PAPER Special Section on Reconfigurable Systems

Scalability Analysis of Deeply Pipelined Tsunami Simulation with Multiple FPGAs*

Antoniette MONDIGO[†a)], ***Nonmember***, **Tomohiro UENO**[††b)], **Kentaro SANO**[††c)], ***and*** **Hiroyuki TAKIZAWA**[†d)], ***Members***

SUMMARY Since the hardware resource of a single FPGA is limited, one idea to scale the performance of FPGA-based HPC applications is to expand the design space with multiple FPGAs. This paper presents a scalable architecture of a deeply pipelined stream computing platform, where available parallelism and inter-FPGA link characteristics are investigated to achieve a scaled performance. For a practical exploration of this vast design space, a performance model is presented and verified with the evaluation of a tsunami simulation application implemented on Intel Arria 10 FPGAs. Finally, scalability analysis is performed, where speedup is achieved when increasing the computing pipeline over multiple FPGAs while maintaining the problem size of computation. Performance is scaled with multiple FPGAs; however, performance degradation occurs with insufficient available bandwidth and large pipeline overhead brought by inadequate data stream size. Tsunami simulation results show that the highest scaled performance for 8 cascaded Arria 10 FPGAs is achieved with a single pipeline of 5 stream processing elements (SPEs), which obtained a scaled performance of 2.5 TFlops and a parallel efficiency of 98%, indicating the strong scalability of the multi-FPGA stream computing platform.

key words: *tsunami simulation, stream computing, scalability, multiple FPGAs, high-performance computing*

1. Introduction

In the recent decades, field programmable gate arrays (FPGAs) have become consistently promising in the area of high-performance computing (HPC). Due to advancements of current FPGAs, which include support for high performance floating-point (FP) operations, availability of various macros, and a larger transistor density, they are becoming attractive solutions to accelerate HPC applications [3]–[6].

Despite FPGA's typical lower operating frequency range than GPUs' and CPUs', creating custom hardware allows massively parallel operations with high utilization rates. By constructing a pipeline with regular memory access, continuous data can be streamed from memory to computing units in the FPGA, which in effect, conceals latency. This makes stream computing with a data flow model suitable for low operational intensity applications such as stencil computing algorithms in FPGAs, which has been successfully demonstrated in [7]–[9]. However, the resource budget of a single FPGA limits further performance scaling.

To overcome this constraint, extending the stream computing pipeline with multiple FPGAs is promising. This paper presents the design and architecture of a stream computing platform, where custom computing units are cascaded over multiple FPGAs in a 1D ring topology. To efficiently utilize the available resources on multiple FPGAs, we rely on extending the pipeline with temporal and spatial parallelism [9]–[11], which introduces a vast design space.

Since 1D ring of FPGAs does not provide infinite scalability, the main goal of this work is to know its performance characteristics by performing scalability analysis. We present a performance model for multiple FPGAs, which considers parallelism options of the stream computing pipeline [9]. We also investigated the performance characteristics of inter-FPGA communication links, which interconnect FPGAs through their high-speed transceivers. In addition, we explored the extended design space, where we implemented a custom computing application on 8 cascaded FPGAs and verified the performance model. Preliminary portions of this paper were published in [1], [2].

To achieve the final goal, we performed a scalability analysis of the deeply pipelined FPGAs by evaluating speedup against its parallel efficiency. For benchmark purposes, we applied the proposed stream computing approach to a practical tsunami simulation with real ocean-depth data. The specific contributions are as follows:

1. A deeply pipelined hardware platform for multiple FPGAs with inter-FPGA communication subsystem;
2. A fine-grained scalable architecture of custom computing units for spatial and temporal parallelism;
3. Investigation of performance characteristics in the inter-FPGA communication links;
4. Performance model of the multi-FPGA approach; and
5. Implementation and performance evaluation of tsunami simulation using cascaded Intel Arria 10 FPGAs.

By implementing 5 stream processing elements (SPEs) to each of the 8 Arria 10 FPGAs, a measured sustained performance of 2.5 TFlops is achieved, where the estimated speedup is attained with a parallel efficiency of 98%. This suggests multiple FPGAs' potential to extend performance scalability.

Manuscript received July 27, 2018.
Manuscript revised December 4, 2018.
Manuscript publicized February 5, 2019.

[†]The authors are with the Graduate School of Information Sciences, Tohoku University, Sendai-shi, 980–8578 Japan.
[††]The authors are with the Processor Research Team, Riken Center for Computational Science, Kobe-shi, 650–0047 Japan.

*Preliminary portions of this paper appeared in [1], [2].

a) E-mail: amondigo@dc.tohoku.ac.jp
b) E-mail: tomohiro.ueno@riken.jp
c) E-mail: kentaro.sano@riken.jp (Corresponding author)
d) E-mail: takizawa@tohoku.ac.jp

DOI: 10.1587/transinf.2018RCP0007

This paper is organized as follows. Section 2 summarizes related work. Section 3 presents the design of multi-FPGA stream computing platform, parallelism options, inter-FPGA communication subsystem, and proposed performance model. Section 4 describes the prototype implementation with the evaluation and discussion of results. Finally, Sect. 5 gives the conclusions and future work.

2. Related Work

In FPGA-based custom computations, several approaches can be implemented to achieve high performance, such as latency hiding of independent functions and data streaming through pipelined operations [12]. Azarian and Cardoso [13] investigated the coarse/fine-grained dataflow synchronization approaches to achieve pipelining execution of the tasks in FPGA-based multicore architectures, in which results show a speedup in the overall execution through the use of multiple cores provided by FPGAs. Xilinx Virtex 5 FPGA was used with MicroBlaze soft microprocessors as cores. Since the intended custom computing units in this paper are constructed as a pipeline of hardware operations, then, it is expected to give a better scaled performance, through a more efficient FPGA resource utilization.

Murtaza et al. [14] demonstrated a streamed computation with Lattice Boltzmann method (LBM) application in Maxwell, a multi-FPGA system, by applying spatial parallelism to a massively-parallel accelerator implementation of FP-based cellular automata. Results showed that speedup diverges from linear scalability for more than 8 FPGAs, since parts of the computations were co-processed by a CPU. In [9], a fully-streamed computation for all LBM computing stages was created and processed on a single FPGA, where the CPU co-processing was eliminated. Results demonstrated 97.9% utilization of the peak performance with a single pipeline of 18 cascaded computing units, where dedicated dataflow-based FP operations are defined. However, it was also discovered that 99.6% consumption of FP digital signal processors (DSPs) in a single FPGA limits the scalability. As with LBM, tsunami simulation in [11] is capable of delivering high throughput with FPGA-based stream computing approach. It was previously demonstrated that for a single Arria 10 FGPA, the highest sustained performance was achieved by a single pipeline with 6 cascaded computing units, where its scalability is also limited by the available FP-DSPs. Similar to [9], this suggests the feasibility of extending the pipeline depth into multiple FPGAs.

Performance models for FPGA applications are important for scalability analysis and estimating achievable performance in different variants including future devices, as done by [8], [9], [11], [15]. Dohi et al. [8] introduced performance modeling of stream-based stencil computations on a single Maxeler Technology FPGA accelerator. However, the presented model is specific to both architecture and communication patterns on its platform. With multiple FPGAs, inter-FPGA communication is introduced as a new factor to be considered in analysis, as discussed in this paper.

While the TCP/IP network protocol is popular for internetworking systems, it is resource-heavy and designed for complex, unpredictable network, such as the Internet. A customized protocol, BlueLink [16] using high-speed serial links, showed better area-performance characteristics than existing network protocols for their custom computing requirements. Jun et al. [17] presented a parameterized, low overhead transport layer network with virtual channels and end-to-end flow control for distributed FPGA applications. Their prototype cluster is made up of 20 Xilinx VC707 FPGA boards connected through their high-speed serial links. In this current work, we implemented a high-speed inter-FPGA communication subsystem, where it is built on a lightweight, vendor-provided protocol that shares some similarities with BlueLink. In addition, we added a credit-based flow control mechanism [18] for backpressure propagation between FPGAs. Unlike in [17], however, careful analysis based on the physical constraints is done for the communication buffer requirements, which will be discussed in detail within the next section. We selected the credit-based scheme due to the advantages presented in [19], [20] such as: it is faster than its rate-based counterparts; there is no data loss if there is any congestion; and data rate can be as high as the full link speed with no data loss, which promises good network resource utilization.

3. Stream Computing Platform on Multiple FPGAs

3.1 Stream Computing and Parallelism Options

Stream computing is an approach that can be effectively utilized to achieve high throughput even with constant and limited memory bandwidth. To obtain computational results from a custom computing region in an FPGA, data is read from an external memory and continuously supplied as inputs to the computing units. Figure 1a presents a generalized stream computing unit, which is a computing pipeline with FP operations of a custom application. It takes in W_{in} words of input stream and generates a computational output stream of W_{out} words synchronously every clock cycle. Figure 1a shows a unit pipeline, where it takes D_{pipe} cycles to produce the computational output, which is proportional

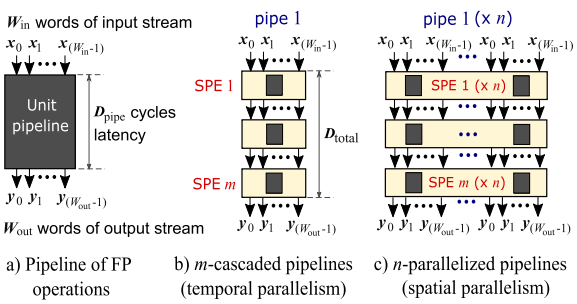


Fig. 1 Generalized stream computing model with stream processing elements (SPEs). Detailed discussions of tsunami simulation's SPEs and hardware algorithm are in [11].

to the number of pipeline stages. A deeper pipeline means more computational operations are performed at a constant throughput. One domain appropriate for stream computing is iterative stencil kernels, which are commonly found in scientific and engineering solutions.

Figure 1 also shows the available parallelism on the design space with the computational pipeline. Here, we define an SPE to contain a single unit pipeline. With each input stream, an SPE computes for its corresponding single time-step output. When $W_{\text{in}} = W_{\text{out}}$ words of stream width, the computational output of an SPE can be connected as the next time-step input to a replicated SPE. Figure 1b illustrates cascading m SPEs to form a single deep pipeline, which enables multiple time-step computations with a single data stream. This is similar to loop unrolling approach, which exploits temporal parallelism. This allows an increase in performance without increasing the number of memory accesses while concealing memory access latency. However, a deeper pipeline produces a large inefficient overhead and while computations are processed, the intermediate results are not stored to memory.

An SPE can also contain n-parallelized unit pipelines, which has n times higher performance in a single time-step, which exploits spatial parallelism. These pipelines can take in successive words from the input data stream, which made domain decomposition unnecessary for parallel computation. For the same number of operations utilized in temporal parallelism, there is lesser overhead in spatial parallelism since the pipeline depth is reduced. However, this approach increases the input stream bandwidth requirement because it needs an n times wider data stream. If available bandwidth is insufficient, stalls may occur, often leading to a decrease in performance. Figure 1c shows m-cascaded SPEs with n-parallelized unit pipelines.

In the FPGA's design space, SPEs can be either cascaded or parallelized to exploit fine-grained temporal or spatial parallelism, as presented in Figs. 1b and c, respectively. This (n, m) SPE configuration is placed in the FPGA's user-defined logic region, called a computing core or simply a *core*, which contains custom implementation of a stream computing algorithm.

Since there is a trade-off between temporal versus spatial parallelism, careful analysis should be done to balance the performance versus pipelining effect. Furthermore, a single FPGA has limited resources; thus, the number of SPEs is also limited. A workaround for this is to increase the number of SPEs over multiple FPGAs to further scale the performance.

3.2 Deeper Pipeline with Cascaded FPGAs

When the (n, m) SPE configuration on a computing core is replicated over multiple FPGAs, an even deeper computing pipeline is implemented. There are several choices on how to connect the FPGAs, but a 1D ring is the most straightforward topology, where it allows the inter-FPGA transceiver links to be bundled together to double the available bandwidth and achieve a higher network throughput.

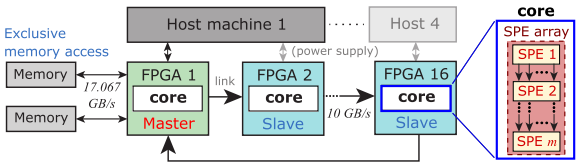


Fig. 2 FPGA cluster in 1D ring topology. SPEs are placed in the computing core of each FPGA.

In this approach, one *master* FPGA is adopted and from it, the cascaded FPGAs are called *slaves*. Having one master involves the simplicity in the localization of computational data in a single memory space. Here, the master FPGA will have an exclusive memory access; therefore, eliminating the necessity of a complicated control mechanism over a shared memory space across the other FPGAs.

Through the PCIe interface, the pre-computational data streams are transferred from the host to the master FPGA's memory. The master and slave FPGAs are implemented with stream computing pipelines, which accepts the data streams through the master. After the master handles the initial computations, the cascaded slave FPGAs receive the resulting data streams and handle all the consequent computations before returning the final results back for storage to memory. The 1D ring topology is shown in Fig. 2, which illustrates the master-slave configuration on 16 clustered FPGAs connected to 4 host machines, with an option to scale the number of hosts and FPGAs if necessary.

3.3 Inter-FPGA Communication Subsystem

One identified challenge with multiple FPGAs is synchronization of data streams, since the FPGAs are operating in different clock domains. Typically, dual-clock FIFOs at both transceiver ends handle this by allowing FIFO read and write access at different clocks. Flow control manages data synchronization between different asynchronous units such as FIFOs and SPEs by supplying backpressure signals. Furthermore, backpressure should also be available between two communicating FPGAs. With this, we designed and implemented a credit-based flow controller [18] (FC core) to add a reliability layer to any network protocol for the transceiver links.

FC core is designed to be general purpose with full-duplex symmetry, where it is implemented on both communicating ends. In addition, it must be able to operate on either half or full-duplex mode without setting any user-defined parameter. TX must regularly send *credits*, which is an approach used to mimic the backpressure effect. FC cores on both ends are keeping a running count of the transmitted payload, sent in *data flits*. Flit is a unit for smaller chunks of a large network packet that is sent in one cycle. The credit-based scheme allows TX to transmit payload only when there is available RX buffer space downstream.

Credit is embedded in a *control flit*, which is the first flit sent at every TX burst. On the receiving end, RX uses the received credit to update its credit counter. Figure 3 shows

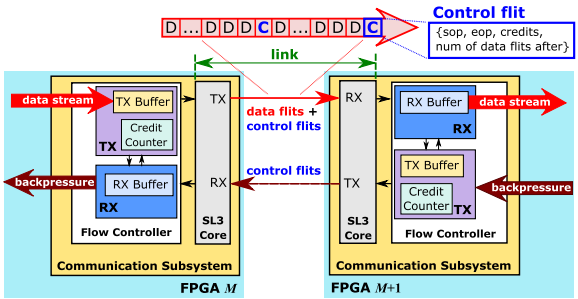


Fig. 3 Communication subsystem with FC and SL3 modules

the case of half-duplex mode between two FPGAs. Each communication subsystem includes an FC core and Serial-Lite III (SL3) core, a lightweight network protocol for high-bandwidth streaming data without a ready signal for back-pressure propagation.

In this work, communication buffer depths are considered to minimize area consumption without sacrificing performance. TX buffer stores outgoing data flits, where it only transmits them when an end-of-packet (EOP) signal is detected or when it is full. This means that TX buffer should be small enough to minimize induced waiting time before transmission. However, it should also be large enough to store more data flits in a single burst. Equation (1) shows the inter-FPGA link delay:

$$D_{\text{link}} = ((\text{link latency}) \times F) + (\text{TX buffer depth}) + (\text{RX buffer write-forward cycles}) \quad [\text{cycles}], \tag{1}$$

where (link latency) is the time it takes for a TX-sent flit to reach RX, F is the operating frequency, (TX buffer depth) is the TX buffering delay, and (RX buffer write-forward cycles) is the number of cycles before received flits become available from the RX buffer.

We also define the transmission overhead in Eq. (2), which is the ratio of control flit to the total number of flits sent in one burst. Transmission of more data flits per burst leads to a lesser overhead, which maximizes network bandwidth utilization. Since there is one control flit per burst, then:

$$(\text{TX overhead}) = \frac{1}{1 + (\text{TX buffer depth})}. \tag{2}$$

To operate at a high rate, RX buffer depth must be sufficiently larger than the round-trip link delay and credit update delay [18]. For bursty traffic, this large allocation allows high link utilization. Equation (3) summarizes RX buffer depth requirement:

$$(\text{RX buffer depth}) > (D_{\text{link}} \times 2) + D_{\text{CU}}, \tag{3}$$

where $(D_{\text{link}} \times 2)$ is the round-trip link delay, and D_{CU} is the interval at which RX sends a credit upstream.

Based on these equations, we selected the communication buffer depths. Figure 4 shows the effective link throughput when sending different data stream sizes with different

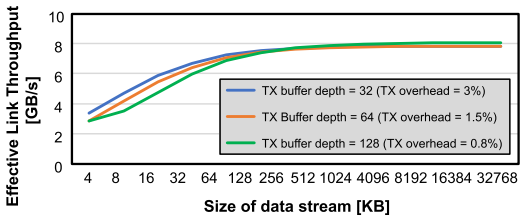


Fig. 4 Effective link throughput

TX buffer depths. For shorter data streams, smaller allocation has a higher effective throughput due to a smaller TX buffering overhead. However, this overhead becomes negligible in longer data streams, where the 3 different buffer depths converged to an effective link throughput of 7.9 GB/s. Since we target to design a general-purpose credit-based FC, we chose to use TX buffer depth = 32. Meanwhile, link latency is 446 ns, which is 100 cycles at F = 225 MHz. With this, $D_{\text{link}} = (100)+(32)+(3) = 135$ cycles, where TX buffer depth = 32 and RX buffer write-forward = 3 cycles. We then selected RX buffer depth = 512, which is sufficiently larger than $(135 \times 2) + (128) = 398$. Here, $D_{\text{CU}} = 128$ cycles and is a statically chosen interval that satisfies Eq. (3).

3.4 Performance Model

The performance model for stream computing with (n, m) SPE configuration [9], [11] is:

$$P_{\text{theory}}(n,m) = \frac{nmFO_{\text{pipe}}}{1 + \frac{mD_{\text{pipe}}}{C_{\text{stream}}}} \frac{\min(B_{\text{mem}}, b_{\text{core}})}{b_{\text{core}}} \quad [\text{GFlops}], \tag{4}$$

where O_{pipe} is the number of operations per unit pipeline, $D_{\text{pipe}}(n)$ is the pipeline depth of a unit pipeline, B_{mem} is available memory bandwidth, and $b_{\text{core}}(n)$ is the required computing core bandwidth. Here, $b_{\text{core}}(n) = nW_{\text{pipe}}F$, where W_{pipe} is the input/output width of a unit pipeline [bytes]. Since having n-parallelized pipelines requires n times wider data, the total number of stream cycles C_{stream} is inversely proportional to n: $C_{\text{stream}}(n) = \lceil N_{\text{grid}}/n \rceil$, where N_{grid} is the number of computational grid points to stream. We extended Eq. (4) to estimate the performance in the multi-FPGA platform. The contributing parameters are summarized in Table 1.

We introduce O_{FPGA} as the number of operations per FPGA, where $O_{\text{FPGA}}(n,m) = nmO_{\text{pipe}}$. To stream data with C_{stream} cycles, the total number of operations with M cascaded FPGAs is:

$$O_{\text{total}} = MO_{\text{FPGA}}C_{\text{stream}} = nmMO_{\text{pipe}}C_{\text{stream}} \quad [\text{ops}]. \tag{5}$$

Cascading FPGAs introduces the communication links into the model. Let B_{link} be the available link bandwidth. In general, when there is insufficient available bandwidth, pipeline stalls will occur. Here, we define stall ratio r_{stall} as the ratio of stall cycles to total cycles and utilization ratio $(1 - r_{\text{stall}})$ as the ratio of utilized cycles to total cycles:

$$r_{\text{stall}} = \begin{cases} 1 - \frac{B_{\text{link}}}{b_{\text{core}}} & , B_{\text{link}} < min(B_{\text{mem}}, b_{\text{core}}) \\ 1 - \frac{B_{\text{mem}}}{b_{\text{core}}} & , B_{\text{mem}} < min(B_{\text{link}}, b_{\text{core}}) \\ 0 & , \text{otherwise;} \end{cases} \quad (6)$$

$$(1 - r_{\text{stall}}) = \frac{\min(B_{\text{link}}, B_{\text{mem}}, b_{\text{core}})}{b_{\text{core}}}. \quad (7)$$

The entire computation for a single data stream takes $(C_{\text{stream}} + D_{\text{total}})$ cycles, where $D_{\text{total}}(M)$ is the total propagation delay from start to end of the entire computing pipeline. Here, $D_{\text{total}}(M) = M(D_{\text{core}} + D_{\text{link}})$, where core delay $D_{\text{core}}(m) = mD_{\text{pipe}}$ and D_{link} is the inter-FPGA link delay, as introduced in Eq. (1). Since pipeline stalls are anticipated with an insufficient available bandwidth, the total number of cycles for computation is:

$$\begin{aligned} C_{\text{total}} &= \frac{C_{\text{stream}} + M(D_{\text{core}} + D_{\text{link}})}{(1 - r_{\text{stall}})} \\ &= \frac{C_{\text{stream}} + M(mD_{\text{pipe}} + D_{\text{link}})}{(1 - r_{\text{stall}})} \quad \text{[cycles].} \end{aligned} \quad (8)$$

Finally, total performance $P_{\text{theory}}(M, n, m)$ is:

$$\begin{aligned} P_{\text{theory}} &= \frac{\text{(Total number of operations)}}{\text{(Total computing time)}} \quad \text{[GFlops]} \\ &= \frac{O_{\text{total}}}{C_{\text{total}}\left(\frac{1}{F}\right)} = \frac{MFO_{\text{FPGA}}C_{\text{stream}}}{C_{\text{total}}} \\ &= \frac{nmMFO_{\text{pipe}}C_{\text{stream}}(1 - r_{\text{stall}})}{C_{\text{stream}} + M(mD_{\text{pipe}} + D_{\text{link}})} \\ &= \frac{nmMFO_{\text{pipe}}}{1 + \frac{M(mD_{\text{pipe}} + D_{\text{link}})}{C_{\text{stream}}}} \frac{\min(B_{\text{link}}, B_{\text{mem}}, b_{\text{core}})}{b_{\text{core}}}. \end{aligned} \quad (9)$$

Based on Eq. (9), the following scaling factors are identified. First, $nmMFO_{\text{pipe}}$ defines the peak performance with M cascaded FPGAs, where $nmFO_{\text{pipe}}$ is the peak for a single FPGA. On the other hand, performance degradation due to pipeline overhead is indicated by $M(mD_{\text{pipe}} + D_{\text{link}})/C_{\text{stream}}$. This suggests that the overhead increases as the data stream size gets larger with respect to the total pipeline depth $M(mD_{\text{pipe}} + D_{\text{link}})$. Since M FPGAs would also scale the total propagation delay, therefore, adding more FPGAs will likewise contribute to the pipeline overhead. Finally, $\frac{\min(B_{\text{link}}, B_{\text{mem}}, b_{\text{core}})}{b_{\text{core}}}$ is the effect of insufficient available bandwidth, caused by either the links or by having n-parallelized pipelines in the core.

Table 1 Performance parameters

Parameters	Description	Unit
O_{total}	Total number of operations	[ops]
C_{total}	Total computing cycles	[cycles]*
F	Operating frequency	[Hz]
C_{stream}	Number of elements in data stream	-
m	Number of cascaded SPEs per FPGA	-
n	Number of parallel pipelines per FPGA	-
M	Number of cascaded FPGAs	-
O_{pipe}	Number of operations in a unit pipeline	[ops]
O_{FPGA}	Number of operations per FPGA	[ops]
B_{mem}	Available memory bandwidth	[Bytes/s]
B_{link}	Available inter-FPGA link bandwidth	[Bytes/s]
W_{link}	Width of inter-FPGA link	[Bytes]
b_{core}	Required computing core bandwidth	[Bytes/s]
W_{pipe}	Input and output width of a unit pipeline	[Bytes]
r_{stall}	Stall ratio	-
D_{total}	Total propagation delay	[cycles]*
D_{pipe}	Pipeline stages/delay in an unit pipeline	[cycles]*
D_{core}	Pipeline stages/delay in a core	[cycles]*
D_{link}	Inter-FPGA link delay	[cycles]*

*All delays are measured in cycles at the same operating frequency F.

4. Results and Discussion

4.1 Implementation

The acceleration platform with master and slave FPGAs is shown in Fig. 5. Currently, it is implemented with 8 Terasic DE5A-NET boards on 2 host machines, with the intention of extending up to 16 boards on 4 hosts, as introduced in Fig. 2b. Each board includes an Intel Arria 10 10AX115N3F45I2SG FPGA, 2 DDR3-2133 SDRAMs, a PCI-Express (PCIe) Gen2 x8 interface, and 4 high-speed, low-latency quad small form-factor pluggable (QSFP+) transceiver links, each of which has a bandwidth of 40 Gbps. Other necessary peripherals include: 2 DDR3 controllers, 4 scatter-gather direct memory access (SGDMA) modules, data width converters, hardware cycle counters, and dual clock FIFOs (DCFIFO). Data-width converters (WidthConv) convert the required bit-stream width for the computation and communication modules. Different clock domains are also utilized: 250 MHz for PCIe, 266.67 MHz for DDR3 controllers, and an operating range of up to 225 MHz is available for the computing core. Each SDRAM has a peak bandwidth of 17.067 GB/s, while the two bundled 40Gbps transceiver links claimed to reach 10 GB/s. However, as shown in Fig. 4, sustained link throughput with the communication subsystems averages at 7.9 GB/s. In a single FPGA, 2 communication subsystems are implemented, where the credit-based FC and SL3 cores are placed, and as reflected in Fig. 6, they leave a tiny footprint in the FPGA fabric.

For the custom computing core of the tsunami simu-

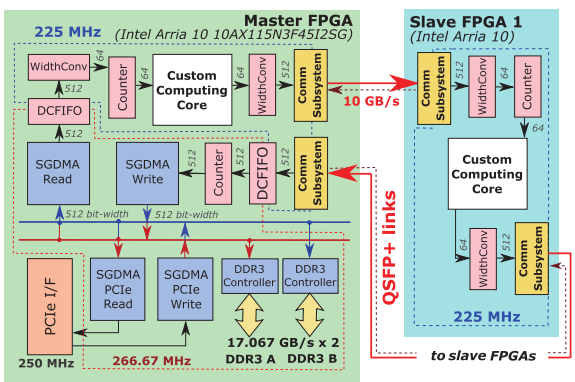


Fig. 5 Acceleration platform with master-slave FPGAs

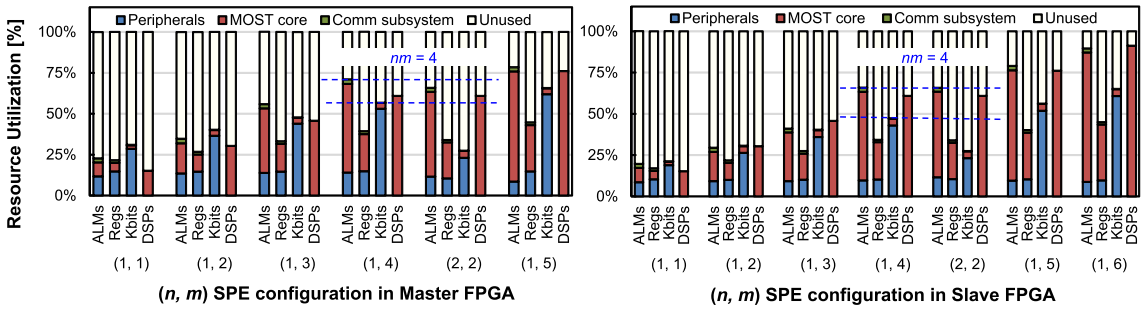


Fig. 6 Resource utilization with different SPE array configurations

lation, different (n, m) SPE configurations are generated using our domain specific language-based stream computing compiler (SPGen) [10], where all operations are in IEEE754 single precision FP format. Adders and multipliers are implemented using Intel IP cores on the FP-DSP blocks, while dividers and square root logic are generated using a FP-generation tool, FloPoCo [21]. We used $(n, m) = (1, 1), (1, 2), (1, 3), (1, 4), (2, 2), (1, 5), (1, 6)$ for design space exploration and utilized Intel Quartus Prime Pro 18.0 to generate the entire acceleration platform design.

As a result of master FPGA having more peripherals than its slave counterpart, it can fit only up to 5 SPEs, whereas the slave can accommodate 6 SPEs, where the limiting factor is the combination of both adaptive logic modules (ALMs) and DSPs. It is also noteworthy that the utilization of DSPs is solely by the computing core only, as shown in Fig. 6. For $(n, m) = (1, 4)$ and $(2, 2)$, they used the same number of DSPs in their SPEs ($nm = 4$); however, there is a noticeable difference in other areas like ALMs, registers (Regs), and block memories (Kbits) since having n-parallelized pipelines allowed them to share the same stencil buffers in the SPEs [11].

4.2 Benchmark Application: Tsunami Simulation

For benchmarking, tsunami simulation is implemented as the computing core in the multi-FPGA platform. Its algorithm is based on Method of Splitting Tsunami (MOST) [22], [23], which is a numerical method to solve shallow water equations for ocean-wide wave propagation. Implementation details about MOST-based SPEs were published in [11], where evaluation on a single Intel Arria 10 FPGA was presented. SPE configurations that are utilized in this paper have been previously verified to obtain the same computational results with a software-based simulation. Likewise, we used a 2581 x 2879 grid of real ocean-depth data to simulate the tsunami propagation in this work. Samples of FPGA-based computation visualizations can also be seen in [11].

4.3 Verification and Evaluation

We investigate the scalability and performance of the multi-FPGA platform using tsunami simulation's SPEs, in which one SPE with $nm = 1$ has 288 operations. Based on

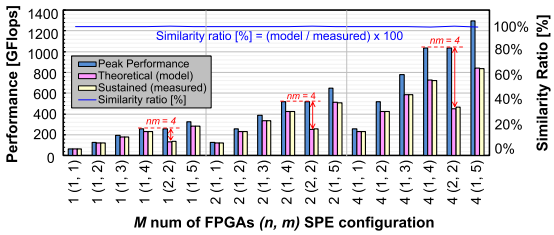
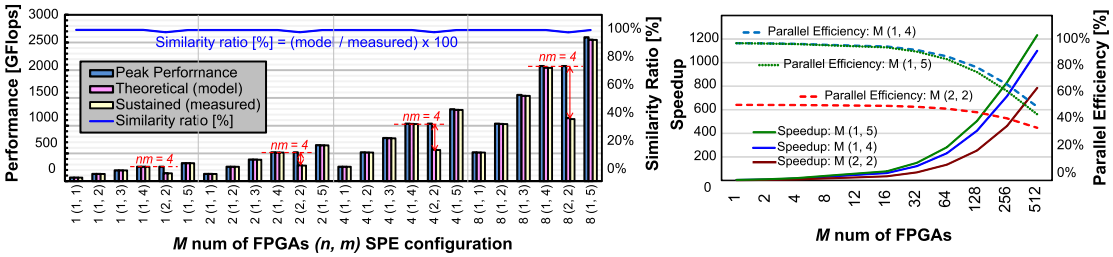


Fig. 7 Validation of performance model with $C_{\text{stream}} = 116{,}104$ cycles

Eq. (9), the peak performance $nmMFO_{\text{pipe}}$ is obtained with $F = 225$ MHz and $O_{\text{pipe}} = 288$. To obtain the theoretical sustained performance brought by degradation factors, $D_{\text{pipe}}(n) = 3099$ and 1808 for $n = 1$ and 2, respectively; while $D_{\text{link}} = 135$, as introduced in Sect. 3.3. For the available bandwidth, $B_{\text{mem}} = 17.067$ GB/s, and sustained $B_{\text{link}} = 7.9$ GB/s. The required bandwidth for tsunami simulation core is $b_{\text{core}}(n) = n \times 32 \times 0.225 = 7.2n$ GB/s, where $W_{\text{pipe}} = 32$ Bytes.

Using actual ocean-depth data requires a sufficiently large N_{grid}, in this case, with 2581×2879 data grid, which is equivalent to $C_{\text{stream}}(1) = 7{,}430{,}699$ cycles. To initially validate the model in Eq. (9), we first use a relatively smaller N_{grid} with $C_{\text{stream}}(1) = 116{,}104$ cycles, which is roughly 64 times smaller than the N_{grid} for tsunami simulation. Using up to $M = 4$ FPGAs, we obtained the peak, theoretical P_{theory}, and sustained performances of different SPE configurations, as shown in Fig. 7. Using the hardware counters in Fig. 5, we measured the stall cycles and total computing cycles for the sustained performance ratings. Figure 7 also shows the similarity ratio between theoretical and sustained performances, which is close to 100%, therefore, validating the model in Eq. (9).

Figure 8 shows the performance evaluation of tsunami simulation with actual ocean-depth data using up to $M = 8$ FPGAs. In Fig. 8a, its peak, theoretical, and sustained performances are illustrated for $C_{\text{stream}}(1) = 7{,}430{,}699$. For $n = 1$, the available bandwidth is sufficient ($B_{\text{mem}} > B_{\text{link}} > b_{\text{core}}(1)$), making sustained performance close to its peak. In the case of $nm = 4$, two SPE configurations are implemented: $(1, 4)$ and $(2, 2)$, where $n = 2$ caused an increase of bandwidth requirement ($B_{\text{mem}} > b_{\text{core}}(2) > B_{\text{link}}$) when $b_{\text{core}}(2) = 2 \times 32 \times 0.225 = 14.4$ GB/s. This led to pipeline stalls resulting to a lower sustained performance. For $M = 8$ FPGAs, $(n, m) = (1, 5)$ obtained the highest performance



a) Performance ratings using actual ocean-depth data

b) Speedup vs. Parallel efficiency

Fig. 8 Performance evaluation of tsunami simulation

with 2.5 TFlops, which is 98% of its peak performance. This shows that the total pipeline depth of $8 \times 5 = 40$ cascaded SPEs is sufficiently enough to accommodate the input C_{stream}, without being affected by the pipeline overhead. In the case where $C_{\text{stream}}(1) = 116{,}104$ cycles, as shown in Fig. 7, the pipeline overhead is visibly reflected with the significant difference between the peak and sustained performances as the pipeline depth increases.

Figure 8b shows the speedup and parallel efficiency of the largest SPE configurations that can fit the master and slave FPGAs for efficient resource utilization: $(1, 5), (1, 4)$, and $(2, 4)$. The expected speedup is achieved for $n = 1$, when M FPGAs are increased. The differences among the 3 SPE configurations are significantly observed with more FPGAs due to the pipeline overhead caused by the fixed ocean-depth data grid. $(1, 5)$ has the best speedup but its parallel efficiency is slightly lower than $(1, 4)$'s, due to the pipeline overhead when the problem size of computation is maintained. $(2, 2)$ has the lowest speedup rate and efficiency because of the insufficient bandwidth caused by having n-parallelized pipelines. This illustrates the performance model's prediction on the factors causing performance degradation, which in this case, is the bottleneck in the inter-FPGA links due to insufficient B_{link}.

With this, expanding the design space with multiple FPGAs supports further performance scaling. The key is finding the balance between the M, n, and m to achieve the best speedup and parallel efficiency rates. In the case of tsunami simulation, the large ocean-depth data grid allowed performance scaling when we increase the pipeline depth over multiple FPGAs. Ideally, n-parallelized pipelines would be the best approach since it would mean lesser computing cycles. However, this requires a larger bandwidth requirement, where B_{link} was not able to satisfy. Based on Fig. 8b, implementing $(n, m) = (1, 5)$ SPE array is the best option in terms of area, speedup, and efficiency, even though the latter is slightly lower than $(1, 4)$'s. With the currently utilized data grid, cascading up to $16(1, 5) = 80$ SPEs and $32(1, 5) = 160$ SPEs, have parallel efficiencies of 97% and 94%, respectively, which can still be acceptable rates. However, $M > 32$ FPGAs will bring a rapid rate of decreasing efficiency due to pipeline overhead.

5. Conclusions

This paper presents the design and architecture of a deeply pipelined stream computing platform on a 1D ring of master and slave FPGAs with communication subsystem. Temporal and spatial parallelism in the custom computing core are explored to efficiently utilize the hardware resources. Fine-grained temporal parallelism is achieved by m-cascaded SPEs while spatial parallelism is explored by having n-parallelized pipelines. By cascading the SPEs on multiple FPGAs, a deeper computing pipeline with a vast design space is achieved to support further performance scaling. Performance characteristics of the inter-FPGA links were also investigated, where communication buffer depths affect the sustained performance.

A performance model is also presented and validated by implementing a custom practical application on the stream computing prototype platform with 8 cascaded FPGAs at 80 Gbps links. With this, a practical and efficient exploration of the vast design space can be achieved with the model. Tsunami simulation is implemented and evaluated on the master and slave FPGAs. The highest scaled performance on 8 FPGAs is achieved with a single pipeline of 5 cascaded SPEs $(n, m) = (1, 5)$, where 40 SPEs in a deep pipeline delivered a scaled performance of 2.5 TFlops and a parallel efficiency of 98%.

Future work includes performance estimation and parameter optimization for forthcoming FPGAs such as the Intel Stratix 10 FPGAs, which have a larger design area and wider available bandwidth, where we can explore multiple n-parallelized pipelines. In addition, we intend to include applying the stream computing platform to other network topologies such as a 2D torus with multiple FPGAs to further scale the performance.

Acknowledgments

This research was partially supported by Grant-in-Aid for Scientific Research (B) No.17H01706 from MEXT, Japan. The authors thank the support of Intel university program.

References

[1] A. Mondigo, T. Ueno, D. Tanaka, K. Sano, and S. Yamamoto, "Design and scalability analysis of bandwidth-compressed stream computing with multiple FPGAs," Proceedings of the 12th International Symposium on Reconfigurable Communication-Centric Systems-on-Chip, ReCoSoC 2017, Madrid, Spain, pp.1–8, IEEE, 2017.

[2] A. Mondigo, K. Sano, and H. Takizawa, "Performance estimation of deeply pipelined fluid simulation on multiple FPGAs with high-

speed communication subsystem," Proceedings of the 29th Annual IEEE International Conference on Application-specific Systems, Architectures and Processors, ASAP 2018, pp.1–4, IEEE, 2018.
[3] W. Vanderbauwhede and K. Benkrid, eds., High-Performance Computing Using FPGAs, Springer New York, New York, NY, 2013.
[4] M. Vestias and H. Neto, "Trends of CPU, GPU and FPGA for high-performance computing," Proceedings of the 2014 24th International Conference on Field Programmable Logic and Applications (FPL), Munich, Germany, pp.1–6, IEEE, Sept. 2014.
[5] M. Parker, "Understanding peak floating-point performance claims," Technical report (white paper): Intel, WP-01222-1.1, 2017.
[6] M. Langhammer and B. Pasca, "Floating-point DSP block architecture for FPGAs," Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays - FPGA '15, New York, New York, USA, pp.117–125, ACM, 2015.
[7] M. Lin, S. Cheng, and J. Wawrzynek, "Cascading deep pipelines to achieve high throughput in numerical reduction operations," 2010 International Conference on Reconfigurable Computing Cascading, Quintana Roo, Mexico, pp.103–108, IEEE, 2010.
[8] K. Dohi, K. Okina, R. Soejima, Y. Shibata, and K. Oguri, "Performance modeling of stencil computing on a stream-based FPGA accelerator for efficient design space Exploration," PAPER Special Section on Reconfigurable Systems, IEICE Transactions, vol.E98-D, no.2, pp.298–308, 2015.
[9] 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, 2017.
[10] K. Sano, "DSL-based design space exploration for temporal and spatial parallelism of custom stream computing," Proceedings of the Second International Workshop on FPGAs for Software Programmers (FSP 2015), pp.29–34, Aug. 2015.
[11] K. Nagasu, K. Sano, F. Kono, and N. Nakasato, "FPGA-based tsunami simulation: Performance comparison with GPUs, and roofline model for scalability analysis," Journal of Parallel and Distributed Computing, vol.106, pp.153–169, Aug. 2016.
[12] M.C. Herbordt, T. VanCourt, Y. Gu, B. Sukhwani, A. Conti, J. Model, and D. DiSabello, "Achieving high performance with FPGA-based computing," Computer, vol.40, no.3, pp.50–57, March 2007.
[13] A. Azarian and J.M.P. Cardoso, "Coarse/fine-grained approaches for pipelining computing stages in FPGA-based multicore architectures," Proceedings of the European Conference on Parallel Processing: Euro-Par 2014: Parallel Processing Workshops, vol.8806, pp.266–278, Springer, 2014.
[14] S. Murtaza, A.G. Hoekstra, and P.M.A. Sloot, "Cellular automata simulations on a FPGA cluster," The International Journal of High Performance Computing Applications, vol.25, no.2, pp.193–204, May 2011.
[15] Y. Kono, K. Sano, and S. Yamamoto, "Scalability analysis of tightly-coupled FPGA-cluster for lattice Boltzman computation," Proceedings of the 22nd International Conference on Field Programmable Logic and Applications (FPL 2012), pp.120–127, IEEE, 2012.
[16] A.T. Markettos, P.J. Fox, S.W. Moore, and A.W. Moore, "Interconnect for commodity FPGA clusters: Standardized or customized?," Conference Digest - 24th International Conference on Field Programmable Logic and Applications, FPL 2014, pp.1–8, 2014.
[17] S.-W. Jun, M. Liu, S. Xu, and Arvind, "A transport-layer network for distributed FPGA platforms," 2015 25th International Conference on Field Programmable Logic and Applications (FPL), London, UK, pp.1–4, IEEE, Sept. 2015.
[18] N.T. Kung and R. Morris, "Credit-based flow control for ATM networks," IEEE Netw., vol.9, no.2, pp.40–48, 1995.
[19] R. Jain, "Congestion control and traffic management in ATM networks: Recent advances and a survey," Computer Networks and ISDN Systems, vol.28, no.13, pp.1723–1738, Oct. 1996.
[20] S. Kamolphiwong, A.E. Karbowiak, and H. Mehrpour, "Flow control in ATM networks: a survey," Elsevier Computer Communications, vol.21, no.11, pp.951–968, 1998.
[21] "FloPoCo Project WEB."
[22] V. Titov and F. Gonzales, "Implementation and testing of the Method of Splitting Tsunami (MOST) Model," 1997.
[23] M. Lavrentiev-jr, A. Romanenko, V. Titov, and A. Vazhenin, "High-performance tsunami wave propagation modeling," Parallel Computing Technologies, Lecture Notes in Computer Science, vol.5698, pp.423–434, Springer, Berlin, Heidelberg, 2009.

Antoniette Mondigo received B.S. and M.E. degrees in Computer Engineering from University of San Carlos, Philippines in 2006 and 2012, respectively. She is now a Ph.D. student in the Graduate School of Information Sciences at Tohoku University and at the same time, a student trainee in the Processor Research Team at Riken Center for Computational Science. Her research interests include architecture and design of reconfigurable systems with FPGAs. She is a student member of IEEE.

Tomohiro Ueno is currently a postdoctoral researcher of the Processor Research Team at Riken Center for Computational Science. He received his Ph.D. from the Graduate School of Information Sciences, Tohoku University, in 2016. Since then he had been a postdoctoral researcher in the Graduate School of Engineering at Tohoku University for a member of ImPACT project until 2017. His research interests include hardware architectures, hardware algorithms, data compression, and communication networks for HPC. He is a member of IEEE and IPSJ.

Kentaro Sano is the leader of the Processor Research Team at Riken Center for Computational Science. He received his Ph.D. from the Graduate School of Information Sciences, Tohoku University, in 2000. Until 2017, he had been an associate professor in the Graduate School of Information Sciences at Tohoku University. His research interests include novel architectures, hardware algorithms, system software, HLS compilers for HPC systems and reconfigurable systems with FPGAs. He is a member of IEEE, ACM, and IPSJ.

Hiroyuki Takizawa is currently a professor of Cyberscience Center and Graduate School of Information Sciences, Tohoku University. His research interests include HPC systems and their applications. He received the B.E. Degree in Mechanical Engineering, and the M.S. and Ph.D. Degrees in Information Sciences from Tohoku University in 1995, 1997, and 1999, respectively. He is a member of IEEE CS, ACM SIGHPC, and IPSJ.