ACCELERATING SOLVERS FOR GLOBAL ATMOSPHERIC EQUATIONS THROUGH MIXED-PRECISION DATA FLOW ENGINE

Lin Gan[1,2,3]*, Haohuan Fu*[2]*, Wayne Luk*[4]*, Chao Yang*[5]*, Wei Xue*[1,2,3]*,*
Xiaomeng Huang[2]*, Youhui Zhang*[1,3]*, and Guangwen Yang*[1,2,3]
1.Department of Computer Science and Technology, Tsinghua University
2.Ministry of Education Key Lab. for Earth System Modeling, Center for Earth System Science, Tsinghua University
3.Tsinghua National Laboratory for Information Science and Technology (TNList)
4.Department of Computing, Imperial College London
5.Institute of Software, Chinese Academy of Sciences
email: l-gan11@mails.tsinghua.edu.cn, haohuan@tsinghua.edu.cn, w.luk@imperial.ac.uk, yangchao@iscas.ac.cn
{xuewei,hxm,zyh02,ygw}@tsinghua.edu.cn

ABSTRACT

One of the most essential and challenging components in a climate system model is the atmospheric model. To solve the multi-physical atmospheric equations, developers have to face extremely complex stencil kernels. In this paper, we propose a hybrid CPU-FPGA algorithm that applies single and multiple FPGAs to compute the upwind stencil for the global shallow water equations. Through mixed-precision arithmetic, we manage to build a fully pipelined upwind stencil design on a single FPGA, which can perform 428 floating-point and 235 fixed-point operations per cycle. The CPU-FPGA algorithm using one Virtex-6 FPGA provides 100 times speedup over a 6-core CPU and 4 times speedup over a hybrid node with 12 CPU cores and a Fermi GPU card. The algorithm using four FPGAs provides 330 times speedup over a 6-core CPU; it is also 14 times faster and 9 times more power efficient than the hybrid CPU-GPU node.

1. INTRODUCTION

Due to the climate's significant influence on human activities and the huge losses caused by extreme weather events (at least 485 billion USD a year for US alone [1]), climate change has been one of the most important research subjects among governments and research organizations. Meanwhile, the complexity of the climate system makes computer-based modeling the only method to study climate changing mechanisms and make predictions into the future. Among all the different parts in a climate system model, the global atmospheric model is one of the most challenging components. Developers have to face the complex equation sets that bring tough challenges for the computing capability of different platforms.

Current high performance computing (HPC) platforms have already provided peta-scale performance for highly-scalable applications. However, they have to face the constraints of memory and communication bandwidth when dealing with applications with complex computation and heavy communication. Reconfigurable dataflow engines achieve parallel performance through a deep pipeline of thousands of concurrent operations, and have achieved high performance in many application fields, such as exploration geophysics [2] and financial computing [3]. Moreover, the support for mixed precision brings an additional optimization space and potential for extra performance boost.

In this paper, we propose a hybrid CPU-FPGA algorithm to solve the global shallow water equations, which demonstrate most of the essential dynamics of the atmosphere.

Our major contributions are:

• a hybrid domain decomposition that achieves efficient utilization of both CPU and FPGA to compute the complex upwind stencil (Section 3);

• a mixed-precision floating-point and fixed-point design that fits the resource-demanding SWE stencil kernel into one Virtex-6 FPGA (Section 4).

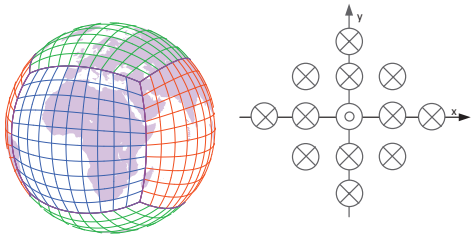
• significant improvements in performance and power efficiency over multi-core CPU and hybrid CPU-GPU platforms (Section 5).

2. BACKGROUND

2.1 Related Work. In recent years, we start to see some FPGA-based acceleration for modules within a global model ([4],[5]), and for regional weather predictions ([6]).

Smith *et al.* [4] accelerate the Parallel Spectral Transform shallow water model using ORNL's SRC Computers. Only some subroutines (FFT or LT) is deployed on the FPGA and a small speedup is gained over CPUs. Wilhelm *et al.* [5] analyze a high-level approach for programming preconditioners for an ocean model in climate simulations on FPGAs but do not manage actual acceleration. Oriato *et al.* [6] accelerate a realistic dynamic core of LAM model using FPGAs. It is a successful trial on reducing resource usage through fixed-point arithmetic. However, LAM is a simpli-

978-1-4799-0004-6/13/$31.00



(a) Cubed-sphere mesh. (b) The 13 point stencil.

Fig. 1. The Mesh and stencil in the SWE algorithm.

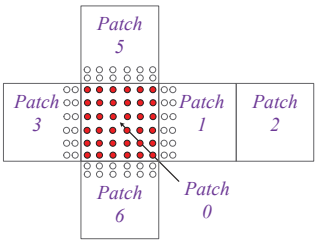


Fig. 2. Mesh points (solid dots) to be calculated inside a patch and its halo meshes (empty dots) from other patches.

fied weather prediction model that only covers regional area. In contrast, our work targets on accelerating a more complex atmospheric kernel in a global scale.

Compared with conventional 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 ([7], [8]).

Another important issue for mixed-precision designs is how to choose the optimal precision that can both maximize the performance and satisfy the accuracy requirement. For kernels with a specific error requirement, Lee *et al.* [9] design MiniBit, a tool to optimize bit widths of fixed-point numbers. However, for numeric simulations that run for thousands of time steps, such as in the atmospheric simulation, it is difficult to determine the optimal bit width through analytic methods.

2.2 Equations and Discretization. Shallow water equations (SWEs) are a set of conservation laws to simulate the wave propagation and model the essential characteristics of the atmosphere. We choose cubed-sphere mesh as the computational mesh, which is obtained by mapping a cube to the surface of the sphere (Figure 1(a)).

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

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

where $(x^1, x^2) \in [-\pi/4, \pi/4]$ are the local coordinates, $Q = (h, hu^1, hu^2)^T$ is the prognostic variable, $F^i = u^i Q$ $(i = 1, 2)$ are the convective fluxes, S is the source term.

Spatially discretized with a cell-centered finite volume method and integrated with a second-order accurate TVD Runge-Kutta method [10], SWE solvers are transformed into the computation of a 13-point upwind stencil (Figure 1(b)). To get the prognostic components (h, hu^1 and hu^2) of the central point, its neighboring 12 points need to be accessed.

2.3 SWE Algorithm and Challenges. Figure 2 shows the six patches of the cubed-sphere. For each patch, to calculate all the mesh points (solid dots), we should store two more layers of meshes (empty points, named as halo meshes) from its adjacent patches. The SWE Algorithm is shown in Algorithm 1. First, halos must be updated. Each patch needs to fetch the halo values from its four neighboring patches.

Second, we do the stencil calculation. For each point, we first compute its local coordinate and then compute the Fluxes in four directions (left, right, bottom, top). In Algorithm 1, we explain the computation of the left Flux in detail (line 7-9), including Boundary Interpolation, State Reconstruction and Riemann Solver. The other three directions have the similar operations. In the end, we compute the Source terms and gather Fluxes and Sources (line 12 in Algorithm 1).

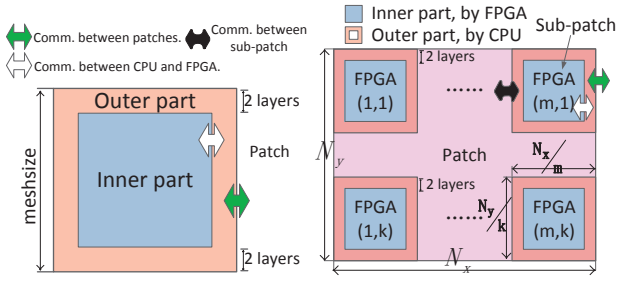
Algorithm 1 The SWE Algorithm

```
1:  for patch 0 to patch 5 do
2:      Halo Updating
3:      for j ← 0 to n_j do                                  ▷ Upwind Stencil
4:          for i ← 0 to n_i do
5:              Compute Local Coordinate
6:              Compute Left Flux, Including{
7:                  if( on Left Boundary) {Boundary Interpolation}
8:                  State Reconstruction
9:                  Riemann Solver
10:             }
11:             Compute Right, Bottom, Top Fluxes Including Interpolation
12:             Compute Source Terms for h, hu^1, hu^2
13:         end for
14:     end for
15: end for
```

Efficient solutions of SWEs bring serious design challenges. Halo exchange brings data communication among patches. Boundary Interpolation includes a lot of complex conditional statements which consume a lot of the limited FPGA resources. Moreover, although the upwind stencil from SWEs only involves 13 points (Figure 1(b)), 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.

3. CPU-FPGA HYBRID DESIGNS

3.1 Domain Decomposition. Instead of deploying the whole SWEs algorithm into the FPGA, we design a hybrid algorithm that utilizes both the host CPU and the FPGA.



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

Fig. 3. Domain decomposition of our hybrid CPU-FPGA algorithm.

We decompose each of the cubed-sphere patch into the outer part that includes two layers of boundary area and the inner part (Figure 3(a)). Then we can find that all the halo exchanges (Comm. between patches arrow) and boundary interpolation in Algorithm 1 only happen in the outer part. Therefore, we assign CPU to process the outer part and assign FPGA to perform the more regular inner-part computation. Shown in Algorithm 2, the CPU will process the halo exchanges and the boundary computing while FPGA only needs to process the inner-part stencil computation. When both the inner part and the outer part are finished, meshes along the inner-outer boundary will be exchanged (Comm. between CPU and FPGA arrow in Figure 3). Unlike traditional mechanism in which the CPU is usually idle while the FPGA is working, CPU keeps working in our algorithm. Moreover, with careful adjustment, the CPU time for communication and calculations can be overlapped by the FPGA computing. Such computation communication overlapping will be essential for atmospheric simulations to hide the heavy communications when mesh points increase to a large scale.

Algorithm 2 the CPU-FPGA Hybrid Algorithm

```
for patch 0 to patch 5 do
    CPU:
    for mesh points of the outer part do
        Halo Updating
        Compute Local Coordinate
        Boundary Interpolation (Left, Right, Bottom, Top)
    end for
    FPGA:(simultaneously with CPU)
    for mesh points of the inner part do
        Compute Local Coordinate
        Compute Reconstruction and Riemann (Left, Right, Bottom, Top)
        Compute Source Terms for h,hu^1,hu^2
    end for
    CPU-FPGA Exchange
end for
```

The hybrid algorithm can also be applied to platforms with multiple FPGAs. Supposing we have a computing node with $m \times k$ FPGAs and multi-core CPUs, and are handling the meshsize of $N_x \times N_y$, we first decompose the original patch into $m \times k$ sub-patches (Figure 3(b)), so that the meshsize for each sub-patch is $(N_x/m) \times (N_y/k)$. Here we assume that the N_x and N_y can be divided exactly by m and k, respectively. Such inner patch decomposition will bring extra communications between each sub-patches (Comm. between sub-patch arrow in Figure 3). Now we can find that a sub-patch has the similar computational and communicating mechanism with the original patch, with only $1/(m \times k)$ of the computing area.

3.2 Bandwidth Requirement. In the hybrid algorithm, FPGA only processes the inner-part points. Data streams will go through the FPGA data flow engine 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 data flow engine at each time step, 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 the input and output streams can be prepared in one physical cycle. For cases that can not satisfy Equation (3), we can either increase $Band_s$, or decrease $Band_r$. To improve $Band_s$, we can use medium with higher accessing bandwidth to replace the original network. To decrease $Band_r$, we can optimize the applications to decrease the number of streams and the data bytes. We usually do not decrease the frequency of FPGA since such behavior will slow down the physical cycle.

3.3 Implementation. The hybrid algorithm can be applicable to any host systems with FPGAs as accelerators. Here we use the Maxeler FPGA platforms [11] to implement it.

We have a MaxWorkstation, which includes one Intel i7 quad-core CPU and one accelerating board with a Virtex-6 SX475T FPGA and 24GB on-board memory (DRAM). We also have a MaxNode, which consists of 12 Intel Xeon CPU cores and four accelerating boards.

We set the meshsize of the SWE patch to be 1024×1024, *i.e.* $N_x = N_y = 1024$. For the Workstation, the single FPGA will process the whole mesh. For the MaxNode, we further decompose the patch into 2×2 sub-patches, each with a meshsize of 512×512.

For both designs, FPGA will only process the inner-part computing, while the host CPU will process the outer-part computing and halo updating. We also apply OpenMP in the CPU side to fully explore the multi-core resources.

As for the bandwidth requirement, in the SWE data flow engine, there are 11 double-precision streams. Assuming the FPGA runs at 100 MHz, $Band_r = 8.8$ Gbytes/s according to Equation (2). If all data are stored in the host CPU, $Band_s$ equals to the bandwidth of PCIe 2.0 (8 Gbytes/s), which cannot satisfy Equation (3). So we use the

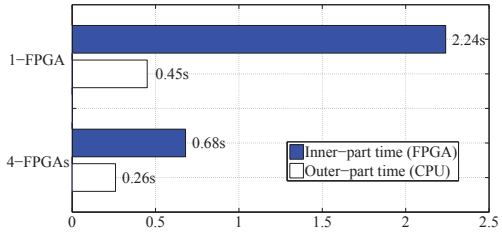


Fig. 4. Communication-computation overlapping.

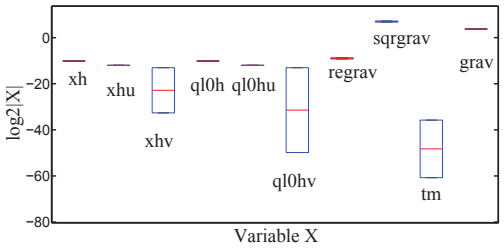


Fig. 5. Dynamic range of variable $log_2|X|$.

Table 1. The resource cost for the baseline FPGA design and the designs with different algorithmic optimizations.

Resource	FPGA-baseline	Rom-based	Extract-CF
LUTs	299%	283%	240%
FFs	220%	210%	176%
BRAMs	20%	23%	17%
DSPs	189%	178%	149%

DRAM, which has a much higher accessing bandwidth (38 Gbytes/s) for the FPGA, to increase $Band_s$ in Equation (3). 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.

The results of the computation communication overlapping are shown in Figure 4. The CPU time for handling the outer-part communications and calculations (blank bar) is completely overlapped by the FPGA operation (blue bar).

4. MIXED-PRECISION FPGA DESIGN

4.1 Algorithmic Optimizations. The resources required for a straightforward double-precision version on Virtex-6 SX475T can be found in Table 1 (shown as FPGA-baseline). Except for BRAMs, other resource requirements are far beyond what the FPGA can supply.

We therefore conduct some algorithmic optimizations to reduce the requirement for resources. In Algorithm 1, local coordinate computation only relates to index i or j. Since i and j are looped from 0 to n_i or n_j, the local coordinate can be pre-calculated during the compiling stage and stored in ROMs, which are implemented with BRAMs. In this way, extra BRAMs are occupied in exchange for other more demanded resources (Rom-based column in Table 1). Another method is to erase the redundant computations (Extract-CF column in Table 1). We extract all the common factors that happen many times in the algorithm to avoid repeated calculations. For example, if factor X happens many times as common denominator, we extract and pre-calculate the value of $1/X$, and multiply it with other factors when needed.

While the above optimizations reduce the resource cost by 20%, the resulting design is still too large to fit into one Virtex-6 SX475T FPGA. We therefore design a mixed-precision algorithm to reduce the resource demand.

4.2 Precision Optimizations. In the SWE kernel, although the overall requirement for data precision is high, variables in different parts of the program demonstrate different range and precision behaviors. We can therefore explore the design space of using different number representations and precisions at different parts.

4.2.1 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 replace the region from floating-point arithmetic to fixed-point arithmetic.

For all 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 5, while some variables (*e.g.*, $xhv, ql0hv$, and tm) cover a wide dynamic range, some other variables (*e.g.*, xh, xhu, $ql0h$, $ql0hu$) only change within a small range. As those variables all locate in the process of State Reconstructions, we can extract the four-direction Reconstruction parts, and use fixed-point data type in that module. Most variables in the remaining parts (four-direction Riemann and the Sources Terms) cover a wide range, which we can then apply reduced floating-point number to represent.

Another discovery is that the maximum dynamic range of the base-two logarithmic values of the variables would be smaller than 60. Therefore, floating-point with 8-bit exponent would be good enough for representing the range.

4.2.2 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 floating-point mantissa bits, we explore

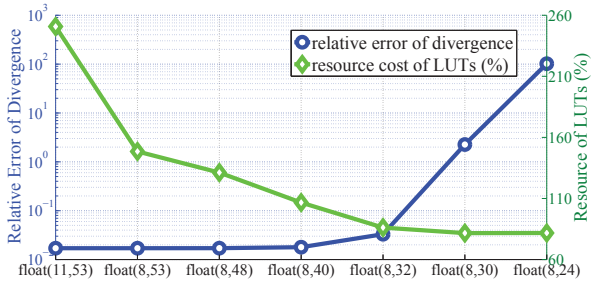


Fig. 6. The relative error of divergence and resource cost of LUTs according to different floating-point bit-widths.

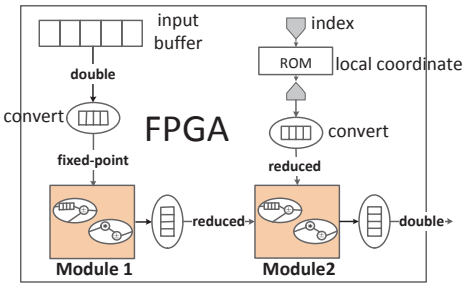


Fig. 7. General Architecture of Mixed-Precision Design. Module1 (fixed-point): Left, Right, Top, Bottom Reconstructions. Module2(reduced floating-point): Left, Right, Top, Bottom Riemann, and the Source Terms

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).

The relative error of $divergence$ is computed by comparing the simulated divergence against the standard data set validated in [12], 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 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.

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%.

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

Based on the above considerations, we pick float(8,32) as the number representations in the algorithm. For the fixed-point variables in the Reconstruction parts, we apply a similar approach to determine the fractional bit-width to be 38.

4.3 The Architecture of the Mixed-Precision Design. The general architecture of the mixed-precision design is shown in Figure 7. The input streams are originally in double precision, and will be later converted into fixed-point and go through Module 1 for the all-direction State Reconstructions. Then it will be converted into 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.

Through mixed-precision floating point and fixed-point arithmetic, the resource usage of LUTs, FFs, BRAMs and DSPs reduce to 76.17%, 53.41%, 12.59% and 44.84%, which enables us to fit one complete SWE kernel into one FPGA.

5. RESULTS AND ANALYSIS

5.1 Reference Designs. The CPU-GPU design is based on Tianhe-1A, China's largest supercomputer with 7168 computing nodes. Each Tianhe-1A node is equipped with two six-core Intel X5670 CPUs and one NVIDIA M2050 GPU. We have performed systematic optimizations [13] for both the GPU and CPU sides, including OpenMP multi-threading and GPU shared memory. The performance on Tianhe-1A is used here to be a comparison basis for our FPGA designs. Note that we have in this paper optimized the SWEs algorithm in Section 4.1, so we also apply those optimizations in our CPU code for a fair comparison.

5.2 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.* [12]. The test runs in 100 time steps, and the meshsize is fixed to $1024 \times 1024 \times 6$.

The numerical solutions of our programs are close in accuracy to the standard reference which has been validated in [13]. We further use mass conservation, one of the most essential integral invariants in atmospheric simulation, to give a more concrete accuracy comparison. Theoretically the relative error of mass should be zero, conservative with the previous time step. However, considering influences from hardware precision, a relatively small difference (less than 10^{-11}) is acceptable. Figure 8 shows the mass relative error at each time step. CPU-double refers to the CPU standard version. 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.

5.3 Performance and Power Efficiency. Table 2 shows the performance (total mesh point processed per second) and the power efficiency measured on different platforms .

The performance of CPU-FPGA algorithm using one Virtex-6 FPGA (MaxWorstation) gains 100 times speedup

Table 2. Performance and power efficiency for different platforms

Mesh size: $1024 \times 1024 \times 6$					
platform	performance (points/second)	speedup	power (Watt)	efficiency (points/(second·Watt))	power efficiency
6-core CPU	4.66K	1	225	20.71	1
Tianhe-1A node	110.38K	23x	360	306.6	14.8x
MaxWorkstation	468.11K	**100x**	186	2.52K	**121.6x**
MaxNode	1.54M	**330x**	514	3K	**144.9x**

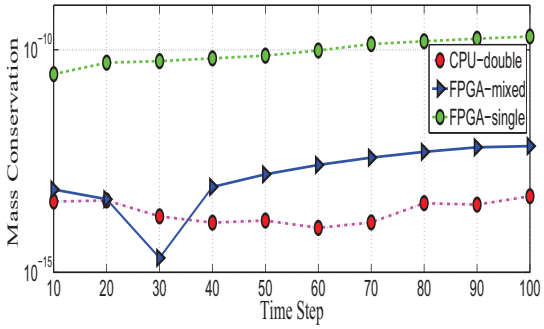


Fig. 8. Mass Conservation. Values should be less than 10^{-11}.

over 6-core CPU and 4 times over a Tianhe-1A node with 12 CPU cores and a 448-cores GPU. With 4 FPGAs running simultaneously, the performance of MaxNode gains a speedup of 330 over a 6-core CPU and 14 times over the Tianhe-1A node. The performance of MaxNode is equivalent to 14 nodes in the Tianhe-1A supercomputer.

Even though the FPGA device works at a frequency of 100MHz, we manage to build the complex kernel on a fully-pipelined FPGA card, which can perform 428 floating-point and 235 fixed-point operations per cycle. Meanwhile, the fully-pipelined design also provides much higher efficiency than that of the CPU and GPU based platforms. The combination of the high parallelism and the high efficiency leads to the ultra-high performance of our design.

As for the power efficiency (evaluated by the performance per watt), our CPU-FPGA algorithm with 4 FPGAs is up to 9 times more power efficient than a Tianhe-1A node.

6. CONCLUSION

In this paper, we propose a hybrid CPU-FPGA algorithm to solve the global SWEs. Platforms based on single and multiple FPGAs are employed to scale the performance. We manage to reduce the great resource demand through mixed-precision floating-point and fixed-point method and build the extremely complex upwind stencil into one FPGA card.

Our work manages to achieve significant acceleration through employing FPGAs to solve the global atmospheric equations. The results show great potential in utilizing FPGA for the state-of-the-art global atmosphere study.

Acknowledgment

This work is supported in part by the National High-Tech R&D (863) Program of China under grant 2011AA01A203, the 973-Program of China under grant 2010CB951903, the NSFC US-China Collaborative Software Research and the NSF China under grant 51190101 and 61073165.

We would also like to thank Maxeler Technologies for providing the MAX devices.

7. REFERENCES

[1] J. Lazo, M. Lawson, P. Larsen, and D. Waldman, "US economic sensitivity to weather variability," *Bulletin of the American Meteorological Society*, p. 709, 2011.

[2] H. Fu, R. Clapp, O. Lindtjorn, T. Wei, and G. Yang, "Revisiting finite difference and spectral migration methods on diverse parallel architectures," *Computers & Geosciences*, 2012.

[3] A. Tse, D. Thomas, K. Tsoi, and W. Luk, "Reconfigurable control variate Monte-Carlo designs for pricing exotic options," in *FPL 2010*, 2010, pp. 364–367.

[4] M. C. Smith, J. S. Vetter, and X. Liang, "Accelerating scientific applications with the SRC-6 reconfigurable computer: Methodologies and analysis," in *IPDPS*. IEEE, 2005, pp. 157b–157b.

[5] D.-M. F. Wilhelm and N. ad Weinstraße, "Parallel preconditioners for an ocean model in climate simulations."

[6] D. Oriato, S. Tilbury, M. Marrocu, and G. Pusceddu, "Acceleration of a Meteorological Limited Area Model with Dataflow Engines," in *2012 Symposium on SAAHPC*, 2012, pp. 129–132.

[7] G. Mingas and C. Bouganis, "A Custom Precision Based Architecture for Accelerating Parallel Tempering MCMC on FPGAs without Introducing Sampling Error," in *FCCM 2012*, 2012, pp. 153–156.

[8] G. Chow, A. Tse, Q. Jin, W. Luk, P. Leong, and D. Thomas, "A mixed precision Monte Carlo methodology for reconfigurable accelerator systems," in *Proc. FPGA*, 2012, pp. 57–66.

[9] D. Lee, A. Gaffar, R. Cheung, O. Mencer, W. Luk, and G. Constantinides, "Accuracy-guaranteed bit-width optimization," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,*, vol. 25, no. 10, pp. 1990–2000, 2006.

[10] S. Gottlieb, C. Shu, and E. Tadmor, "Strong stability-preserving high-order time discretization methods," *SIAM review*, vol. 43, no. 1, pp. 89–112, 2001.

[11] O. Pell and V. Averbukh, "Maximum Performance Computing with Dataflow Engines," *Computing in Science & Engineering*, pp. 98–103, 2012.

[12] D. Williamson, J. Drake, J. Hack, R. Jakob, and P. Swarztrauber, "A standard test set for numerical approximations to the shallow water equations in spherical geometry," *Journal of Computational Physics*, vol. 102, no. 1, pp. 211–224, 1992.

[13] C. Yang, W. Xue, H. Fu, L. Gan, L. Li, Y. Xu, Y. Lu, J. Sun, G. Yang, and W. Zheng, "A peta-scalable CPU-GPU algorithm for global atmospheric simulations," in *Proceedings of PPoPP'13*, 2013, pp. 1–12.