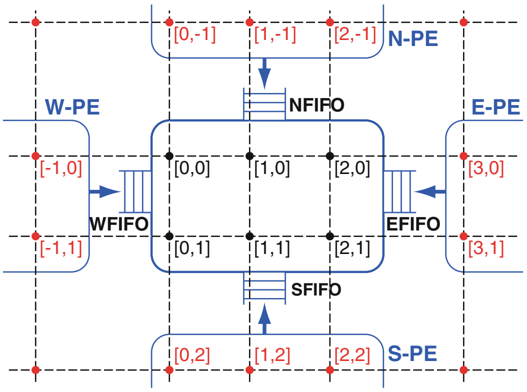
Table 1 Instruction set of the SCM array (in an assembly language of the SCM array)

Type	opcode	dst1,	dst2,	src1,	src2	Description
Control instructions	nop					No operation
	halt					Halt
	lset	ITERS,	ADDR			Loop-counter$_i$:= ITERS Bne-reg$_i$:= ADDR (for the i-th nested loop)
	bne					Branch to Bne-reg$_i$ if Loop-counter$_i$ is not eq. to zero.
Execution instructions	mulp	-,	L1,	L2,	EFIFO	FMACout = M[L2] $\times$ E-FIFO, M[L1] := FMAC-out
	accp	SE,	L1,	L2,	L3	FMAC-out = FMAC-out + M[L2] $\times$ M[L3], {S,E}-FIFOs := FMAC-out, M[L1] := FMAC-out
Merged	accpbne	-,	L1,	L2,	L3	accp & bne

```
QUAD  =        0.25            // LABEL definition
      lset     1600,        LOOP
LOOP: mulp          ,            ,     v[0;0],     QUAD
      mulp          ,            ,     v[1;0],     QUAD
      mulp          ,            ,     v[2;0],     QUAD
      accp          ,            ,     v[1;0],     QUAD
      accp          ,            ,     v[2;0],     QUAD
      accp          ,            ,      EFIFO,     QUAD
      accp          ,            ,      WFIFO,     QUAD
      accp          ,            ,     v[0;0],     QUAD
      accp          ,            ,     v[1;0],     QUAD
      accp          ,            ,      NFIFO,     QUAD
      accp          ,            ,      NFIFO,     QUAD
      accp          ,            ,      NFIFO,     QUAD
      accp        SW,   vnew[0;0],     v[0;1],     QUAD
      accpbne      S,   vnew[1;0],     v[1;1],     QUAD
      accp        SE,   vnew[2;0],     v[2;1],     QUAD
```

Fig. 6 Example code of (1) for (0,0), (1,0) and (2,0) on the sub-gird of Fig. 7. QUAD and “v[0;0]” are labels for local-memory addresses

Fig. 7 Assignment of 3 × 2 sub-grids for Jacobi computation with PEs



In multi-FPGA implementation, the more devices the system has, the more difficult it is to uniformly distribute a single clock source to them. To avoid distributing a single clock to all devices in a large system, we introduce *a globally asynchronous and locally synchronous (GALS) design* [13], where each FPGA is given an independent clock-domain. We transfer data between different clock-domains without meta-stability by using dual-clock FIFOs (DcFIFOs), which have different clock sources for input and output. Multiple clock-domains allow us to easily build an extensible and scalable system with many FPGAs.

4.3.1 Peak-Bandwidth Reduction Mechanism (BRM)

From a programming point of view, we should allow the sub-arrays over multiple FPGAs to logically operate as a single array. However, multiple-FPGA implementation presents several problems related to the off-chip bandwidth and

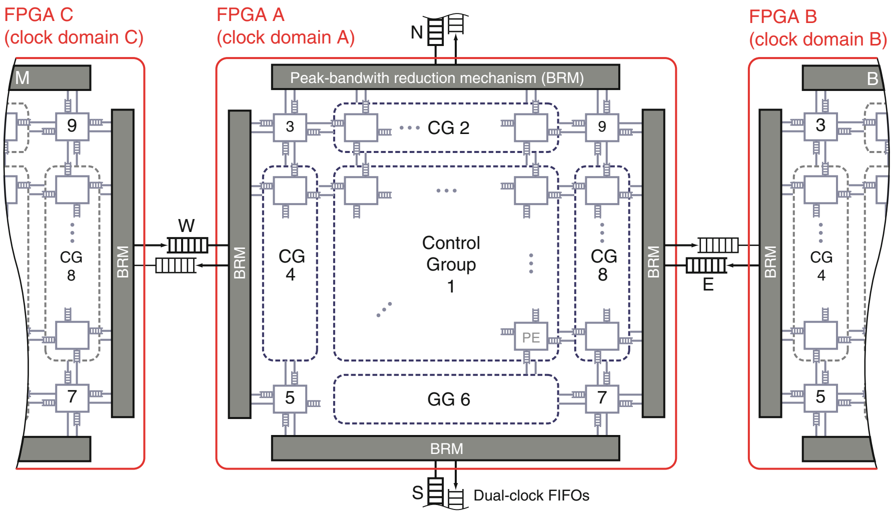


Fig. 8 SCM sub-arrays implemented over FPGAs A and B with different clock-domains. The clock-domains are bridged by the dual-clock FIFOs

synchronization, which are not seen in the case of single-FPGA implementation. In the SCM array implemented with a single FPGA, the adjacent PEs are directly connected and therefore the PE can send data to the adjacent PEs every cycle. The abundant on-chip wiring-resources make such direct connection available among PEs. On the other hand, the off-chip I/O bandwidth is much limited compared to the internal wires, and therefore it is more difficult for all the PEs to be directly connected between different FPGAs.

Fortunately, the data transfer between PEs is less frequent than read/write of the local memory, and we can utilize this characteristic to solve the off-chip bandwidth problem [22]. When a PE sends the result of computing (1), the data-transfer occurs only after the cycles necessary for FMAC unit to accumulate the terms. For example, the code of Fig. 6 has only the three instructions to send data in the fifteen instructions of the loop body. In addition, only the border grid-points of a sub-grid cause data transfer between adjacent PEs. Thus, actual stencil computation requires much less net-bandwidth than the peak bandwidth for sending data every cycle. Accordingly, the inter-FPGA bandwidth does not have to be as high as the aggregate memory-bandwidth of the border PEs in the SCM array.

However we still need a technique to handle the successive data-transfers conducted every cycle, which locally request the peak bandwidth, even if the inter-FPGA bandwidth is more than the average data-transfer rate. We designed a module with a peak-bandwidth reduction mechanism (BRM) to buffer the successive data-transfer requests, which is fully described in [22]. BRM is based on time-division multiplexing (TDM) for multi-cycle data-transfer with a buffer FIFO. For explanation, assume that the inter-FPGA bandwidth has the one n-th of the aggregate memory bandwidth of the border PEs. In this case, a set of words sent by the border PEs at a cycle is buffered by BRM, and takes n cycles to arrive at the adjacent FPGA. We refer to such BRM as "$n : 1$ BRM," which reduces the peak bandwidth to the one n-th with an n-times longer delay. If BRM has a buffer with a sufficient size for successive data-transfer, it can average out the net bandwidth. Thus BRM has a trade-off between the peak-bandwidth reduction and the delay increase. Therefore, we have to additionally take care of the increased delay in scheduling instructions to send data and use the received data.

4.3.2 Local-and-Global Stall Mechanism

In addition to the off-chip bandwidth problem, multi-FPGA implementation causes a synchronization problem of execution and data-transfer among devices because of the slight but inevitable difference in frequency among different clock domains even if their sources have the same configuration of frequency. Assume that we have two clock oscillators for 100 MHz. They can be different from each other, for example, 100.001 and 100.002 MHz. Since different frequencies cause the PEs to execute at different speeds, we need to synchronize the instructions executed by the PEs to send and receive data. The inter-PE communication is performed by explicitly executing instructions for communication, which send data to the adjacent PEs and read data from the communication FIFOs. If all the PEs are synchronized to a single clock, we can statically schedule instructions for adequate communication. However, if we use different clocks, PEs operating at a higher frequency could read an empty FIFO before the corresponding datum is written to the FIFO, and/or write a datum to a full FIFO before the FIFO is read by another PE. To avoid these data-transfer problems, we need some hardware mechanism to suspend the PEs in the case of a read-empty (RE) or write-full (WF).

We can easily think of the simplest design of the SCM array where all the sequencers and PEs simultaneously stall just after RE or WF is detected. However, this design is impractical. Although each sequencer can stall immediately when it locally detects RE or WF, the stall signal takes more than one cycle to be distributed from the sequencer to the others and make them stall. This is because the stall-signal distribution requires at least one OR operation for RE and WF detection, long wires with large fan-out, and another OR operation to generate the global-stall signal. If we implement the stall-signal distribution within one cycle, it reduces the operating frequency.

To solve this problem, we introduce *a LGSM* to the CGs, which is based on the very simple concept that sequencers stall immediately when they detect RE or WF, and the others stall several cycles later. Figure 9 shows an overview of the LGSM. Sequencer 2 of CG 2 observes $w.\mathrm{N}$ and $f.\mathrm{S}$ to detect RE and WF, which are the write signal of N(north)-FIFOs of its own PEs and the almost-full signal of S(south)-FIFOs of the PEs in the different clock-domain, B, respectively. With $w.\mathrm{N}$, the sequencer counts the number of remaining data in the N-FIFOs for the issued read-operation at present. When the sequencer issues a read operation for an

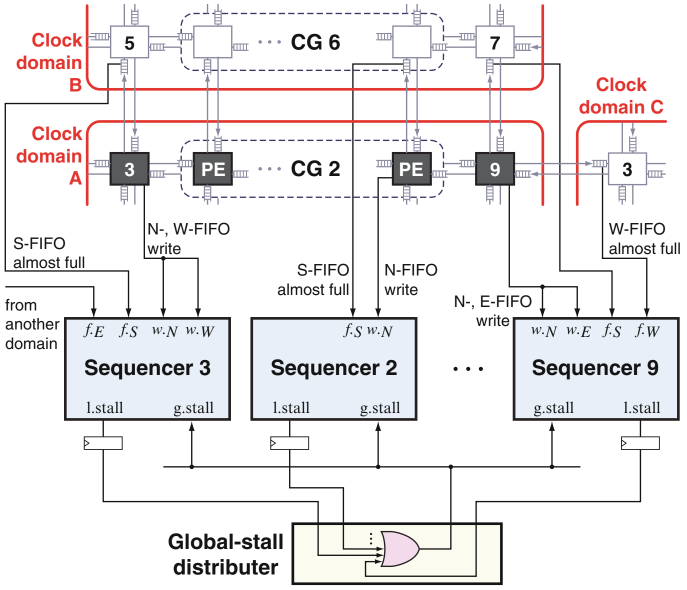


Fig. 9 Local-and-global stall mechanism (LGSM) to guarantee data-transfer synchronization in the GALS design

empty FIFO or a write operation for an almost-full FIFO, the sequencer immediately becomes the stall state. Then it sends a local-stall (l.stall) signal to *the global-stall distributor* (GSD).

The l.stall signal is latched at the output of each sequencer and reaches GSD at the next cycle. GSD has the inputs of the l.stall signals from all the sequencers. In GSD, the OR operation with all the l.stall signals generates a global-stall (g.stall) signal, which is distributed to all the sequencers. If necessary, we can insert additional latches into the distribution tree of the g.stall signal to prevent the operating frequency from decreasing. Sequencers in the execution state stall immediately when they receive the g.stall signal. Note that the sequencers with delayed stall issue one more instruction than Sequencer 2 because it locally stalls prior to the global stall. Therefore, Sequencer 2 issues the instruction before the other sequencers resume execution. This function is provided by a stall-control unit in each sequencer. The details of LGSM are described in [13].

5 Implementation and Evaluation

5.1 Implementation

With implementation of SCM arrays on multiple-FPGAs, we demonstrate that SCM arrays have scalability according to the number of FPGAs. We also evaluate the

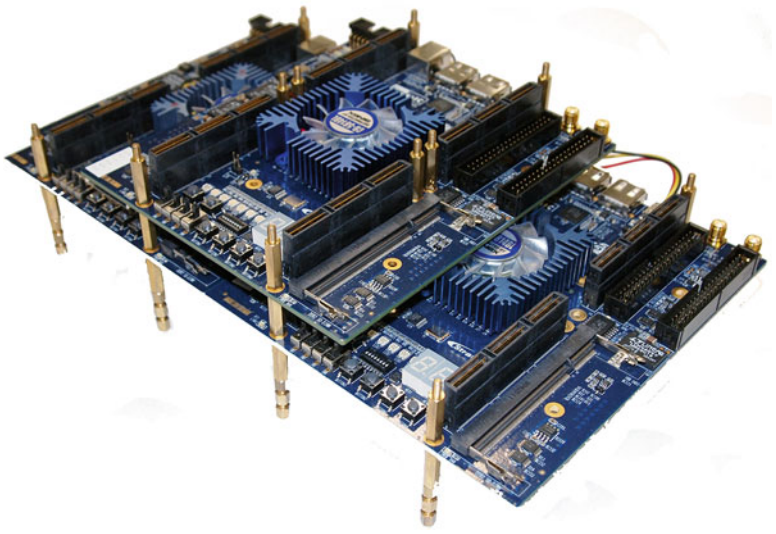


Fig. 10 1×3 FPGA-array of 3 DE3 board

overhead caused by LGSM for its frequency degradation and resource consumption by comparing 3×3, 4×4, and 8×8 SCM arrays with and without LGSM. We use Terasic DE3 boards [28] to construct the 1×3 FPGA-array of Fig. 10, while our final goal is to build a 2D FPGA-array as shown in Fig. 11.

Figure 12 shows the block diagram of the DE3 board. The DE3 board has an ALTERA Stratix III EP3SL150 FPGA [1] and four HSTC connectors, which can be used to connect DE3 boards to each other with a single-end or LVDS signaling. Each HSTC connector has 130 I/O pins including clock lines. Each pair of the three DE3 boards is connected by the two HSTC connectors for bi-directional data-transfer. By using 64 pins of the 130 pins at 100 MHz in single-end, each HSTC connector provides a uni-directional data-transfer of 0.8 GB/s between FPGAs. The 4:1 BRM allows at most 8×8 PEs to send or receive data between FPGAs with this connection. The Stratix III EP3S150 FPGA has 113,600 ALUTs (adaptive LUTs), which is equivalent to 142,000 LEs, block RAMs with a total of 6,390 Kbits, and 96 36-bit DSP blocks. The block RAMs consist of 355 M9K blocks and 16 M144K blocks. The size of each M9K block is 9 Kbits while each M144K has 144 Kbits.

Figure 13 shows a block diagram of a system implemented on each FPGA. We wrote verilog-HDL codes of 3×3, 4×4, and 8×8 SCM arrays with or without LGSM. We implement the 4:1 BRMs and the distributors to reduce the peak-bandwidth requirement to one-fourth for inter-FPGA connection. We use DcFIFOs to connect the logics in different clock-domains. We also implement ALTERA's system on programmable chip (SOPC) with an NIOS II processor for the USB interface. The host PC can read and write data and commands to the SCM array's

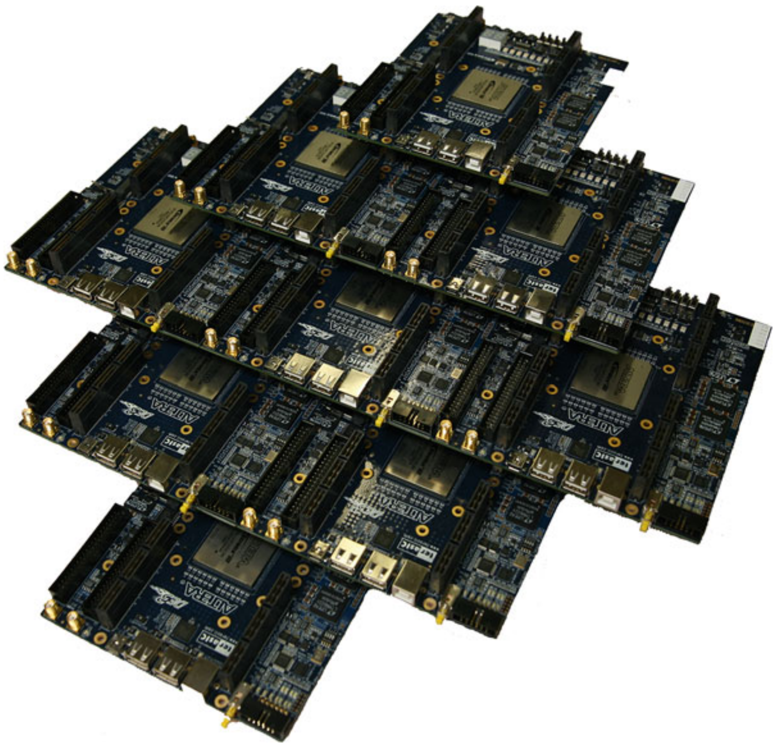


Fig. 11 3 × 3 FPGA-array of 9 DE3 boards, which is planed to be implemented

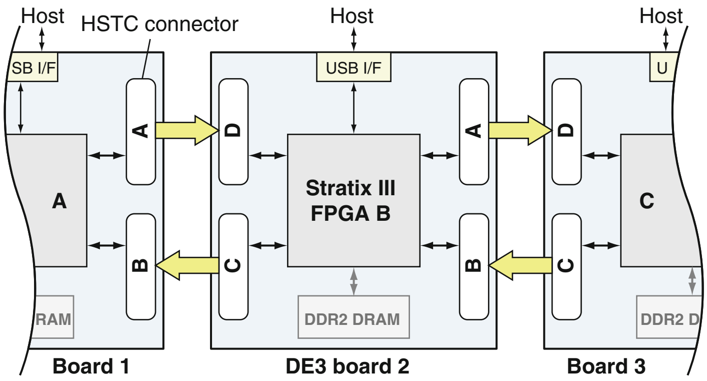


Fig. 12 Block diagram of DE3 board and a 1 × 3 FPGA array

memories via USB. We compiled the system by using ALTERA Quartus II compiler version 9.1 with the options of "area," "standard fit," and "incremental-compilation off."

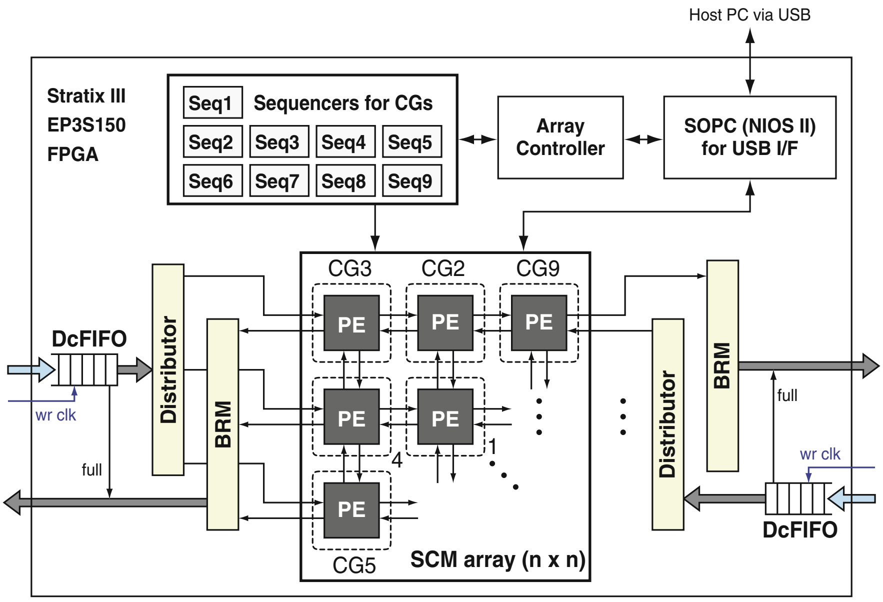


Fig. 13 Block diagram of a system including a sub-SCM-array implemented on an FPGA

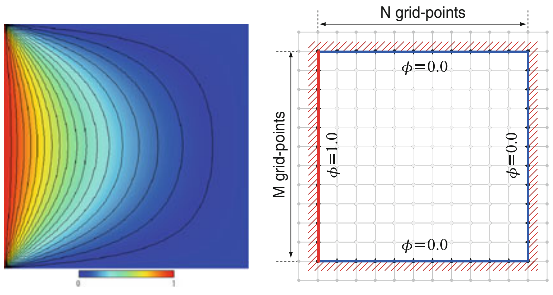
5.2 Benchmark Computations

PEs perform floating-point operations for (1), and therefore the prototyped system can actually compute real applications based on the finite difference method. We program the PEs of the SCM array in the dedicated assembler language [22]. For benchmarks, we use the applications summarized in Fig. 14. The red-black successive over-relaxation method [9], *RB-SOR*, is one of the parallelized iterative solvers for Poisson's equation or Laplace's equation. In the RB-SOR method, the grid points are treated as a checkerboard with red and black points, and each iteration is split into a *red*-step and a *black*-step. The red- and black-steps compute the red points and the black points, respectively. We solved the heat-conduction problem on a 2D square plate with RB-SOR. Each PE computes with an 8×24 sub-grid so that the 9×3 SCM array on the three FPGAs computes a 72×72 grid for 2.0×10^6 iterations.

The fractional-method [27], *FRAC*, is a typical and widely used numerical method for computing incompressible viscous flows by numerically solving the Navier–Stokes equations. We simulated the 2D square driven cavity flow with the FRAC, giving the result shown in Fig. 14. The left, right, and lower walls of the square cavity are stable, and only the upper surface is moving to the right with a velocity of $u = 1.0$. Each PE takes charge of a 5×15 sub-grid. For the 9×3 SCM array on the three FPGAs, we compute 5,000 time-steps with a 45×45 grid while the Jacobi computation is performed for 1,000 iterations at each time-step.

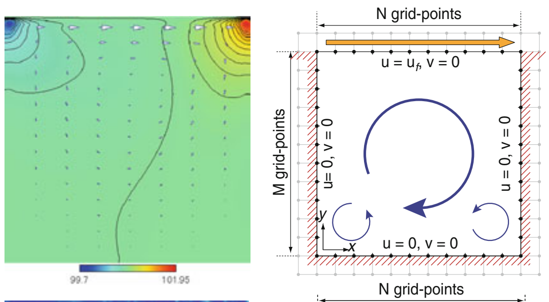
Red-black SOR (RB-SOR)

Numerical solver of Laplace's equation: $\nabla^2\phi = 0$. The 2D time-independent heat-conduction is computed with a 2D grid by using the red-black SOR method. The red-black SOR is a parallelized version of the SOR (Successive Over-Relaxation) method, which is computed with a checkerboard-like grid of red and black points. Computation of each iteration is divided into a red step and a black step, which are independent to each other.



Fractional-step method (FRAC)

Numerical method to compute incompressible viscous flow. 2D square driven cavity flow is computed giving a time-independent result. The fractional-step method is composed of calculating tentative velocities, solving the Poisson's equation of pressure, and calculating the true velocities. We use Jacobi method to solve the Poisson's equation of pressure.



FDTD method (FDTD)

Numerical method to solve the Maxwell's equations for electromagnetic problems. The 2D propagation of electromagnetic waves is computed with a 2D grid. The square-wave source was placed at the left-bottom corner of the grid. We use the Mur's first-order absorbing boundary condition for the border of the grid.

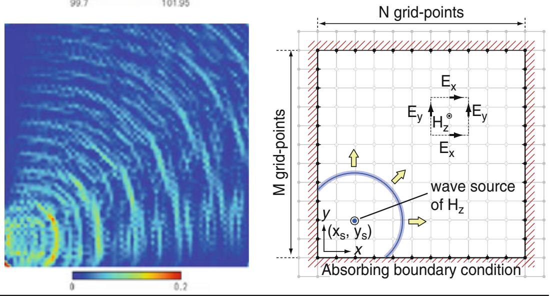


Fig. 14 Benchmark computations

The FDTD method [31], is a powerful and widely used tool to solve a wide variety of electro-magnetic problems, which provides a direct time-domain solution for Maxwell's Equations discretized by difference schemes on a uniform grid and at time intervals. Since the FDTD method is very flexible and gives accurate results for many non-specific problems, it is widely used for solving a wide variety of electro-magnetic problems. We compute the FDTD method to simulate 2D propagation of electro-magnetic waves with a square-wave source. At the left-bottom corner, we put the square-wave source with an amplitude of 1 and a period of 80 time-steps. On the border, Mur's first-order absorbing boundary condition is applied. Each PE takes charge of a 4×6 sub-grid. For the 9×3 SCM array on the three FPGAs, we compute 1.6×10^6 time-steps with a 36×18 grid.

Table 2 Synthesis results for Stratix III FPGA

		Without LGSM			With LGSM		
Array size		3×3	4×4	8×8	3×3	4×4	8×8
$f_{\max}$[MHz]		128	123	123	128	125	118
ALUTs	SCMA	12,248 (10.8%)	20,984 (18.5%)	78,189 (68.8%)	12,420 (10.9%)	21,179 (18.6%)	78,341 (69.0%)
	BRM	881 (0.776%)	1,123 (0.989%)	2,274 (2.00%)	878 (0.773%)	1,128 (0.993%)	2,260 (1.99%)
	Others	10,041 (8.84%)	7,743 (6.82%)	6,133 (5.40%)	9,080 (7.99%)	10,784 (9.49%)	6,133 (5.40%)
36-bit DSP blocks		9 (9.38%)	16 (16.7%)	64 (66.7%)	9 (9.38%)	16 (16.7%)	64 (66.7%)
Total memory bits		1,842,752 (32.7%)	2,072,128 (36.8%)	3,645,592 (64.7%)	1,842,752 (32.7%)	2,072,128 (36.8%)	3,645,592 (64.7%)

5.3 Synthesis Results

Table 2 shows the synthesis results of SCM arrays with and without LGSM for 3×3, 4×4, and 8×8 PEs. The larger the SCM array, the lower the frequency that is available. This is because of the longer critical paths in the larger array. However, the 8×8 array can still operate at more than 118 MHz, which is sufficiently higher than 100 MHz. The 8×8 array consumes about 75% of the ALUTs, 67% of the 36-bit DSP blocks, and 65% of the total memory bits. Based on these data, we estimate that we can implement up to 100 PEs on this FPGA to form a 10×10 array, giving 1.5 times higher performance than that of the 8×8 array.

LGSM slightly decreases the maximum operating frequency, $f_{\max}$, by only a few MHz. The difference in frequency of the 8×8 array between SCM arrays with and without LGSM is only 5 MHz. This means that LGSM does not have a major impact on the scalability to the array size. There is almost no performance degradation caused by introducing LGSM. LGSM slightly increases resource consumption by 0.17% at most for the ALUTs of the SCM array and do not increase the total memory bits. Note that the ratio of resources increased by LGSM is almost constant regardless of the array size. This is because the number of sequencers, 9, is unchanged. These results show that we can use LGSM at very low cost in terms of resources.

5.4 Performance Results

Although we set 100 MHz to all the clocks of the FPGAs, they can be slightly different because they are generated by different clock sources. The stall cycles caused by this difference may considerably increase the total cycles of computation

Table 3 Execution and stall cycles, performance and utilizations for 1, 2 and 3 FPGAs

Computation		1 FPGA	2 FPGAs	3 FPGAs
RB-SOR	Exec cycles	1,440,003,040	1,440,003,040	1,440,003,040
	Stall cycles	0 (0.0%)	11,207 (0.00078%)	11,250 (0.00078%)
	GFlop/s	11.4	22.9 ($\times$2.01)	34.4 ($\times$3.02)
	Utilization	89.2%	89.5%	89.7%
FRAC	Exec cycles	977,382,044	977,382,044	977,382,044
	Stall cycles	0 (0.0%)	7,592 (0.00078%)	7,591 (0.00078%)
	GFlop/s	11.2	22.4 ($\times$2.01)	33.7 ($\times$3.02)
	Utilization	87.2%	87.6%	87.7%
FDTD	Exec cycles	950,432,027	950,432,027	950,432,027
	Stall cycles	0 (0.0%)	7,220 (0.00076%)	7,212 (0.00076%)
	GFlop/s	10.2	20.6 ($\times$2.02)	30.9 ($\times$3.04)
	Utilization	79.6%	80.4%	80.6%

and spoil the scalability by using multiple devices. To evaluate the stall cycles of SCM arrays operating on multiple FPGAs, we implement the 8×8 array with LGSM on each FPGA and execute the benchmark programs of RB-SOR, FRAC, and FDTD with SCM arrays on a single FPGA, two FPGAs and three FPGAs connected as a 1D array. For the execution, we gave the constant size of the sub-grid computed by each PE so that the size of the entire grid is proportional to the number of FPGAs. Therefore, the SCM arrays on a different number of FPGAs require the same cycles for computation while larger SCM arrays provide higher performance with larger grids computed by more PEs.

Table 3 shows the numbers of execution cycles and stall cycles for the SCM arrays on the single-, double-, and triple-FPGA arrays. Since the SCM array on a single FPGA operates with a single clock-domain, no stall cycle is observed for all the benchmark computations on the single-FPGA SCM array. On the other hand, the double-FPGA and triple-FPGA SCM arrays have stall cycles due to the difference in frequency among clock domains, which cause a slight increase in the total number of cycles. However, the ratios of the stall cycles to the total execution cycles are very small and ignorable and are about 8×10^{-4}%. With these results, we made sure that the clock frequencies of 100 MHz have small but inevitable differences, and that the stall mechanism works well to guarantee the data synchronization with slight loss of performance.

Table 3 also shows the actual floating-point performance for each benchmark computation in GFlop/s. We used the measured cycles including the execution and stall cycles to obtain the performance. We calculated the utilization that is defined as the actual performance divided by the peak performance. The results show that high utilization of 80–90% is achieved for the benchmark computations executed on the SCM arrays irrespective of the number of FPGAs. Note that almost the same utilization is maintained in increasing the number of FPGAs. The utilization is slightly improving rather than declining because the ratio of border grid-points is reduced.

These results show that the SCM array provides complete scalability to the array size and the number of devices, so that m FPGAs achieve m times higher performance to compute an m times larger grid. If we implement a 10×10 sub-array on each FPGA, the three FPGAs are expected to give 53.8 GFlop/s to RB-SOR. Note that the total bandwidth of the local memories is also completely proportional to the size of the SCM array. The single-FPGA, double-FPGA, and triple-FPGA SCM arrays have the internal bandwidth to read and write local memories of 76.8, 153.6, and 230.4 GByte/s at 100 MHz, respectively. If we implement a 10×10 sub-array on each FPGA instead of a 8×8 sub-array, they could provide 120, 240, and 360 GByte/s, respectively.

5.5 Feasibility and Performance Estimation for State-of-the-Art FPGAs

Here we discuss the feasibility of implementation with 2D FPGA arrays and estimate their peak performance for high-end FPGA series. We consider the three high-end ALTERA FPGAs: Stratix III EP3SL340 FPGA (65 nm), Stratix IV EP4SGX530 FPGA (40 nm), and Stratix V 5SGSD8 FPGA [1]. The resources of these FPGAs are summarized in Table 4. We obtain the total I/O bandwidth of each FPGA by multiplying the number of transceivers and the bandwidth per transceiver. We assume that implementation optimization allows PEs to operate at 125 MHz on these FPGAs. Since an FMAC requires one single-precision floating-point multiplier to be implemented with one 36-bit or 27-bit DSP block, we assume that the number of PEs available on each FPGA is the same as the number of 36-bit or 27-bit DSP blocks. Each PE can perform both multiplication and addition at 125 MHz. Therefore, the peak performance of each FPGA is obtained by $0.25 \times$ (the number of PEs) [GFlop/s].

As shown in Table 4, while Stratix III and IV FPGAs have a moderate peak-performance, Stratix V FPGA has a peak performance of 491 GFlop/s per chip. This is due to integration of many DSP blocks on state-of-the-art FPGA. Since an SCM array performs stencil computation with a utilization of about 85%, the Strativ V FPGA is expected to achieve a sustained performance of $491 \times 0.85 = 417$ GFlop/s per chip. This performance is higher than the sustained performance of stencil computation on a single GPU. Furthermore, due to the complete scalability of an SCM array, we can efficiently scale the performance by using multiple FPGAs. For example, we can obtain a sustained performance of 41.7 TFlop/s with 100 Stratix V FPGAs. The cluster of 100 FPGAs can be cheaper than a larger GPU cluster to provide comparable performance with much less utilization.

For feasibility, we should discuss whether the I/O bandwidth is sufficient to connect the FPGAs with a 2D mesh network because a 2D FPGA array is suitable to scale a 2D SCM array. We also assume an $N \times N$ square array of PEs where $N = \sqrt{\text{(the number of PEs)}}$ on each FPGA, the required bandwidth for four links

Table 4 Estimation of array size and available bandwidth for a 2D array of high-end FPGAs

		Stratix III L EP3SL340	Stratix IV GX EP4SGX530	Stratix V 5SGSD8
Technology		64 nm	40 nm	28 nm
Equivalent LEs		337,500	531,200	695,000
Memory [KBytes]		2,034	2,592	6,417
36/27-bit DSPs		144	256	1,963
Transceivers (Gbps/ch)		132 (0.156)	48 (8.5)	48 (1.41)
Total I/O bandwidth [GB/s]		**20.6**	**51**	**84.6**
Assumed PE freq. [MHz]		125	125	125
Estimated # of PEs		144	256	1,963
Peak GFlop/s per FPGA		36	64	491
Required uni-directional BW for 4 links [GB/s]		**24.0**	**32**	**88.6**
Reduced BW by $n:1$ BRM [GB/s]	$n=2$	**12.0**	**16.0**	**44.3**
	$n=4$	**6.0**	**8.0**	**22.2**
	$n=8$	**3.0**	**4.0**	**11.1**

(Peak GFlop/s) = (# of PEs) $\times$ 2 operations $\times$ 0.125 GHz)
(Req. unidir BW for 4 links) = 4 $\times$ $\sqrt{(\# \text{ of PEs})}$ $\times$ 4 words $\times$ 0.125 GHz

of a 2D mesh network is calculated by $4 \times N \times 4 \times 0.125$ GByte/s. In Table 4, the required bandwidth for four links is slightly higher than the available I/O bandwidth for Stratix III and V FPGAs. However, BRM reduces the required bandwidth less than the I/O bandwidth. Especially, much less bandwidth is required if we use 4:1 BRM. These estimations show that implementation of an SCM array with multiple FPGAs is feasible and we can build an array of a large number of FPGAs without inter-FPGA communication bottleneck.

6 Summary

This chapter presents the SCM array for programmable stencil computations. The target computation of an SCM array is the neighboring accumulation for 3×3 star-stencil computations. The SCM array is based on the SCM architecture, which combines the systolic array and the computational memory approach to scale both computing performance and aggregate memory-bandwidth in accordance with the array size. After the structure and behavior of processing elements and their sequencers are described, we show GALS implementation for multiple FPGAs with different clock-domains. We also present two techniques, the peak-bandwidth reduction mechanism (BRM) and the LGSM, which are necessary to solve the bandwidth and synchronization problems of inter-FPGA data-transfer.

We implement a prototype SCMA with three Stratix III FPGAs. We demonstrate that the prototyped SCMAs compute the three benchmark problems: RB-SOR, FRAC, and FDTD. By implementing 3×3, 4×4, and 8×8 SCMAs, we evaluate their impact on the synthesis results including operating frequency and resource consumption. We also evaluate the overhead of LGSM in terms of operating frequency and resource consumption. We show that the size of the array and LGSM have a slight influence on the operating frequency, but only frequency degradation is limited. Thus our design of an SCMA with LGSM is scalable for available resources on an FPGA.

To evaluate performance scalability for multiple FPGAs, we compare single-FPGA, double-FPGA, and triple-FPGA SCMA, where each sub-SCMA has an 8×8 array. The number of FPGAs gives complete scalability of sustained performance maintaining a utilization of 80–90% for each benchmark. As a result, the three FPGAs operating at 100 MHz achieve 31–34 GFlop/s for single-precision floating-point computations. We expect that a 10×10 sub-SCMA can be implemented on each FPGA to provide 1.5 times higher performance than that of the 8×8 sub-SCMA. We ensure the LGSM provides the necessary stalls for the differences in frequency of the 100 MHz clocks; however, the number of stall cycles is very small, just $8 \times 10^{-4}\%$ of the total cycles. This means that sub-SCMAs in different clock-domains are synchronized by LGSM, but the overhead is ignorable for computing performance.

A feasibility study of implementation with the high-end FPGA series showed that the Stratix V FPGA is expected to achieve a peak performance of 419 GFlop/s and a sustained performance of about 417 GFlop/s. We showed that BRM allows a large SCM array to be implemented with many FPGAs without a bottleneck in inter-FPGA communication.

In future work, we will construct a 2D FPGA array and implement a large SCMA on it for larger 3D computations. We will also develop a compiler for SCM arrays based on the prototype version of a compiler for a domain-specific language [14].

Acknowledgements This research and development were supported by Grant-in-Aid for Young Scientists(B) No. 20700040, Grant-in-Aid for Scientific Research (B) No. 23300012, and Grant-in-Aid for Challenging Exploratory Research No. 23650021 from the Ministry of Education, Culture, Sports, Science and Technology, Japan.

References

1. Altera Corporation (2012), http://www.altera.com/literature/
2. R. Baxter, S. Booth, M. Bull, G. Cawood, J. Perry, M. Parsons, A. Simpson, A. Trew, A. McCormick, G. Smart, R. Smart, A. Cantle, R. Chamberlain, G. Genest, Maxwell a 64 FPGA supercomputer, in *Proceedings AHS2007 Conference Secound NASA/ESA Conference on Adaptive Hardware and Systems* (2007), pp. 287–294, http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4291933
3. W. Chen, P. Kosmas, M. Leeser, C. Rappaport, An fpga implementation of the two-dimensional finite-difference time-domain (FDTD) algorithm, in *Proceedings of the 2004 ACM/SIGDA*

12th International Symposium on Field Programmable Gate Arrays (FPGA2004) (2004), pp. 213–222, http://dl.acm.org/citation.cfm?id=968311

4. K. Datta, M. Murphy, V. Volkov, S. Williams, J. Carter, L. Oliker, D. Patterson, J. Shalf, K. Yelick, Stencil computation optimization and auto-tuning on state-of-the-art multicore architectures, in *Proceedings of the 2008 ACM/IEEE Conference on Supercomputing* (2008), pp. 1–12, http://dl.acm.org/citation.cfm?id=1413375
5. J.D. Davis, C.P. Thacker, C. Chang, BEE3: revitalizing computer architecture research. MSR-TR-2009-45 (Microsoft Research Redmond, WA, 2009)
6. J.P. Durbano, F.E. Ortiz, J.R. Humphrey, P.F. Curt, D.W. Prather, FPGA-based acceleration of the 3D finite-difference time-domain method, in *Proceedings of the 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines* (2004), pp. 156–163, http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1364626
7. D.G. Elliott, M. Stumm, W. Snelgrove, C. Cojocaru, R. Mckenzie, Computational RAM: implementing processors in memory. Des. Test Comput. **16**(1), 32–41 (1999)
8. A. George, H. Lam, G. Stitt, Novo-G: at the forefront of scalable reconfigurable supercomputing. Comput. Sci. Eng. **13**(1), 82–86 (2011)
9. L.A. Hageman, D.M. Young, *Applied Iterative Methods* (Academic, New York, 1981)
10. T. Hauser, A flow solver for a reconfigurable FPGA-based hypercomputer. AIAA Aerosp. Sci. Meet. Exhib. **AIAA-2005-1382** (2005)
11. K.T. Johnson, A. Hurson, B. Shirazi, General-purpose systolic arrays. Computer **26**(11), 20–31 (1993)
12. H.T. Kung, Why systolic architecture? Computer **15**(1), 37–46 (1982)
13. W. Luzhou, K. Sano, S. Yamamoto, Local-and-global stall mechanism for systolic computational-memory array on extensible multi-FPGA system, in *Proceedings of the International Conference on Field-Programmable Technology (FPT2010)* (2010), pp. 102–109, http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5681763
14. W. Luzhow, K. Sano, S. Yamamoto, Domain-specific language and compiler for stencil computation on FPGA-based systolic computational-memory array, in *Proceedings of the International Symposium on Applied Reconfigurable Computing (ARC2012)* Springer, (2012), pp. 26–39, http://link.springer.com/chapter/10.1007%2F978-3-642-28365-9_3?LI=true
15. O. Mencer, K.H. Tsoi, S. Craimer, T. Todman, W. Luk, M.Y. Wong, P.H.W. Leong, Cube: a 512-FPGA cluster, in *Proceedings of the IEEE Southern Programable Logic Conference 2009* (2009), pp. 51–57, http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4914907
16. H. Morishita, Y. Osana, N. Fujita, H. Amano, Exploiting memory hierarchy for a computational fluid dynamics accelerator on FPGAs, in *Proceedings of the International Conference on Field-Programmable Technology (FPT2008)* (2008), pp. 193–200, http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4762383
17. D. Patterson, T. Anderson, N. Cardwell, R. Fromm, K. Keeton, C. Kozyrakis, R. Thomas, K. Yelick, A case for intelligent RAM: IRAM. IEEE Micro **17**(2), 34–44 (1997)
18. D. Patterson, K. Asanovic, A. Brown, R. Fromm, J. Golbus, B. Gribstad, K. Keeton, C. Kozyrakis, D. Martin, S. Perissakis, R. Thomas, N. Treuhaft, K. Yelick, Intelligent RAM(IRAM): the industrial setting, applications, and architectures, in *Proceedings of the International Conference on Computer Design* (1997), pp. 2–9, http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=628842
19. E.H. Phillips, M. Fatica, Implementing the himeno benchmark with CUDA on GPU clusters, in *Proceedings of International Symposium on Parallel and Distributed Processing (IPDPS)* (2010), pp. 1–10, http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5470394
20. K. Sano, T. Iizuka, S. Yamamoto, Systolic architecture for computational fluid dynamics on FPGAs, in *Proceedings of the 15th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)* (2007), pp. 107–116, http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4297248
21. K. Sano, W. Luzhou, Y. Hatsuda, S. Yamamoto, Scalable FPGA-array for high-performance and power-efficient computation based on difference schemes, in *Proceedings of the International Workshop on High-Performance Reconfigurable Computing Technology and Applications (HPRCTA)* (2008), http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4745679

22. K. Sano, W. Luzhou, Y. Hatsuda, T. Iizuka, S. Yamamoto, FPGA-array with bandwidth-reduction mechanism for scalable and power-efficient numerical simulations based on finite difference methods. ACM Trans. Reconfigurable Technol. Syst. (TRETS) **3**(4), Article No. 21 (2010)
23. K. Sano, W. Luzhou, S. Yamamoto, Prototype implementation of array-processor extensible over multiple FPGAs for scalable stencil computation. ACM SIGARCH Computer Architecture News (HEART special issue), **38**(4), 80–86 (2010)
24. K. Sano, Y. Hatsuda, S. Yamamoto, Scalable streaming-array of simple soft-processors for stencil computations with constant memory-bandwidth, in *Proceedings of the 19th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)* (2011), pp. 234–241, http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5771279
25. R.N. Schneider, L.E. Turner, M.M. Okoniewski, Application of fpga technology to accelerate the finite-difference time-domain (FDTD) method, in *Proceedings of the 2002 ACM/SIGDA 10th International Symposium on Field Programmable Gate Arrays (FPGA2002)* (2002), pp. 97–105, http://dl.acm.org/citation.cfm?id=503063
26. W.D. Smith, A.R. Schnore, Towards an RCC-based accelerator for computational fluid dynamics applications. J. Supercomput. **30**(3), 239–261 (2003)
27. J.C. Strikwerda, Y.S. Lee, The accuracy of the fractional step method. SIAM J. Numer. Anal. **37**(1), 37–47 (1999)
28. TERASIC Corp. (2012), Accessed 30th January 2013, http://www.terasic.com.tw
29. J.E. Vuillemin, P. Bertin, D. Roncin, M. Shand, H.H. Touati, P. Boucard, Programmable active memories: reconfigurable systems come of age. IEEE Trans. Very Large Scale Integr. (VLSI) Syst. **4**(1), 56–69 (1996)
30. S. Williams, A. Waterman, D. Patterson, Roofline: an insightful visual performance model for multicore architectures. Comm. ACM **52**(4), 65–76 (2009)
31. K.S. Yee, Numerical solution of inital boundary value problems involving maxwell's equations in isotropic media. IEEE Trans. Antennas Propag. **14**, 302–307 (1966)