Solving the Global Atmospheric Equations through Heterogeneous Reconfigurable Platforms

LIN GAN and HAOHUAN FU, Tsinghua University
WAYNE LUK, Imperial College London
CHAO YANG, Institute of Software, and State Key Laboratory of Computer Science, Chinese Academy of Sciences
WEI XUE, XIAOMENG HUANG, YOUHUI ZHANG, and GUANGWEN YANG, Tsinghua University

One of the most essential and challenging components in climate modeling is the atmospheric model. To solve multiphysical atmospheric equations, developers have to face extremely complex stencil kernels that are costly in terms of both computing and memory resources. This article aims to accelerate the solution of global shallow water equations (SWEs), which is one of the most essential equation sets describing atmospheric dynamics. We first design a hybrid methodology that employs both the host CPU cores and the field-programmable gate array (FPGA) accelerators to work in parallel. Through a careful adjustment of the computational domains, we achieve a balanced resource utilization and a further improvement of the overall performance. By decomposing the resource-demanding SWE kernel, we manage to map the double-precision algorithm into three FPGAs. Moreover, by using fixed-point and reduced-precision floating point arithmetic, we manage to build a fully pipelined mixed-precision design on a single FPGA, which can perform 428 floating-point and 235 fixed-point operations per cycle. The mixed-precision design with four FPGAs running together can achieve a speedup of 20 over a fully optimized design on a CPU rack with two eight-core processorsand is 8 times faster than the fully optimized Kepler GPU design. As for power efficiency, the mixed-precision design with four FPGAs is 10 times more power efficient than a Tianhe-1A supercomputer node.

Categories and Subject Descriptors: C.3.e [**Special-Purpose and Application-Based Systems**]: Reconfigurable Hardware

General Terms: Design, Algorithms, Performance

Additional Key Words and Phrases: Atmospheric equations, high performance computing, FPGAs, hybrid algorithm, mixed-precision design

---

Direct comments and questions about this article to Dr. Haohuan Fu (haohuan@tsinghua.edu.cn).
This work was supported in part by the National Natural Science Foundation of China (grant nos. 61303003 and 41374113), the National High-Tech R&D (863) Program of China (grant no. 2013AA01A208), UK EPSRC, the European Union Seventh Framework Programme (grant agreement nos. 257906, 287804, and 318521), HiPEAC NoE, the Maxeler University Program, and Xilinx.
Authors' addresses: L. Gan, H. Fu, X. Huang, and G. Yang, Room S817, Meng Minwei Science, Tsinghua University, Beijing, China, 100084; emails: l-gan11@mails.tsinghua.edu.cn, {haohuan, hxm, ygw}@tsinghua.edu.cn; W. Luk, Department of Computing, Imperial College London, 180 Queen's Gate, London SW7 2BZ, England; email: wl@doc.ic.ac.uk; C. Yang, Institute of Software, Chinese Academy of Sciences, Beijing, China, 100190; email: yangchao@iscas.ac.cn; W. Xue, Room S813, Meng Minwei Science, Tsinghua University, Beijing, China, 100084; email: xuewei@tsinghua.edu.cn; Y. Zhang, Room 8-209, East Main Building, Tsinghua University, Beijing, China, 100084; email: zyh02@tsinghua.edu.cn.
Authors' current address: L. Gan and H. Fu, Ministry of Education Key Laboratory for Earth System Modeling, and Center for Earth System Science, Tsinghua University.

DOI: http://dx.doi.org/10.1145/2629581

ACM Reference Format:
Lin Gan, Haohuan Fu, Wayne Luk, Chao Yang, Wei Xue, Xiaomeng Huang, Youhui Zhang, and Guangwen Yang. 2015. Solving the global atmospheric equations through heterogeneous reconfigurable platforms. ACM Trans. Reconfig. Technol. Syst. 8, 2, Article 11 (March 2015), 16 pages.
DOI: http://dx.doi.org/10.1145/2629581

1. INTRODUCTION

In recent decades, climate change has brought significant influence on human activities. Investigating the climate change mechanism has become an important research issue among governments and research institutes. However, due to the complicated nature and extremely large simulating domain, it is difficult for scientists to verify their theories through controlled experiments, such as those in physics and chemistry. Computer-based modeling becomes the key method to study the climate change mechanisms and make predictions into future climate risks. Since the first climate modeling program [Charney and Eliassen 1949] running on ENIAC (the first electronic digital computer), climate research has made rapid progress with the technical revolution of computers.

Among all of the different components in a climate system, the global atmospheric circulation model is one of the most essential and challenging ones. Developers have to face the difficulties from the complex kernels that involve multiple scales and multiple physics, and from the requirement for high resolution that involves billions of mesh points. These issues bring a tough challenge to the computing capability of platforms and call for a wise manner to handle the large dataset.

Current high-performance platforms, such as the CPU, GPU, and Intel Many Integrated Core (MIC), are mostly based on multicore or many-core architectures and can improve the performance not only through the increasing computing capabilities of new architectures but also through enabling applications to run on hundreds or even thousands of computing cores or vector units. However, as multicore and many-core architectures achieve parallelism through single instruction multiple data (SIMD) or single thread multiple data (STMD) approaches, applications with complex and irregular computation and heavy communications, such as the upwind stencil in atmospheric equations, would generally face the constraints of memory and communication bandwidth. Moreover, the atmospheric modeling usually desires large-scale scenarios, which brings great power consumptions and resource usages when running on traditional platforms.

Reconfigurable dataflow engines (DFEs), such as field-programmable gate arrays (FPGAs), achieve high parallelism through a deep pipeline of computing units and can deploy the computation blocks through user-defined circuits rather than through processors that take instructions. The customizable features on data presentations enable the combination of different data types and precisions, which brings a big design space on improving the performance and reducing the resource cost at the same time. Furthermore, the magnitude of low chip frequency generally leads to great power efficiency than traditional computing platforms.

In this article, we propose a hybrid algorithm to solve the global shallow water equations (SWEs), which is one the most essential equation sets among the atmospheric simulation. Our main work and contributions are as follows:

—We first design a hybrid methodology that divides the computing domain into two parts and employs both the CPU host and FPGA accelerators to work in parallel. Through carefully adjusting the computational domains, we achieve a more balanced resource usage than Gan et al. [2013] and further improve the overall performance.

—Through decomposing the resource-demanding SWE kernels, we manage to map the double-precision algorithm into three FPGAs. Moreover, by using fixed-point and reduced-precision floating point method, we manage to build a fully pipelined mixed-precision design on a single FPGA, which can perform 428 floating-point and 235 fixed-point operations per cycle.
—The experimental results of the optimal multiple FPGA design demonstrate magnitude of improvement in both the performance and the power efficiency, and reveal great potential on applying FPGA platforms in atmospheric simulation.

The mixed-precision design with four FPGAs running together can achieve a speedup of 20 over a fully optimized design on a CPU rack with two eight-core processors and is 8 times faster than the fully optimized Kepler GPU design. As for the power efficiency, the mixed-precision design with four FPGAs is 10 times more power efficient than a hybrid CPU-GPU Tianhe-1A supercomputer node.

The rest of the article is organized as follows. Section 2 presents related work. Section 3 introduces the equations, discretization, and the CPU algorithm. The hybrid CPU-FPGA methodology is introduced in Section 4, followed by the description of the FPGA double-precision design in Section 5 and mixed-precision design in Section 6. Section 7 discusses the bandwidth requirement and the implementations. The experimental results and analysis is given in Section 8, and the article concludes in Section 9.

2. RELATED WORK

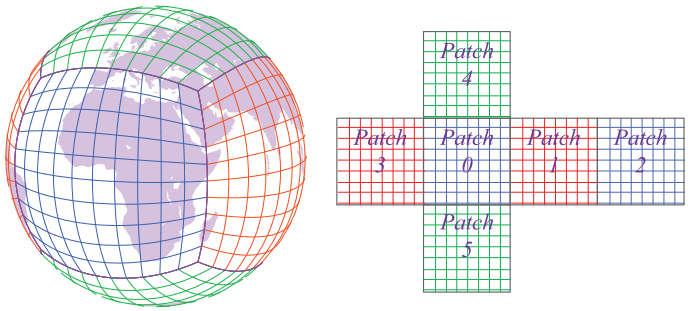
There are a number of atmospheric studies based on traditional platforms such as CPUs [Strand 2011; Skamarock et al. 2005; Johns et al. 2003] and GPUs [Henderson et al. 2011; Shimokawabe et al. 2010, 2011; Mielikainen et al. 2012; Yang et al. 2013]. Even though some experimental results demonstrate high speedup and high scalability, the constraint of data presentations, and the fact that traditional platforms generally consume much more power, bring a tough challenge as the atmospheric kernel becomes more computationally insensitive and the resolution becomes more fine grained.

In recent years, we start to see some promising results using FPGAs as accelerators in some key applications, such as the exploration geophysics Fu et al. [2012] and financial computing [Tse et al. 2010; Mingas and Bouganis 2012]. The high density of computing logics and the reconfigurable feature on data presentations provide big optimizing space to improve the performance.

There has been related work on mapping atmospheric simulation onto reconfigurable platforms. Smith et al. [2005] accelerate the Parallel Spectral Transform shallow water model using ORNL's SRC Computers. Only some subroutines (FFT or LT) are deployed on the FPGA, and a small speedup is gained over CPUs. Wilhelm [2012] analyzes a high-level approach for programming preconditioners for an ocean model in climate simulations on FPGAs but do not manage any actual acceleration. Oriato et al. [2012] accelerate a realistic dynamic core of the LAM model using FPGAs. It is a successful trial on reducing resource usage through fixed-point arithmetic.

Compared to traditional architectures, reconfigurable systems have their unique advantage in supporting mixed precisions. Significant performance improvement has been achieved in recent efforts on applying mixed-precision designs for Monte Carlo simulations [Mingas and Bouganis 2012; Chow et al. 2012].

For kernels with a specific error requirement, Lee et al. [2006] design MiniBit, a tool to optimize bit widths of fixed-point numbers. However, for numeric simulations that run for thousands of timesteps, such as the atmospheric simulation, it is difficult to determine the optimal bit width through analytic methods. In our work, we design a method that can choose the best data predictions while guaranteeing the final accuracy.



(a) The cubed-sphere mesh (b) The computational domain

Fig. 1. Mesh and computational domain.

3. GLOBAL SHALLOW WATER EQUATIONS

3.1. Equations and Discretization

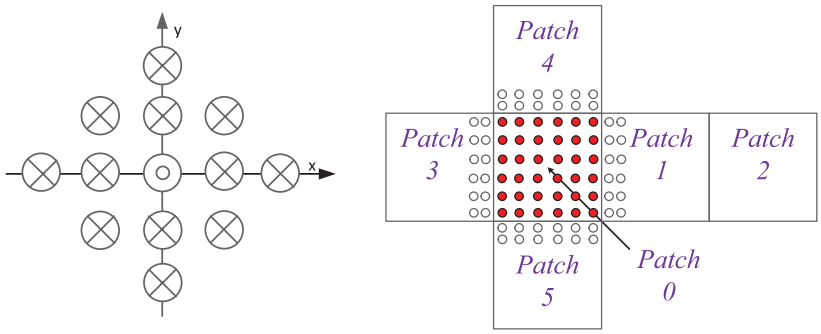
To simulate the atmospheric dynamics, there are several different equation sets available. SWEs are one of the most basic and important ones that are widely used as the test bed for designing new simulation methods. SWEs are a set of conservation laws to simulate the wave propagation and model the essential characteristics and dynamics of the atmosphere. We choose a gnomonic cubed-sphere mesh as our computational mesh in this article. The cubed-sphere mesh (Figure 1(a)) is obtained by mapping a cube to the surface of the sphere. The computational domain is then the six patches (Figure 1(b)), each of which is covered with rectangular meshes. Compared to other choices, such as the latitude-longitude mesh, the cubed-sphere mesh provides better load balance for pole regions.

When written in local coordinates, SWEs have an identical expression on the six patches, which is

$$\frac{\partial \boldsymbol{Q}}{\partial t} + \frac{1}{\Lambda}\frac{\partial(\Lambda u^1 \boldsymbol{Q})}{\partial x^1} + \frac{1}{\Lambda}\frac{\partial(\Lambda u^2 \boldsymbol{Q})}{\partial x^2} + S = 0, \tag{1}$$

where $(x^1, x^2) \in [-\pi/4, \pi/4]$ are the local coordinates, $\boldsymbol{Q} = (h, hu^1, hu^2)^T$ is the prognostic variable with h being the thickness of the atmosphere, and u^1, u^2 being the two horizontal velocity components. In Equation (1), the variable coefficient Λ depends only on (x^1, x^2) and is invariant with time. The source term S has a complicated form due to not only to the nonorthogonality of the cubed sphere but also the inclusion of a possibly nonflat bottom topography (see Yang et al. [2013] for more details).

To model the dynamics of the shallow water wave propagation in a certain period of time, we need to loop a certain number of timesteps to update the prognostic components (h, hu^1, and hu^2) of every mesh point. Spatially discretized with a cell-centered finite volume method and integrated with a second-order accurate TVD Runge-Kutta method [Gottlieb et al. 2001], solving Equation (1) is transformed into the computation of a two-dimension 13-point upwind stencil (Figure 2(a)). At each timestep and to get the prognostic components of the central point, the neighboring 12 points need to be accessed. For those points in the boundary of the patch, points from neighboring patches (the empty points in Figure 2(b), named as halo) need to be accessed. Because the six patches of the cubed-sphere mesh is not smoothly connected, we need to do a one-dimension interpolation to properly transfer those halo data between patches.



(a) The 13 point stencil (b) Mesh points (solid dots) and its halo meshes (empty dots)

Fig. 2. Stencil and halos.

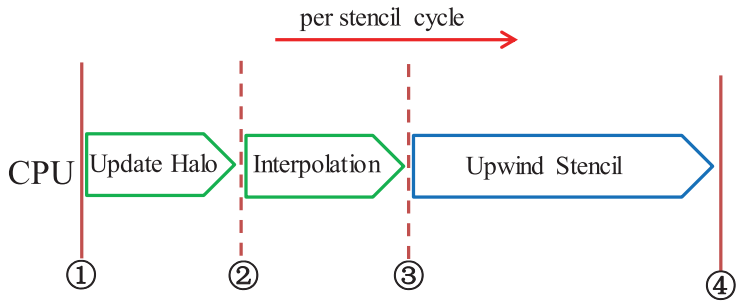
3.2. The CPU-Only Algorithm and Challenges

Algorithm 1 shows the CPU algorithm to solve the SWEs at each timestep. For each of the six cubed-sphere patches, halos must be updated first (line 2). Each patch needs to fetch the halo values from its four neighboring patches. We use the neighboring communication functions from the framework of PETSc (the Portable Extensible Toolkit for Scientific computation [Balay et al. 2013]) to help finish the update. Second, a linear interpolation (line 3) is carried out on the halo across patch interfaces to properly transfer halo information for stencil computations. We then do the stencil calculation (lines 4 through 8), which includes the computation of local coordinates based on global index j and i, and the computation of Flux variables, State Reconstruction, Reimann Solver, and Source Terms (h, hu^1, hu^2). The work flow of the CPU-only algorithm is shown in Figure 3(a), where all steps are doing in serial (from ① to ④).

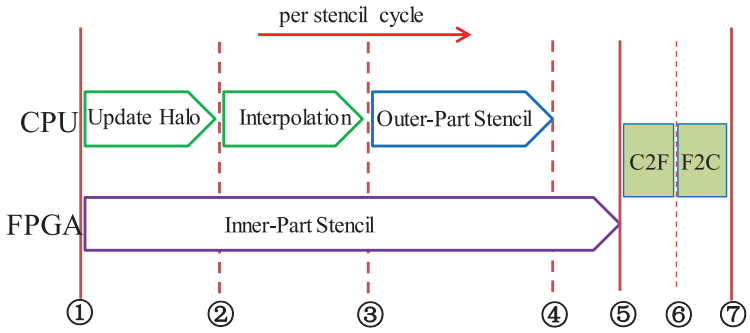
ALGORITHM 1: The CPU-only algorithm for each stencil cycle

```
1:  for all the six patches do
2:    Halo Updating
3:    Interpolations on halos when necessary
4:    for all the mesh cells in each patch do      //Upwind Stencil
5:      Compute Local Coordinate based on global index (j, i)
6:      Compute Flux, State Reconstruction, and Riemann Solver
7:      Compute Source Terms for h, hu^1, hu^2
8:    end for
9:  end for
```

The SWE algorithm brings serious design challenges for efficient solutions on the FPGA platform. Halo updating and interpolations bring data communication between patches. The communication must be carefully handled because it would be extremely heavy and greatly impact the overall performance when the mesh points increase to a large scale. Boundary interpolation also includes a lot of complex conditional statements, which would consume a lot of the limited FPGA resources. Moreover, although the upwind stencil from SWEs only involves 13 points (Figure 2(a)), the computational complexity is much higher than normal stencil kernels. To compute one mesh point, we will need at least 434 ADD/SUB operations, 570 multiplications, 99 divisions, 25 square roots, and 20 sine/cosine operations. The high arithmetic density and the irregular operations bring further challenge for the limited on-chip resources.



(a) CPU-only work flow.



(b) CPU-FPGA hybrid work flow. C2F: CPU to FPGA. F2C: FPGA to CPU.

Fig. 3. Workflow of different algorithms.

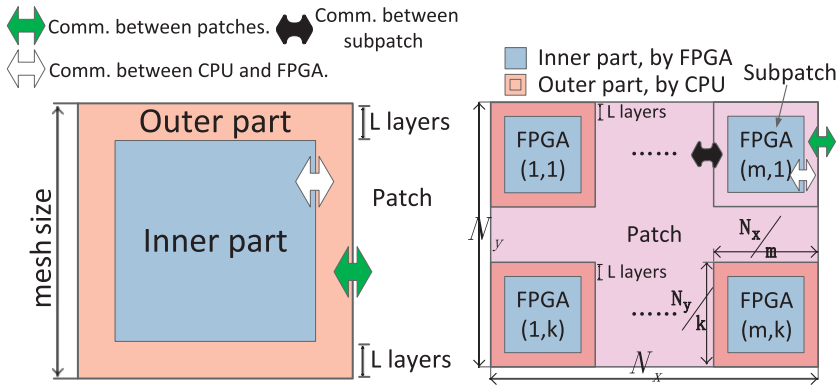
4. THE HYBRID CPU-FPGA DESIGN

4.1. Hybrid Domain Decomposition Methodology

Instead of deploying the whole computational domain into the FPGA, we design a hybrid algorithm that utilizes both the host CPU and the FPGA simultaneously. We decompose each of the cubed-sphere patches into the inner part and two layers ($L = 2$) of the outer part according to Figure 4(a). Then we can find that all halo exchanges ("Comm. between patches" arrow in Figure 4(a)) and boundary interpolation in Algorithm 1 only happen in the outer part. Therefore, we assign the CPU to process the outer part and assign the FPGA to perform the more regular inner-part stencil computation. Figure 3(b) shows the work flow of the hybrid design. The CPU will process the halo exchanges (① → ②), interpolations (② → ③), and the outer-part stencil computing (③ → ④), while simultaneously the FPGA will process the inner-part stencil computation (① → ⑤). When both the inner part and the outer part are finished (⑤), meshes along the inner-outer boundary will be exchanged (⑤ → ⑦) ("Comm. between CPU and FPGA" arrow in Figure 4).

Our proposed decomposition methodology has the following advantages:

—The CPU is now working simultaneously with the FPGA to solve the problem, which achieves an efficient usage of both computing resources.



(a) CPU and 1 FPGA (b) CPU and $m \times k$ FPGAs

Fig. 4. Domain decomposition of our hybrid CPU-FPGA algorithm.

—The CPU time for communication and computation can be hidden in FPGA computing, which achieves a well computation-communication overlapping.
—Specifically for SWEs, all complex conditional statements, which are expensive in resources for the FPGA to implement, are now assigned to the CPU to compute.

The hybrid algorithm can also be applied to platforms with multiple FPGAs. Supposing that we have a computing node with $m \times k$ FPGAs and multicore CPUs, and are handling a problem with the mesh size of $N_x \times N_y$ (Figure 4(b)), we first decompose the original patch into $m \times k$ subpatches so that the mesh size for each subpatch is $(N_x/m) \times (N_y/k)$. Here we assume that N_x and N_y can be divided exactly by m and k, respectively. Such inner patch decomposition will bring extra communication between each subpatch ("Comm. between subpatch" arrow in Figure 4(b)). Now we can find that a subpatch has the similar computational and communicating mechanism with the original patch, with only $1/(m \times k)$ of the computing area.

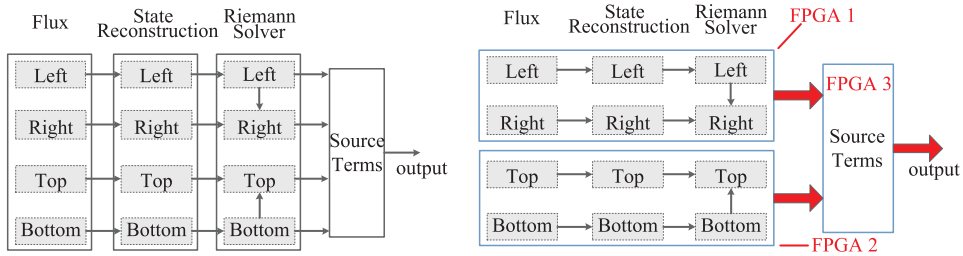
4.2. Balanced Task Partition

Based on the hybrid decomposition methodology, we can further improve performance by adjusting the area of the inner part and outer part to a balanced partition, where the CPU time and FPGA time are closest. The overall performance will increase accordingly. For example, when we set $L = 2$ in Figure 4, the FPGA time for processing the inner part is bigger than the CPU time for processing the outer part [Gan et al. 2013]. Therefore, we can carefully track the CPU and FPGA time according to the increase of the parameter L, and find the optimal point where the CPU time and FPGA time are closest. In this way, parameter L will be decided for the sake of balancing the computing time of the inner and outer parts rather than by the shape of the stencil (i.e., the thickness of the halo).

Based on different resolutions (i.e., the computing size of the problem), the optimal value of L can be different. For a certain resolution, the CPU performance is determined by the outer-part calculation as well as the communication and interpolation. We write a script to automatically search and record the performance of CPU processing the outer part according to different value of L, and compare it with the performance of FPGA for a similar value of L. The FPGA performance for processing the inner part is predicted based on the methodology proposed in Fu et al. [2013].

Table I. Resource Cost for Double Precision and the Designs with Different Optimizations

Resource		LUTs	FFs	BRAMs	DSPs
Double precision		299%	220%	20%	189%
Algorithm optimization		240%	176%	17%	149%
Algorithm decomposition	FPGA 1	62.43%	45.66%	7.77%	29.14%
	FPGA 2	64.24%	46.18%	7.71%	28.77%
	FPGA 3	69.18%	53.27%	27.07%	46.13%
Mixed precision		76.17%	53.41%	12.59%	44.84%



(a). Computational architecture (b). Algorithmic decomposition with 3 FPGAs

Fig. 5. Algorithmic decomposition design.

5. A DOUBLE-PRECISION FPGA DESIGN WITH DECOMPOSED KERNELS

5.1. Resource Analysis and Algorithmic Optimization

Based on the hybrid methodology in Section 4, the FPGA now only needs to process the more regular inner-part stencil.

The resource requirement for a straightforward double-precision version on Virtex-6 SX475T can be found in the second row of Table I. Except for the BRAMs, all of the other resources cannot satisfy the requirement of the double-precision design.

We first conduct some algorithmic optimizations to reduce the resource requirement. In Algorithm 1, local coordinate computation (line 5) only relates to global index j and i. Therefore, it can be precalculated during the compiling stage and stored in ROMs that are implemented with BRAMs. In this way, extra BRAMs are occupied in exchange for other more demanded resources. Another method is to erase the redundant computations. We extract all common factors that happen many times in the algorithm to avoid repeated calculations. For example, if factor X happens many times as a common denominator, we extract and precalculate the value of $1/X$ and multiply it with other factors when needed.

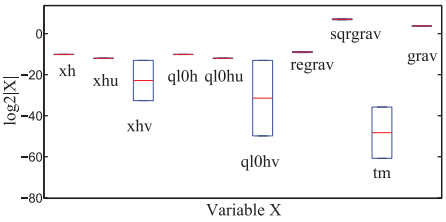
Although the preceding optimizations reduce the resource cost by 20% (third row in Table I), the resulting design is still too large to fit into one Virtex-6 SX475T FPGA.

5.2. Algorithmic Kernel Decomposition

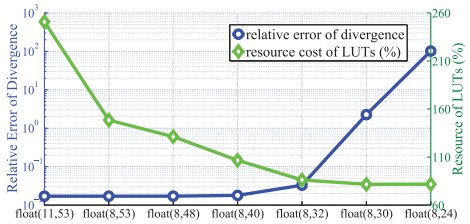
Even though the overall resource requirement is too high, we can decompose the SWE kernel into smaller subkernels and use multiple FPGAs to process them respectively.

When we perform the kernel decomposition, we explore different options to achieve the following two goals: (1) minimized intercommunications between the different subkernels to avoid the interlink between different FPGAs becoming the performance bottleneck, and (2) balanced resource costs of different subkernels so that we can achieve a balanced utilization of different FPGAs.

Figure 5(a) shows the computational architecture of the SWE algorithm. The steps of Flux, State Reconstruction, and Riemann Solver need to be performed for the components in four different directions (left, right, top, and bottom). Although the three



(a) Dynamic range of variable $log_2|X|$



(b) The relative error of divergence and resource cost of LUTs according to different floating-point bit-widths

Fig. 6. Mixed-precision design.

different steps (Flux, State Reconstruction, and Riemann Solver) need to be processed consecutively, the computation for the four different directions are relatively independent. The computation in different directions generally only depend on the variables and previous steps in the same direction. The only exception is the Riemann Solver step, where the right and top components depend on the left and bottom components, respectively. Therefore, to minimize the intercommunications, it is better to decompose the algorithm by directions rather than by different steps.

As for the resource consumption, the left, right, top, and bottom direction each account for around 35% of the resources on the Virtex-6 SX475T FPGA, and the Source Term computing requires nearly 70%. Therefore, to achieve a balanced utilization, a natural choice is to assign the Source Terms computing module to one FPGA and assign the four-direction computing to another two FPGAs. Considering that there are data connections between left and right, as well as between top and bottom, we put the left and right directions into one FPGA, and put the top and bottom directions into another FPGA (Figure 5(b)). The kernel decomposition approach now allows us to map the SWE algorithm into three FPGAs. The resource usage of different FPGAs after decomposition can be found in the fourth row of Table I.

6. A MIXED-PRECISION FPGA DESIGN

In this section, we use the mixed-precision method to decrease the resource requirement and deploy the whole SWE kernel into a single FPGA to improve the performance.

6.1. Range Analysis

Current FPGAs are generally more efficient for fixed-point arithmetic rather than floating-point arithmetic. Therefore, one strategy we take is to locate the region in the program that actually computes in a small range and then replace the region from floating-point arithmetic to fixed-point arithmetic.

For all of the different intermediate variables throughout the kernel, we first perform a range analysis to track the range of their absolute values. As shown in Figure 6(a), whereas some variables (e.g., xhv, $ql0hv$, and tm) cover a wide dynamic range, some other variables (e.g., xh, xhu, $ql0h$, and $ql0hu$) only change within a small range. As those variables all locate in the process of State Reconstructions, we can extract the four-direction State Reconstruction parts and use a fixed-point data type in that module. As the values of all variables in the State Reconstructions are located in the range of $(2^{-20}, 2^{1})$, we set the fractional bit width to be 2, which is big enough to represent all variables.

Most variables in the remaining parts cover a wide range, which we then apply a reduced floating-point number to represent. As the maximum dynamic range of the

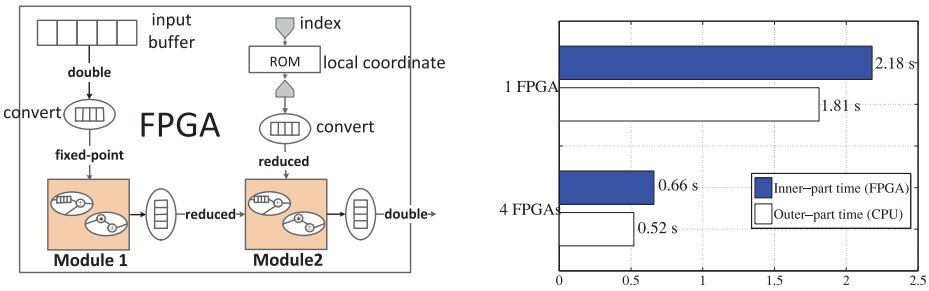


Fig. 7. (a) Mixed-Precision Design. Module1 (fixedpoint): Four-direction Reconstructions. Module2 (reduced floating-point): Four-directionRiemann, and the Source Terms. (b) Computation-communication overlapping with balanced partition.

base-two logarithmic values of those variables are smaller than 60, a floating-point with an eight-bit exponent would be good enough for representing the range.

6.2. Precision Analysis

As the SWE kernel generally involves a large number of iterations, it is difficult to achieve meaningful results through analytic precision analysis approaches due to the conservative assumptions. Therefore, in our approach, we determine the precision bit width through bit-accurate simulations for different bit-width configurations. Note that the simulation is performed based on the data of a typical benchmark scenario (zonal flow over an isolated mountain), which demonstrates the typical features of numerical atmospheric simulation.

To determine the mantissa bits, we explore a set of different bit widths from 53 to 24 and observe the dynamic trend of the relative error of *divergence* and the on-chip resource cost according to different floating-point bit-width configurations (Figure 6(b)).

The relative error of *divergence* is computed by comparing the simulated divergence against the standard dataset validated in Williamson et al. [1992] and can be used as an important indicator for the quick estimation of the accuracy. If the relative error is larger than 5%, the final result will no longer be true.

For brevity, hereafter float(e, m) denotes a floating point with e bits exponent and m bits mantissa, and fixed(i, f) denotes a fixed point with i bits integer and f bits fraction. From Figure 6(b), for float(8,53), float(8,48), and float(8,40) settings, we observe a similar relative error as the double-precision float(11, 53). For float(8,32), we can still achieve a relative error of around 2%. However, when we further reduce the precision to float(8,30), we see a surge of the relative error to a level that is far above the required 5%. The sharp accuracy reduction at float(8,30) indicates the precision threshold [Haohuan et al. 2009] of the SWEs. When the bit width of data keeps decreasing and cannot satisfy the precision threshold, the accuracy will break down sharply.

On the resource cost side, float(8,32) is also a suitable choice that reduces the LUT usage from around 240% to 80% of the total capacity of a Virtex-6 SX475T FPGA.

For the fixed-point variables in the Reconstruction parts, we apply a similar approach to determine the fractional bit width to be 38. Therefore, we pick float(8,32) and fixed(2,38) as the number representation in the algorithm.

6.3. Architecture of the Mixed-Precision Design

Through mixed-precision arithmetic, the resource requirement is greatly decreased (the fifth row of Table I), which enables us to fit the SWE kernel into one FPGA.

The general architecture of the mixed-precision design is shown in Figure 7(a). The input streams are originally in double precision and will later be converted into fixed

point and go through Module 1 for the all-direction State Reconstructions. Then it will be converted into a reduced-precision floating point and go through Module 2 for the computation of all-direction Riemann and the Source Term. During the computation, local coordinates are acquired through looking up the ROMs. After the computation is finished, the results will be converted back into double precision.

7. BANDWIDTH DISCUSSION AND IMPLEMENTATIONS

7.1. Bandwidth Requirement

In the hybrid algorithm, the FPGA only processes the inner-part points. Data streams will go through the FPGA DFE to finish the upwind stencil operation. The bandwidth requirement of an application would be

$$Band_r = S \times b \times f_{FPGA}, \tag{2}$$

where S refers to the total number of the streams that go through the DFE at each timestep, b refers to the number of bytes of the data type, and f_{FPGA} refers to the frequency of the FPGA. If the bandwidth of the network $Band_s$ can satisfy the bandwidth requirement, say

$$Band_s \geq Band_r. \tag{3}$$

It would be ideal so that all input and output streams can be prepared in one physical cycle. For cases that cannot satisfy Equation (3), we can either increase $Band_s$ or decrease $Band_r$. To improve $Band_s$, we can use the medium with higher accessing bandwidth to replace the original network. To decrease $Band_r$, we can either use the hardware (de)compression scheme [Fu et al. 2013] or optimize the algorithm to decrease the number of streams [Yang et al. 2013] and the data bytes. We usually do not decrease the frequency of the FPGA, as such behavior will slow down the physical cycle.

7.2. Experimental Platforms and Implementations

The designs proposed in Section 5 and Section 6 can be applicable to any host systems with FPGAs as accelerators. Here we use the MaxWorkstation and MaxNode FPGA platforms from Maxeler [Pell and Averbukh 2012] to implement the single and multiple FPGA designs, respectively.

The Maxeler MaxWorkstation [Maxeler 2011] is a small factor PC that brings the power of dataflow computing to the desktop. MaxWorkstation contains one Intel Core i7 quad-core CPU with 16GB RAM, and one DFE with a Virtex-6 SX475T FPGA and 24GB on-board memory (DRAM). The DFE connects to the CPU via a PCI Express gen2 x8 with a bandwidth of 8Gbytes/s. The MaxNode (MPC-C series in Maxeler [2011]) is a server-class HPC system and contains 12 Intel Xeon CPU cores and four DFEs. Each DFE has one Virtex-6 SX475 FPGA and 48GB on-board memory (DRAM), which are connected to each other through the MaxRing high-speed interconnect [Lindtjorn et al. 2010]. The DFEs connect to the CPU via a PCI Express gen2 x8. In both platforms, we use the general-purpose MaxCompiler development environment [Pell and Averbukh 2012] to program and optimize our designs.

We set the total mesh size of the SWEs to be $1024 \times 1024 \times 6$—that is, $N_x = N_y = 1024$. Therefore, mesh size per DFE for Workstation and MaxNode would be 1024×1024 and 512×512, respectively. For both designs, FPGA will only process the inner-part computing, whereas the host CPU will process the outer-part computing and halo updating. We also apply OpenMP in the CPU side to fully explore the multicore resources.

In terms of the balanced task partition proposed in Section 4.2, through tracking the performance based on different parameter L in Figure 4, we find that $L = 8$ (instead of $L = 2$ in Gan et al. [2013]) is the best choice for both the one FPGA and four FPGA designs to reach a more balanced partition and further improve the performance. The

result of the communication-computation overlapping is shown in Figure 7(b), from which we can find out that the CPU time is well overlapped by FPGA computing.

As for the bandwidth requirement, in the SWE DFE, there are 11 double-precision streams. Assuming that the FPGA runs at 100MHz, $Band_r = 8.8$ Gbytes/s according to Equation (2). If all data are stored in the host CPU, $Band_s$ is equal to the bandwidth of PCIe 2.0 (8 Gbytes/s), which cannot satisfy Equation (3). So we use the DRAM on DFE, which has a much higher accessing bandwidth (38 Gbytes/s) for the FPGA, to increase $Band_s$. In this way, we only need to perform the data exchange of the boundary part between the CPU and FPGA through the PCIe 2.0 interface.

8. EXPERIMENTS AND ANALYSIS

8.1. Benchmark Designs

Compared to our previous work [Gan et al. 2013], the benchmark designs in this article are based on more powerful platforms with the most state-of-the-art optimizations. Therefore, the performances are greatly improved.

The CPU design is based on a computing rack with two Intel E5-2650 (Sandy Bridge) CPUs. Each CPU has eight cores, and each core has two vector units. OpenMP multithreading, vectorization, and cache blocking are used to improve the performance.

The CPU-GPU design is based on two generations of different GPU cards. The Fermi M2050 GPU comes from Tianhe-1A, one of China's largest supercomputers with 7,168 computing nodes. Each Tianhe-1A node is equipped with two six-core Intel X5670 CPUs and one Fermi GPU. Our previous work [Yang et al. 2013] has scaled the SWEs to more than 3,750 Tianhe-1A nodes and achieved a performance of 803 TFlops in double precision. We also have a GPU node with the more advanced Kepler 20x GPU card. We have performed systematic optimizations [Yang et al. 2013] for both the GPU and CPU sides, including multithreading and GPU shared memory. The performance on those platforms is used here to be a comparison basis for our FPGA designs. Note that in this article we have optimized the SWE algorithm in Section 5.1, so we also apply those optimizations in our CPU and GPU code for a fair comparison.

8.2. Accuracy Validation

Our numerical test is based on a model problem—zonal flow over an isolated mountain—which is taken from the benchmark test set of Williamson et al. [1992]. The test runs in 100 timesteps, and the mesh size is fixed to $1024 \times 1024 \times 6$.

The numerical solutions of our programs are close in accuracy to the standard reference that has been validated in Yang et al. [2013]. We further use mass conservation—one of the most essential integral invariants in atmospheric simulation—to give a more concrete accuracy comparison. Mathematically, the discretization scheme that we employ leads to exact mass conservation. Due to the truncation error, the error of mass conservation is near to machine epsilon (i.e., around 10^{-14} in double precision). This conservation property can be further relaxed to, for instance, 10^{-11}, which indicates that at most 1% of total mass discrepancy is introduced after a billion timesteps. Figure 8 shows the mass relative error at each timestep. Double precision refers to the CPU standard version and the algorithm decomposition design. FPGA mixed refers to the mixed-precision algorithm, whose relative error of FPGA-mixed maintains smaller than 10^{-11}, and therefore satisfies the accuracy requirements. We also show the case of the single-precision FPGA version that does not satisfy the conservation requirement.

8.3. Performance and Power Efficiency

Table II shows the performance (evaluated by total mesh point processed per second) and the power efficiency (evaluated by performance per Watt) on different platforms. Note that the power consumption (Watt) in the single-node scenario is obtained through

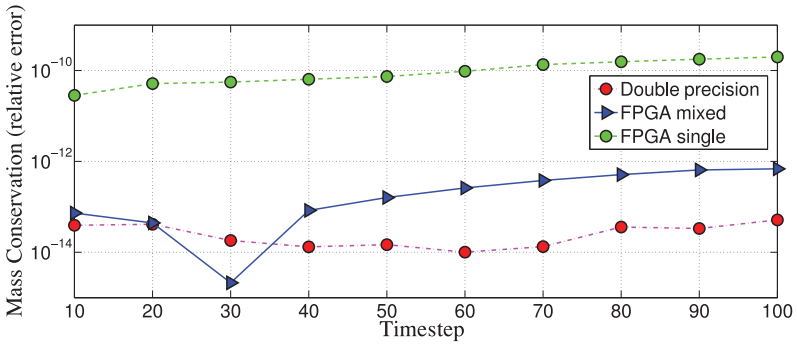


Fig. 8. Mass conservation. Values should be less than 10^{-11}.

Table II. Performance and Power Efficiency

Single Node Scenario (Mesh Size: 1024 × 1024 × 6)					
Platform	Performance (points/second)	Speedup	Power (Watts)	Efficiency (points/(second·Watt))	Power Efficiency
CPU rack	82K	1	377	217.5	1
Tianhe-1A node	110.38K	1.35×	360	306.6	1.4×
Kepler K20x	209K	2.55×	365	572.6	2.6×
Algorithm decomposition[a]	22.12K	**0.27×**	420	52.6	**0.24×**
MaxWorkstation[b]	481K (468.11K)[c]	**6×**	186	2.59K	**12×**
MaxNode[b]	1.59M (1.54M)[c]	**19.4×**	514	3.09K	**14.2×**
Supercomputer Scenario (Mesh Size: 25600 × 25600 × 6)					
Platform	Total Nodes	Performance (points/second)	Power (Watt)	Efficiency (points/(second·Watt))	Power Efficiency
Tianhe-A	3,750	413.93M	1.35M	306.6	1
MaxNodes	264	419.76M	135.7K	3.09K	**10×**

[a]Double-precision design in Section 5. [b]Mixed-precision design in Section 6.
[c]Value in parentheses refers to previous optimal performance in [Gan et al. 2013], without balanced partition.

the direct measurement using a power meter. For hybrid designs, the measured power consumption includes both the accelerators (GPU or DFE card) and the CPU cores that serve as the host controller and outer-part processor.

Due to the data exchange between decomposed kernels, the algorithm decomposition design in Section 5.2 is slower than the CPU design.

Utilizing the mixed-precision method greatly solves the bottleneck. The mixed-precision design with one Virtex-6 FPGA (MaxWorstation) gains 6 times speedup over the CPU rack with two eight-core Sandy Bridge CPUs, 4.4 times over a Tianhe-1A node, and 2.35 times over a Kepler 20× node. With four FPGAs running simultaneously, the performance of mixed-precision MaxNode gains a speedup of 19.4 over the CPU rack, 14.4 times over the Tianhe-1A node, and 8 times over a Kepler 20× node. The performance of MaxNode is equivalent to 14 nodes in the Tianhe-1A supercomputer.

As for the power efficiency, the mixed-precision design with MaxWorkStation is 8 times more power efficient than the Thianhe-1A node, and the mixed-precision design with MaxNode is up to 10 times more power efficient than a Tianhe-1A node.

Although we do not yet have a large-scale FPGA cluster to run a simulation in the same high resolution that we have managed to run on the Tianhe-1A, in the supercomputer scenario of Table II, we derive a projected performance on 264 MaxNodes, which achieves equivalent performance to 3,750 Tianhe-1A nodes. In this case, to run the same 25600 × 25600 resolution, the number of FPGA nodes and the power consumption is only 10% of the Tianhe-1A scenario.

8.4. Analysis and Discussions

Even though we have applied the latest CPU and GPU architectures that bring much higher computing power than that in Gan et al. [2013], and have performed the most state-of-the-art optimizations, the performance of SWEs on traditional CPU and GPUs still cannot surpass the FPGA design that is working in an even lower frequency.

The parallelism of CPU and GPU architectures is achieved through the scaling over the computing cores and the vector units. For the CPU design, the performance of SWEs is improved by 15 times through OpenMP multithreading and is further doubled through vectorization to reach the optimal record in Table II. For the GPU design, through carefully considering the block size and the configuration between shared memory and the L1 cache, we have achieved very high computing efficiency. The multiprocessors have been fully occupied (more than 99.99% monitored through the NVVP profiling tool). The performance of SWEs on Kepler is double the performance on Fermi, which mainly comes from the increased computing power of Kepler that has more streaming cores (192) in each of the increased streaming multiprocessor (SMX). Although the CPU and GPU is highly paralleled, the extremely high density of the SWE algorithm has decreased the computing efficiency and prevented the performance from further increasing. For the FPGA, it achieves parallelism through a deep pipeline of concurrency computing units. In our FPGA design, even though the FPGA device works at a low frequency of 100MHz, we manage to build the complex kernel on a fully pipelined FPGA card, which can efficiently perform 428 floating-point and 235 fixed-point operations per cycle. Meanwhile, the usage of DRAM in Section 7 manages to eliminate the bandwidth bottleneck between the CPU and FPGA. Therefore, all of the input data can be prepared to be ready at each cycle and go through the fully pipelined FPGA kernel.

As for the memory optimization, cache blocking in CPU and the usage of GPU L1 cache/shared memory have contributed to the improvement of cache behavior and the data reuse. However, the memory accessing pattern of the stencil evaluation brings a lot of cache misses, as the neighboring points are not sequentially stored. Execution units will have to spend more time on caching or memory addressing to access targeting data. The situation will become even worse with the increase of the simulating size. For our FPGA design, through customizing the BRAMs into a window buffer (shown as the input buffer in Figure 7(a)), we have achieved perfect data reuse in the data access [Fu and Clapp 2011] to efficiently eliminate the memory bottleneck.

In terms of the computation-communication overlapping achieved through the hybrid decomposition methodology and the balanced partition proposed in Section 4, it would be of great importance to hide the communication efficiently, especially in the large-scale simulation, when the data exchange becomes extremely heavy.

9. CONCLUSION

In this article, we propose a hybrid reconfigurable algorithm that employs both the CPU and FPGA to solve the global SWEs in parallel. Compared to the general FPGA methods, we have achieved more balanced utilization of both the CPU and FPGA computational resources and have gained a well computation-communication overlapping. By decomposing the resource-demanding SWE kernel, we manage to map the double-precision algorithm into three FPGAs. Moreover, we manage to reduce the great resource demand through a mixed-precision floating-point and fixed-point method and build the extremely complex upwind stencil into one FPGA card.

Platforms based on single and multiple FPGAs are employed to accelerate the performance. The experimental results of the optimal FPGA design demonstrate magnitude of improvement in both the performance and the power efficiency over the fully

optimized CPU and GPU programs, and reveal great potential in applying FPGA platforms in atmospheric simulation. Current and future work includes extending our designs to cover various algorithms for climate modeling and automating design exploration to enable rapid and efficient implementations.

REFERENCES

S. Balay, J. Brown, K. Buschelman, V. Eijkhout, W. Gropp, D. Kaushik, M. Knepley, L. Curfman McInnes, B. Smith, and H. Zhang. 2013. *PETSc Users Manual Revision 3.4.*

J. G. Charney and A. Eliassen. 1949. A numerical method for predicting the perturbations of the middle latitude westerlies. *Tellus* 1, 2, 38–54.

G. C. T. Chow, A. H. T. Tse, Q. Jin, W. Luk, P. H. W. Leong, and D. B. Thomas. 2012. A mixed precision Monte Carlo methodology for reconfigurable accelerator systems. In *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays*. ACM, New York, NY, 57–66.

H. Fu, W. Osborne, R. G. Clapp, O. Mencer, and W. Luk. 2009. Accelerating seismic computations using customized number representations on FPGAs. *EURASIP Journal on Embedded Systems* 2009, Article No. 3.

H. Fu and R. G. Clapp. 2011. Eliminating the memory bottleneck: An FPGA-based solution for 3d reverse time migration. In *Proceedings of the 19th ACM/SIGDA International Symposium on Field Programmable Gate Arrays*. ACM, New York, NY, 65–74.

H. Fu, R. G. Clapp, O. Lindtjorn, T. Wei, and G. Yang. 2012. Revisiting finite difference and spectral migration methods on diverse parallel architectures. *Computers and Geosciences* 43, 187–196.

H. Fu, L. Gan, R. Clapp, H. Ruan, O. Pell, O. Mencer, M. Flynn, X. Huang, and G. Yang. 2013. Scaling the reverse time migration performance through reconfigurable data-flow engines. *IEEE Micro* 34, 1, 30–40.

L. Gan, H. Fu, W. Luk, C. Yang, W. Xue, X. Huang, Y. Zhang, and G. Yang. 2013. Accelerating solvers for global atmospheric equations through mixed-precision data flow engine. In *Proceedings of the 23rd International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, Los Alamitos, CA, 1–6.

S. Gottlieb, C. W. Shu, and E. Tadmor. 2001. Strong stability-preserving high-order time discretization methods. *SIAM Review* 43, 1, 89–112.

T. Henderson, J. Middlecoff, J. Rosinski, M. Govett, and P. Madden. 2011. Experience applying Fortran GPU compilers to numerical weather prediction. In *Proceedings of the Symposium on Application Accelerators in High-Performance Computing (SAAHPC)*. IEEE, Los Alamitos, CA, 34–41.

T. C. Johns, J. M. Gregory, W. J. Ingram, C. E. Johnson, A. Jones, J. A. Lowe, J. F. B. Mitchell, D. L. Roberts, D. M. H. Sexton, D. S. Stevenson, S. F. B. Tett, and M. J. Woodage. 2003. Anthropogenic climate change for 1860 to 2100 simulated with the HadCM3 model under updated emissions scenarios. *Climate Dynamics* 20, 6, 583–612.

D. U. Lee, A. A. Gaffar, R. C. C. Cheung, O. Mencer, W. Luk, and G. A. Constantinides. 2006. Accuracy-guaranteed bit-width optimization. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 25, 10, 1990–2000.

O. Lindtjorn, R. G. Clapp, and M. J. Flynn. 2010. Surviving the end of scaling of traditional micro processors in HPC. In *Proceedings of HOT CHIPS* 22.

Maxeler. 2011. Maxeler Products. Retrieved February 25, 2015, from http://www.maxeler.com/products/.

J. Mielikainen, B. Huang, H. Huang, and M. D. Goldberg. 2012. GPU acceleration of the updated Goddard shortwave radiation scheme in the weather research and forecasting (WRF) model. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing* 5, 2, 555–562.

G. Mingas and C. Bouganis. 2012. A custom precision based architecture for accelerating parallel tempering MCMC on FPGAs without introducing sampling error. In *Proceedings of the 20th Annual Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, Los Alamitos, CA, 153–156.

D. Oriato, S. Tilbury, M. Marrocu, and G. Pusceddu. 2012. Acceleration of a meteorological limited area model with dataflow engines. In *Proceedings of the Symposium on Application Accelerators in High Performance Computing (SAAHPC)*. IEEE, Los Alamitos, CA, 129–132.

O. Pell and V. Averbukh. 2012. Maximum performance computing with dataflow engines. *Computing in Science and Engineering* 14, 4, 98–103.

T. Shimokawabe, T. Aoki, J. Ishida, K. Kawano, and C. Muroi. 2011. 145 TFlops performance on 3990 GPUs of TSUBAME 2.0 supercomputer for an operational weather prediction. *Procedia Computer Science* 4, 1535–1544.

T. Shimokawabe, T. Aoki, C. Muroi, J. Ishida, K. Kawano, T. Endo, A. Nukada, N. Maruyama, and S. Matsuoka. 2010. An 80-fold speedup, 15.0 TFlops full GPU acceleration of non-hydrostatic weather model ASUCA production code. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis (SC)*. IEEE, Los Alamitos, CA, 1–11.

W. C. Skamarock, J. B. Klemp, J. Dudhia, D. O. Gill, D. M. Barker, W. Wang, and J. G. Powers. 2005. *A Description of the Advanced Research WRF Version 2*. Technical Report. DTIC Document.

M. C. Smith, J. S. Vetter, and X. Liang. 2005. Accelerating scientific applications with the SRC-6 reconfigurable computer: Methodologies and analysis. In *Proceedings of the 19th IEEE International Parallel and Distributed Computing Symposium*. IEEE, Los Alamitos, CA, 157b.

G. Strand. 2011. Community earth system model data management: Policies and challenges. *Procedia Computer Science* 4, 558–566.

A. H. T. Tse, D. B. Thomas, K. H. Tsoi, and W. Luk. 2010. Reconfigurable control variate Monte-Carlo designs for pricing exotic options. In *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, Los Alamitos, CA, 364–367.

F. Wilhelm. 2012. *Parallel Preconditioners for an Ocean Model in Climate Simulations*. Ph.D. Dissertation. Karlsruher Institut für Technologie, Karlsruhe, Germany.

D. L. Williamson, J. B. Drake, J. J. Hack, R. Jakob, and P. N. Swarztrauber. 1992. A standard test set for numerical approximations to the shallow water equations in spherical geometry. *Journal of Computational Physics* 102, 1, 211–224.

C. Yang, W. Xue, H. Fu, L. Gan, L. Li, Y. Xu, Y. Lu, J. Sun, G. Yang, and W. Zheng. 2013. A peta-scalable CPU-GPU algorithm for global atmospheric simulations. In *Proceedings of the 18th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*. ACM, New York, NY, 1–12.

Received December 2013; revised March 2014; accepted March 2014