DESIGN SOLUTION

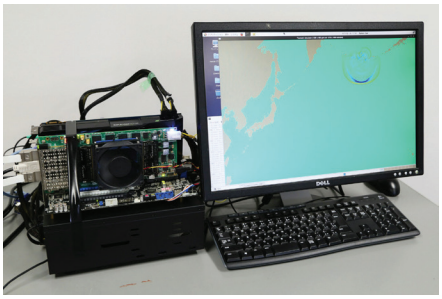
Accelerating Simulation of Tsunamis

By Kentaro Sano, Associate Professor, Tohoku University

TOHOKU UNIVERSITY

The Design Team

Professor Sano, a member of the Department of Computer and Mathematical Sciences at Tohoku University (Sendai, Japan), conducts research in parallel and reconfigurable computing. His team has applied FPGAs and GPUs to acceleration of demanding physics problems.



Challenge

Simulation of tsunamis combines simulation of a physical process with massive data on ocean depths. These calculations are generally done on massively parallel supercomputers, but the result is poor hardware utilization and performance.

Solution

Accelerating the inner loops of the simulation with hardware floating-point FPGAs resulted in performance up to 383 GFLOPS at power efficiency of over 8.4 GFLOPS/W†.

The Project

We needed a high-performance but low-power computing solution to perform tsunami simulation with a real data-set of ocean depth. This requires efficient computation with high utilization of floating-point operators under a limited bandwidth of external memories.

The simulation uses what are known as shallow-water equations, a set of partial differential equations that describe fluid flow in response to a pressure wave. We provide as input to the solver actual bathymetry data—measured depth information about the ocean as it approaches the shore.

The Design Challenge

Tsunami simulation is usually performed by using supercomputers with multi-core processors and many-core accelerators such as graphic processing units (GPUs). Software-based parallel computing with a lot of on-chip cores on computing nodes achieves high-performance simulation.

But such software-based massively parallel computing often suffers from insufficient memory bandwidth and becomes inefficient–especially in the case of algorithms with a low operational intensity. Operational intensity is a ratio of the number of operations to the amount of data that must be loaded from external memories. It is expressed as the number of operations performed per a unit data size, typically in FLOP/Byte. Since each processor has its inherent ratio of the peak arithmetic performance in GFLOPS to the available external memory bandwidth in gigabytes per second (Gbps), sustainable performance is generally limited by the memory bandwidth when algorithms have a low operational intensity.

As a result, such memory-bound computation becomes inefficient in terms of performance per computing core, performance per power, and performance per cost. Although there exist several techniques to improve operational intensity for some specific algorithms, such as temporal blocking with more efficient reuse of data in cache memories, a fixed memory sub-system of CPUs or GPUs is fundamentally not suitable to implement ideal data movement between cores on a chip. For example, CPUs and GPUs need to read/write data from/to a shared cache memory even if we should directly move the data from one core to another.

The Design Solution

Computation of tsunami simulation is expressed as stencil computation for a finite difference algorithm. Stencil computation uses data of adjacent grid cells to update each cell.

The tsunami is simulated as ocean-wide wave propagation by solving the shallow water equations (SWEs), which describe the flow below a pressure surface of a fluid. SWEs are composed of a set of the following partial differential equations (PDEs):

$$\frac{\partial z}{\partial t} + A\frac{\partial z}{\partial x} + B\frac{\partial z}{\partial y} = F$$

Where

$$z = \begin{pmatrix} u \\ v \\ H \end{pmatrix}, \quad A = \begin{pmatrix} u & 0 & g \\ 0 & u & 0 \\ H & 0 & u \end{pmatrix}, \quad B = \begin{pmatrix} v & 0 & 0 \\ 0 & v & g \\ 0 & H & v \end{pmatrix}, \quad F = \begin{pmatrix} gD_x \\ gD_y \\ 0 \end{pmatrix}.$$

Here, t, x, and y are the time, and the coordinates in the longitude and latitude directions, respectively. The u, v, and g denote the longitude and latitude components of the wave velocity, and the acceleration of gravity. H is the length between the surface and the bottom water, which is given by $H=D+\eta$ where D and η denote the mean height of the ocean surface and the height of propagating waves, respectively.

One of the methods to numerically solve SWEs is the method of splitting tsunami(MOST) [1] . In MOST, spatial decomposition is applied along the coordinates so that the SWEs are split and converted into two canonical forms of PDEs for x and y, respectively. For example, the canonical form of the equation for x direction is obtained as:

$$\frac{\partial w}{\partial t} + A'\frac{\partial w}{\partial x} = F'$$

with

$$w = \begin{pmatrix} v' \\ p \\ q \end{pmatrix} = \begin{pmatrix} v \\ u + 2\sqrt{gH} \\ u - 2\sqrt{gH} \end{pmatrix}, \quad A' = \begin{pmatrix} \lambda_1 & 0 & 0 \\ 0 & \lambda_2 & 0 \\ 0 & 0 & \lambda_3 \end{pmatrix}, \quad F' = \begin{pmatrix} 0 \\ gD_x \\ gD_x \end{pmatrix},$$

Where v', p, and q are Riemann invariants of the equation. λ_1, λ_2, and λ_3 are the eigenvalues of A, which are given with $\lambda_1 = u$ and $\lambda_{2,3} = u \pm \sqrt{gH}$. More details are available in [1]. Then we apply the finite difference scheme to make discrete forms of the canonical equations on a 2-dimensional orthogonal grid. For the canonical form of x direction, we obtain:

$$\frac{w_{i,j}^{n+1} - w_{i,j}^{n}}{\Delta t} + A'\frac{w_{i+1,j}^{n} - w_{i-1,j}^{n}}{2\Delta x} - A'\Delta t\frac{A'(w_{i+1,j}^{n} - w_{i,j}^{n}) - A'(w_{i,j}^{n} - w_{i-1,j}^{n})}{2\Delta x^2} = \frac{F'^{\square}_{i+1,j} - F'^{\square}_{i-1,j}}{2\Delta x} - A'\Delta t\frac{F'^{\square}_{i+1,j} - 2F'^{\square}_{i,j} + F'^{\square}_{i-1,j}}{2\Delta x^2},$$

Where n, i, and j denote the time step, a position on the 2D grid, respectively. The discrete equation for y direction is given similarly. Then we can obtain the values for the next time step, (n+1), by numerically solving these discrete equations, one by one, separately along the longitude and latitude coordinates.

We designed custom computing hardware for high-performance tsunami simulation with FPGAs, based on deep pipelining and coarse-grained parallelism for performance scalability on a chip. By pipelining, we can perform a lot of operations per memory access to increase the sustained performance even with a limited bandwidth of external memories. Coarse-grained parallelism allows us to exploit available hardware resources to scale the performance.

Theory of Operation

Figure 1 shows a stream processing element (SPE) which computes a single time step of the tsunami propagation based on MOST. The SPE is designed as a pipeline to take an input of a data stream with grid cells, each of which consists of eight 32 bit words. By traversing a grid cell in the x-direction first order and inputting the sequence of cells to the SPE, we can obtain the grid data updated for a single time step. To construct a deep pipeline, we first merged all the computing stages including boundary treatment for both x- and y- directions into a single loop kernel, and then streamed it for hardware design.

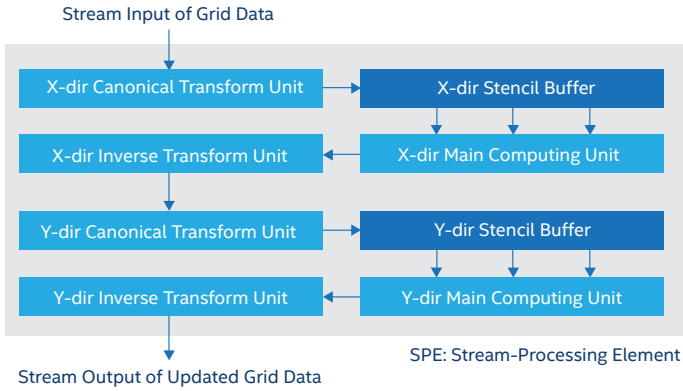


Figure 1. A Stream Processing Element (SPE)

As Figure 1 shows, SPE is composed of two major parts: the x-dir stages and the y-dir stages. Each part consists of a canonical transform unit, a stencil buffer, a main computing unit, and an inverse transform unit. The transform units and the computing units have data-flow structures for all numerical computations with single-precision floating-point operators. The stencil buffer is used to refer the data of the adjacent grid cells in their streamed sequence. The stencil buffer has a structure with shift registers, to delay the outputs of the streamed input for multiple references of different cells. This technique is also known as a line buffer, and it is especially used in image processing applications. In a single SPE, we implemented 147 adders, 121 multipliers, 12 dividers, and 2 square root operators, which are totally counted as 288 equivalent arithmetic operators. Here, we consider a single square root operator as four equivalent arithmetic operations.

Since a state-of-the-art FPGA is very big with a lot of logic elements, registers, and DSP blocks, we cannot use all of them only with a single SPE. We introduced a coarse-grained array with the spatial and temporal parallelism [2] of an SPE, as shown in Figure 2. First, we increase the number of internal pipelines in the SPE to perform more operations every cycle. With n pipelines, we can have n times higher performance if simultaneously n times wider bandwidth is available. If not, the performance is limited by the available bandwidth. Then, we introduced temporal parallelism by pipelining multiple SPEs to compute multiple successive time-steps at one time. In Figure 2, we cascade m SPEs to make the pipeline m times longer. This technique is similar to so-called loop-unrolling used in software optimization. Since cascading SPEs does not increase the bandwidth requirement of a stream, we can fill an FPGA with many SPEs and fully utilize them even with limited bandwidth available.

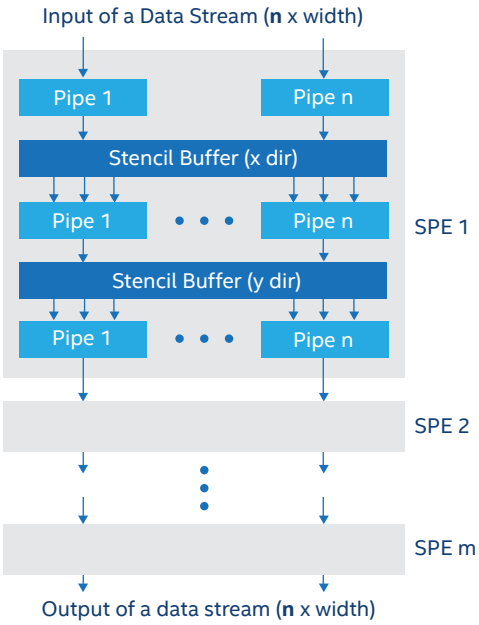


Figure 2. A Pipeline of SPEs

However, the temporal parallelism also has disadvantages. First, the much longer pipeline has a larger pipelining overhead, which causes lower utilization due to the prologue and epilogue in pipelining. When the data stream is not sufficiently long, a long pipeline is likely to cause inefficiency. Second, we cannot save the intermediate results in the pipeline to memory or storage. For example, assuming 1000 SPEs were cascaded, we could send a result to memory only once in every 1000 time-steps. Due to these reasons, it is not feasible to infinitely increase the number of cascaded SPEs. Thus, we need to balance the spatial and temporal parallelism by constructing SPE arrays of Figure 2 with n pipelines and m cascaded SPEs, which is denoted by (n, m).

We implemented the SPE array operating at 225 MHz with an Intel® Arria® 10 10AX115 FPGA on a DE5A-NET board. Since the peak bandwidth of 2 x 12.8 Gbps given by two DDR3 SDRAMs on the board can satisfy only the bandwidth requirement of a single pipelined SPE, we explored the design space for SPE arrays with $(n, m) = (1, m)$. We implemented the data-flow pipeline with single-precision floating point operators by using our own developed compiler, called SPGen [2, 3]. With SPGen, we can generate datapaths for data flow with codes describing computing formulae and calls of functions. We used the hardware floating-point DSP blocks of the Arria 10 FPGA for addition and multiplication, while the resource assignment can be further optimized.

We put the SPE array into the user designed streaming core in the acceleration framework shown in Figure 3. The framework provides fixed functions necessary for stream computation, which includes a PCI Express* interface, DDR3 memory interfaces, scatter-gather(SG) DMAs, dual-clock(DC) FIFOs, and width converters of Avalon® Streaming (Avalon-ST) connections. We also implemented a Linux* PCI Express driver with SGDMA control capability, and software which initializes the grid data, uploads them to the FPGA's external memories, starts computation with the FPGA, downloads the results, and visualizes it. The result is a real-time FPGA-based computation and visualization system, which is shown in Figure 4. The system simulates tsunami propagation over the Pacific ocean with the real geometry data of the sea (called bathymetry data). The computational grid as 2581 x 2879 cells.

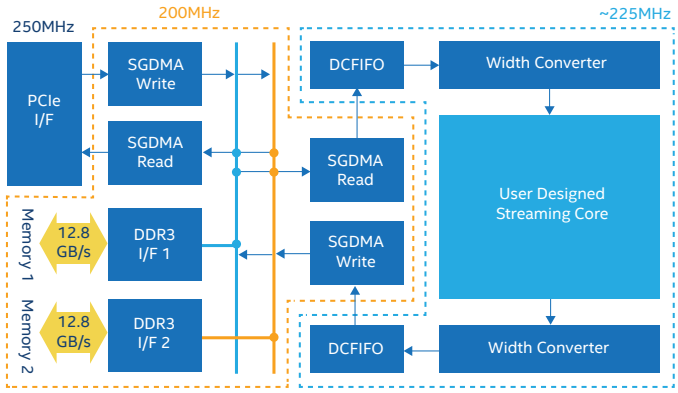


Figure 3. The Acceleration Framework

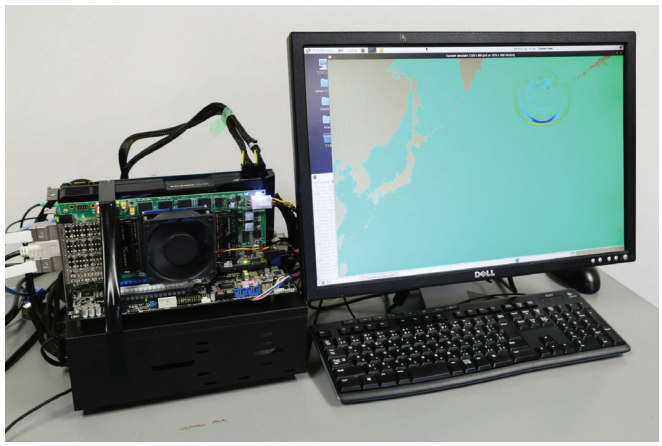


Figure 4. The FPGA Demonstration System

Results

The detailed results are available in [3] for implementation with the Arria 10 FPGA and performance comparison with GPUs. With Arria 10 10AX115 FPGA, we could implement up to m=6: six cascaded SPEs. Figure 5 shows the resource consumption of the entire design including the framework for m=1, 2, 4, and 6. The number of SPEs is limited by both adaptive logic modules (ALMs) and floating-point DSP blocks. The array of six cascaded SPEs use 1386 floating-point DSP blocks, but some of them are not fully utilized for both addition and multiplication yet.

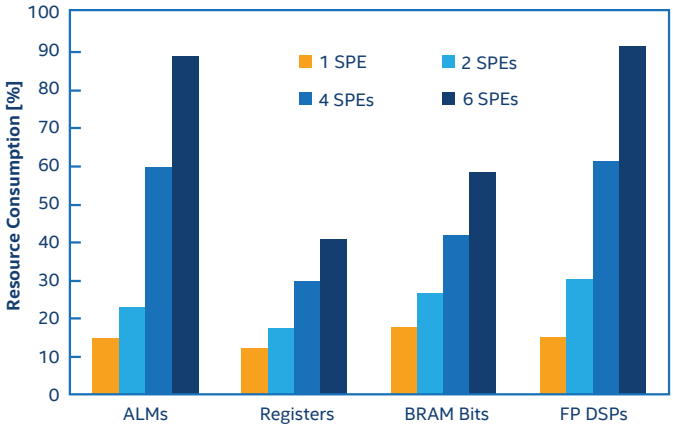


Figure 5. Resource consumption by SPE array (Arria 10 10AX115 FPGA)

We measured a stall ratio—the ratio of stall cycles to the total processing cycles. Then we calculated the sustainable performance of tsunami simulation with the measured stall ratio, the number of equivalent arithmetic operations per SPE (288), and the Arria 10 streaming core operating frequency of 225 MHz. The array with m=6 cascaded SPEs achieved 383 GFlops. We also measure the voltage and current for the DE5A-NET board at the PCI Express edge connector and the aux power supply of the board. Since the power consumption was 45.5 W, the sustainable performance per power was 8.42 GFlops/W for the board†. This includes all power consumed by the board, not just core power consumption.

We also made performance and power measurements for GPUs. As reported in [3], tsunami simulation implementations with Tesla K20 and Radeon R9 280X GPUs achieved 44.6 and 219 GFLOPS at 91.5 and 184.9 W, respectively†. We used CUDA and OpenCL™ to write the codes and optimize them to a certain degree for Tesla K20 and Radeon R9 280X GPUs, respectively. Of course further optimization may still be possible. As a result, these GPUs achieved the sustainable performance per power of 0.49 and 1.19 GFLOPS/W†. These results demonstrate that, at least in comparison with GPUs, the state-of-the-art mid-level FPGA can bring comparable or higher floating-point performance even with a much lower bandwidth of external memories, and can achieve much higher sustainable performance for the power consumed.

References

[1] V. Titov and F. Gonzalez, “Implementation and testing of the method of splitting tsunami (MOST) model,” NOAA Technical Memorandum ERL PMEL-112, 1997.

[2] Kentaro 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), arXiv:1509.00040, September, 2015.

[3] Kohei Nagasu, Kentaro Sano, Fumiya Kono, and Naohito Nakasato, “FPGA-based Tsunami Simulation: Performance Comparison with GPUs, and Roofline Model for Scalability Analysis,” Journal of Parallel and Distributed Computing, http://dx.doi.org/10.1016/j.jpdc.2016.12.015, (in press).

† Tests measure performance of components on a particular test, in specific systems. Differences in hardware, software, or configuration will affect actual performance. Consult other sources of information to evaluate performance as you consider your purchase. For more complete information about performance and benchmark results, visit www.intel.com/benchmarks.

DS-1005-1.0