A Highly-Efficient and Green Data Flow Engine for Solving Euler Atmospheric Equations

Lin Gan[1,2,3], *Haohuan Fu*[1], *Chao Yang*[4], *Wayne Luk*[5], *Wei Xue*[1,2,3]
Oskar Mencer[6], *Xiaomeng Huang*[1], *and Guangwen Yang*[1,2,3]
1.Ministry of Education Key Laboratory for Earth System Modeling, Center for Earth System Science, Tsinghua University
2.Department of Computer Science and Technology, Tsinghua University
3.Tsinghua National Laboratory for Information Science and Technology (TNList)
4.Institute of Software, and State Key Laboratory of Computer Science, Chinese Academy of Sciences
5.Department of Computing, Imperial College London
6.Mexeler Technologies

***Abstract*—Atmospheric modeling is an essential issue in the study of climate change. However, due to the complicated algorithmic and communication models, scientists and researchers are facing tough challenges in finding efficient solutions to solve the atmospheric equations. In this paper, we accelerate a solver for the three-dimensional Euler atmospheric equations through reconfigurable data flow engines. We first propose a hybrid design that achieves efficient resource allocation and data reuse. Furthermore, through algorithmic offsetting, fast memory table, and customizable-precision arithmetic, we map a complex Euler kernel into a single FPGA chip, which can perform 956 floating point operations per cycle. In a 1U-chassis, our CPU-DFE unit with 8 FPGA chips is 18.5 times faster and 8.3 times more power efficient than a multicore system based on two 12-core Intel E5-2697 (Ivy Bridge) CPUs, and is 6.2 times faster and 5.2 times more power efficient than a hybrid unit equipped with two 12-core Intel E5-2697 (Ivy Bridge) CPUs and three Intel Xeon Phi 5120d (MIC) cards.**

***Keywords*-atmospheric simulation; 3D Euler equations; hybrid methodology; FPGA; customizable precision**

I. Introduction

Studying climate change has long become an urgent issue that not only benefits our interests, but also helps protect the planet for future generations. Especially in recent decades, climate problems such as extreme weather events have caused huge losses and brought significant influence on human activities. According to the major report [1] of the United Nations in March 2014, the impacts of global warming are likely to be "severe, pervasive and irreversible".

In a climate system model, the atmospheric component model is one of the most essential and challenging parts, and has nowadays become a popular topic that widely applies the world's most powerful supercomputers (*e.g.*, [2], [3], [4]). However, to solve the complex atmospheric equations, traditional platforms have to face the constraints from data representation [3], memory accessing patterns [5], and data communications. Therefore, it is an urgent demand for developing wise and efficient methods and architectures to solve atmospheric equations.

Meanwhile, reconfigurable data flow engines (DFEs) such as Field Programmable Gate Arrays (FPGAs) start to appear as high performance platforms, and have acquired inspiring results in many key applications such as exploration geophysics [5] and financial modeling [6]. Compared with traditional architectures, reconfigurable platforms can achieve high performance through a deep pipeline of concurrent operations. Its customization on data representation provides great flexibility to optimize algorithms with rigor requirement on data precisions. Furthermore, the low clock frequency can decrease the power consumption to a great degree, and accordingly provides a more green way for computing.

In this paper, we develop a highly-efficient and green DFE solver for the 3D Euler equations, which are the most essential equation sets that describe the mesoscale atmospheric dynamics.

Our major contributions are:

- a hybrid CPU-DFE design to solve the Euler equations in a more efficient mechanism (Section III);
- optimizations based on algorithmic offsetting, fast memory table and customizable precision to decrease the usage of computing resources (Section IV);
- implementation and evaluation of the proposed approaches, with significant improvements in performance and in power efficiency over multicore and manycore processors (Section V and VI).

II. Background

A. Related Work

In atmospheric study, preliminary work through FPGA has achieved promising results. Smith *et al.* [7] accelerate the Parallel Spectral Transform Shallow Water Model using ORNL's SRC Computers. They manage to deploy and accelerate the key subroutines (FFT or LT) on the FPGA clusters. Oriato *et al.* [8] accelerate a realistic dynamic core of LAM model using Maxeler [9] DFE platform. It is a successful trial to reduce the resource usage through fixed-point arithmetic, and acquire satisfying speedup over CPU counterpart. In [10], we manage to solve the global-scale shallow water equations through mixed-precision arithmetic on DFEs. A significant speedup and higher power efficiency are achieved over a hybrid CPU-GPU supercomputer node.

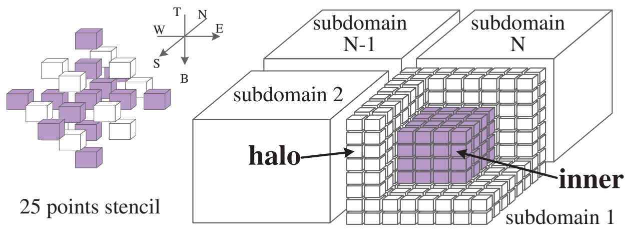

Figure 1. **Left**: 25 points stencil. **Right**: Hybrid domain decomposition: each subdomain is divided into inner (purple) and halo (blank) areas.

Compared with those work, the Euler equations we study are much more complex in the spatial discretization schemes and data layouts, and thus bring more tough challenges.

In terms of customizable precision, Duben *et al.* [11] investigate the usage of stochastic processing hardware and low precision arithmetic in atmospheric models. Their work has achieved convincing accuracy using low-precision data, and proved the feasibility of utilizing mixed-precision in climate science. Another work in [10] proposes two tracking methods to choose the bit width that can obtain the best balance between the resource usage and the accuracy.

B. Euler Atmospheric Equations and Algorithm

It is well recognized that the mesoscale atmosphere can be modeled by the fully compressible Euler equations with almost no assumptions made [12]. In a 3D channel with possibly nonsmooth bottom boundary, ignoring the effect of Coriolis force, the Euler equations can be written as the following set of conservation laws:

$$\frac{\partial Q}{\partial t}+\frac{\partial F}{\partial x}+\frac{\partial G}{\partial y}+\frac{\partial H}{\partial z}+S=0, \tag{1}$$

where
$$\begin{aligned} Q&=(\rho',\rho u,\rho v,\rho w,(\rho\theta)')^T,\\ F&=(\rho u,\rho uu+p',\rho uv,\rho uw,\rho u\theta)^T,\\ G&=(\rho v,\rho vu,\rho vv+p',\rho vw,\rho v\theta)^T,\\ H&=(\rho w,\rho wu,\rho wv,\rho ww+p',\rho w\theta)^T,\\ S&=(0,0,0,\rho' g,0)^T, \end{aligned} \tag{2}$$

where ρ, $\mathbf{v}=(u,v,w)$, p, and θ are the density, the velocity, the pressure and the potential temperature of the atmosphere, respectively. The system is closed with the equation of state

$$p=p_{00}\left(\frac{\rho R\theta}{p_{00}}\right)^{\gamma}, \tag{3}$$

where $p_{00}=1013.25\text{hPa}$ is the ground level pressure, $R=287.04\text{J}/(\text{kg}\cdot\text{K})$ is the gas constant for dry air and $\gamma=1.4$. To minimize roundoff errors, values of $\rho'=\rho-\overline{\rho}$, $(\rho\theta)'=\rho\theta-\overline{\rho}\overline{\theta}$ and $p=p-\overline{p}$ have been shifted according to the hydrostatic state that satisfies $\frac{\partial\overline{p}}{\partial z}=-\overline{\rho}g$.

After using a cell-centered finite volume scheme plus an explicit Runge-Kutta time stepping [2] method, each time step in solving the Euler equations requires two stencil sweeps applied at all mesh elements. As shown in the left panel of Fig. 1, to process a mesh element in the 3D channel, 24 neighboring elements need to be accessed. The right panel of Fig. 1 demonstrates the 3D domain decomposition for parallel computing. We first decompose the whole 3D channel into small subdomains according to the number of paralleling resources, and then all the subdomains will be processed in parallel. Based on the diamond stencil, communication is required between subdomains to update the halo elements, which are the two outer layers of the subdomain (*e.g.* blank elements of subdomain 1 in Fig. 1).

Algorithm 1 Original Euler Algorithm per Stencil Sweep

```
 1: for $(\mathbf{k},\mathbf{j},\mathbf{i}) \leftarrow (0,0,0)$ to $(N_{k-1},N_{j-1},N_{i-1})$ do
 2:     if $(\mathbf{k},\mathbf{j},\mathbf{i}) \in$ Boundary then          ▷ Boundary Condition
 3:         Halo Updating or Boundary Processing
 4:     end if
 5:     Calculate Coordinates                                ▷ Stencil Begins
 6:     Compute Fluxes{
 7:         State Reconstruction
 8:         Riemann Solver }
 9:     Compute Source Terms                                 ▷ Stencil Ends
10: end for
```

Algorithm 1 shows the original algorithm in each stencil sweep. For every mesh element inside a subdomain, if it belongs to the boundary, we apply boundary condition (line 3). After that, we do the stencil computing (line 5-9), including Calculate Coordinates, State Reconstruction (to recover values on the interfaces of each mesh element), Riemann Solver (to estimate the numerical fluxes) and Source Terms Computing (to count in the effect of the gravity force). We remark here that because of the conservative property of the finite volume scheme, the numerical fluxes on a common edge of two consecutive mesh elements are identical.

C. Necessity and Challenges

Efficiently solving the 3D Euler equations is a crucial step to finally build a complete mesoscale atmospheric model. We have to face more challenges than our preliminary work ([3], [10]) based on 2D Shallow Water Equations (SWEs).

The more realistic 3D model built upon the Euler equations brings more complex data communication pattern than the 2D scenario for SWEs. The boundary condition contains heavy halo updating and a large number of resource-demanding conditional statements. Moreover, due to the extremely complicated algorithm, the total number of floating-point operations for original Euler algorithm (at least 2100) is substantially increased, almost doubling the number for SWEs ([3], [10]). The surge in operations greatly challenges the limited FPGA resources, and desires extra optimizations to reduce the resource usage other than simply utilizing customizable data precision.

III. Hybrid CPU-DFE Design

A. Hybrid CPU-DFE Mechanism

In order to avoid the DFE solver processing the complex conditional statements and halo exchange from the Boundary Condition, we first decompose each subdomain into an inner area (purple elements in the right panel of Fig. 1), and two outer layers (halo in the right panel of Fig. 1). The

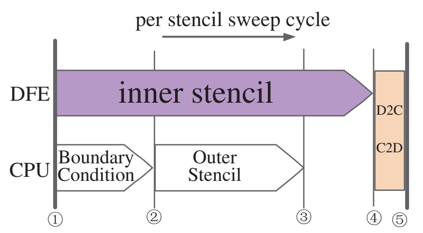

Figure 2. Work flow of the hybrid design. CPU and DFE are working simultaneously. D2C: DFE to CPU, C2D: CPU to DFE

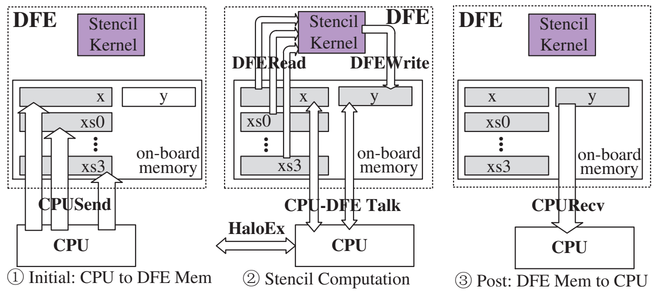

Figure 3. Hybrid data exchange model.

complex Boundary Condition only happens in the two outer layers (halo) of each subdomain. Therefore, we apply DFE to simply process the inner area stencil computations, and apply CPU to handle the outer area, including the Boundary Condition and stencil computations of the outer layers. The work flow of the hybrid design is shown in Fig. 2. In this way, DFEs are now only focusing on the more regular stencil of inner area (① to ④), without worrying about the outer area halo updating and conditional statements (① to ②)that would consume a lot of the FPGA resources. Besides, CPU is now working simultaneously with DFEs, which archives a more balanced resource allocation. After both CPU and DFE finish their work (④), only four layers of elements along the inner-outer boundary need to be exchanged between CPU and DFE (shown as C2D and D2C in Fig. 2).

B. *Hybrid Data Exchange Model*

For heterogeneous architectures, data exchange between the processors and accelerators is usually a key issue that deserves careful consideration. The basic principles are to maximize data reuse while data stays in the card, and to minimize data movement between different cards.

As for atmospheric modeling, the propagate data sets are required to be updated every time step. However, there is also a number of constant data sets that label the basic information of a chosen area. The constant data sets do not need to be exchanged at every time step, and can be stored in the on-board memory for the whole program cycle.

In the original Euler algorithm, even though the data sets required for the stencil computation include six arrays ($x, xs0, xs1, xs2, xs3$ and y), four ($xs0, xs1, xs2$, and $xs3$) of them are constant arrays, representing the physical environment of the selected surface. Accordingly, the constant arrays can be stored on the DFE on-board memory, rather than being exchanged every time step. The hybrid data exchange model is shown in Fig. 3 with three different steps:

1) we send (**CPUSend**) all five input arrays to the DFE on-board memory, which has a higher bandwidth than the PCI express interconnections.
2) at each time step of the stencil computation, DFE reads (**DFERead**) all the input arrays from on-board memory, streams them through the Stencil Kernel for computation, and writes (**DFEWrite**) the output stream to y in the on-board memory. After the computation, only the propagate arrays x and y will be exchanged (**CPU-DFE Talk**) between CPU and DFE. CPU will also be in charge of the halo updating (**HaloEx**) to exchange data with neighboring subdomains.
3) when step ② finishes after a given number of time steps, CPU will read the whole y (**CPURecv**) from the DFE on-board memory.

The overhead of the data copy in step ① and ③ is small and can be ignored in the long-term simulation which contains thousands of stencil computation (②). Besides the hybrid data exchange model, techniques such as hardware data-compression [5] and mathematical reduction [3] could also be applied to optimize the data exchange.

IV. Design of Euler Stencil Kernel

A. *Streaming Computing Model*

In this section, we introduce approaches to decrease the usage of computing resources, so as to fit the complex Euler kernel into one single FPGA. The hardware implementations are introduced in the next section.

Reconfigurable platforms achieve computation through developing a deep pipeline of concurrent units, and processing the targeting problem in a streaming model. In terms of the Euler algorithm, we use the computing resources of the FPGA chip to deploy the hardware kernel (Fig. 4) that contains different modules equivalent to the steps listed in Algorithm 1. Input streams from the inner buffer will go through corresponding modules and get processed. The total number of floating-point operations for the original algorithm is shown in the Original ALG row in Table I.

B. *Algorithmic Offsetting*

The identical rule to compute the fluxes on a common edge of two consecutive mesh elements (remarked in Section II-B) offers us a big optimizing space by means of the streaming offsetting model. We thus carry out an algorithmic offsetting method to simplify the Euler stencil kernel.

Algorithm 2-Part 1 shows a fragment of the State Reconstruction step to compute east-direction intermediate variables $qe[0]$, $qe[1]$, and west-direction intermediate variable $qw[1]$. We can figure out that computing $qe[0]$ and $qw[1]$ is

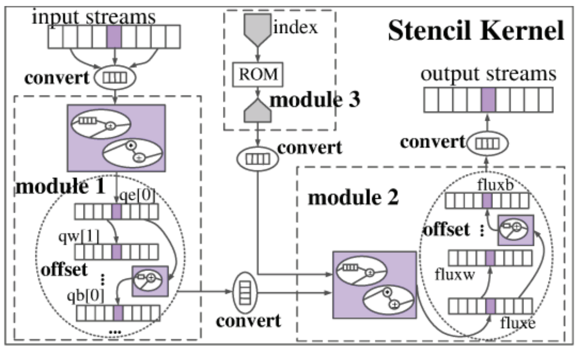

Figure 4. General Architecture of Stencil Kernel. Different modules refer to corresponding steps of processing Euler equations.

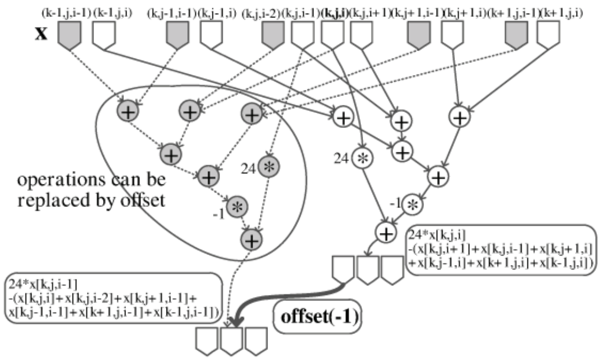
(k-1,j,i-1)
(k-1,j,i)
(k,j-1,i-1)
(k,j-1,i)
(k,j,i-2)
(k,j,i-1)
(k,j,i)
(k,j,i+1)
(k,j+1,i-1)
(k,j+1,i)
(k+1,j,i-1)
(k+1,j,i)
x
operations can be replaced by offset
24
-1
24
-1
24*x[k,j,i]
-(x[k,j,i+1]+x[k,j,i-1]+x[k,j+1,i]
+x[k,j-1,i]+x[k+1,j,i]+x[k-1,j,i])
24*x[k,j,i-1]
-(x[k,j,i]+x[k,j,i-2]+x[k,j+1,i-1]+
x[k,j-1,i-1]+x[k+1,j,i-1]+x[k-1,j,i-1])
offset(-1)

Figure 5. The streaming pipeline to finish the corresponding calculations. Operations by dotted lines can be replaced by offsetting existing streams.

Table I
Number of Floating-Point Operations per Sweep

part 1	$+,-$	$\times$	$\div$	Pow,Sqrt	OFFSET
Original ALG	1192	697	170	48	132^a
ALG Offsetting	619	549	76	21	$30^a + 140^b$
Look Up Table	424	460	51	21	$30^a + 140^b$

a: OFFSET operations on original input streams
b: New OFFSET operations generated after using ALG offsetting

applying an identical rule on different elements from input stream x, and computing $qe[1]$ and $qe[0]$ is applying similar rules on different elements. We further use Fig. 5 to illustrate the streaming pipelines that are deployed to compute the bolded parts of $qe[0]$ and $qw[1]$ in Algorithm 2-Part 1. The operations connected by solid pipe lines are deployed to compute the bolded part of $qe[0]$, and the operations connected by dotted pipe lines are deployed to compute the bolded part of $qw[1]$. Obviously, the operations in the two pipelines are identical, and the input elements for dotted pipeline are one time step backward of elements for solid pipeline. As a result, all the operations connected by dotted lines can be replaced by offsetting existing stream one time step backward (**offset(-1)**). Similarly, we only need to compute the stream $qe[0]$ in Algorithm 2. By offsetting stream $qe[0]$ one time step backward, we can get the stream of $qw[1]$ at current time step. As for $qe[1]$, a similar offsetting methods could also be applied with few modifications. The approach of algorithmic offsetting is shown in Algorithm 2-Part 2.

Through applying algorithmic offsetting on both the State Reconstruction and Riemann Solver steps (dotted circle in Fig. 4), we manage to eliminate 40% of the total floating operations (ALG Offsetting row in Table I).

C. Fast Memory Table and Mixed-precision Arithmetic

In the Calculate Coordinates step of Algorithm 1, all the variables only rely on the index coordinate (k, j, i). Therefore, those variables can be pre-calculated during compiling time by CPU and stored as a table in the on-chip fast memory. When needed, those variables can be acquired through looking up the table (module 3 in Fig. 4). If the on-chip fast memory is not big enough to store all the coordinate variables, we can alternatively store extra variables on the large on-board memory as constant variables. Consequentially, we occupy extra on-chip fast memories in exchange of computations, and further eliminate over 10% of the original operations (Look Up Table row in Table I).

The customizable feature on data representation provides another method to reduce the resource usage. We can apply reduced-precision floating-point number to replace the original double precision. The general-purpose MaxCompiler development environment [9] offers a user-friendly programming interface to specifically assign the data type and bit width, and to analyze the requirement of the computing resources. For the mantissa part, we gradually decrease the bit width and observe the influence on the resource usage and the accuracy. We choose the least mantissa width that could meet the resource requirement and guarantee the accuracy at the same time. The exponent part is decided through tracking the data range of the variables throughout the program, and finding the least exponent width that is still good enough to cover the range of all variables. The method above is introduced in detail in [10]. Therefore in Fig. 4, input streams will be converted into reduced-precision data before being computed, and will be converted back to double precision before being output.

V. Hardware Implementation

A. DFE Implementation

The techniques proposed in Section III and Section IV are applicable to any hybrid systems with DFEs as accelerators, and can be easily generalized to other stencil-based applications. In our experiment, we choose Maxeler dataflow computing platforms [9] to implement the designs. Our test set is based on a model problem, the baroclinic instability test in a 3D channel [13]. We set the mesh size of the channel to be $(x \times y \times z) = (260 \times 240 \times 228)$.

The Maxeler MPC-X is a hybrid computing unit with two 6-core E5650 CPUs (hyper-threading enabled) and 8 DFE accelerator cards. Each DFE card has one Altera Stratix5 D8 FPGA chip and up to 48 GB on-board memory. There is also

Algorithm 2 Demonstration of the Algorithmic Offsetting Method

```
1: Part 1: Original code. ($q_e[0], q_e[1]$, and $q_w[1]$ are intermediate variables in the step of State Reconstruction)
2: $qe[0]$ = 24*x[k,j,i] - (x[k,j,i+1]+x[k,j,i-1]+x[k,j+1,i]+x[k,j-1,i]+x[k+1,j,i]+x[k-1,j,i]) + 3*(x[k,j,i+1]+x[k,j,i-1]) - 2*(x[k,j,i+1]-x[k,j,i-1]);
3: $qe[1]$ = 24*x[k,j,i+1] - (x[k,j,i+2]+x[k,j,i]+x[k,j+1,i+1]+x[k,j-1,i+1]+x[k+1,j,i+1]+x[k-1,j,i+1]) + 3*(x[k,j,i+2]+x[k,j,i]) + 2*(x[k,j,i+2]-x[k,j,i]);
4: $qw[1]$ = 24*x[k,j,i-1] - (x[k,j,i]+x[k,j,i-2]+x[k,j+1,i-1]+x[k,j-1,i-1]+x[k+1,j,i-1]+x[k-1,j,i-1]) + 3*(x[k,j,i]+x[k,j,i-2]) - 2*(x[k,j,i]-x[k,j,i-2]);
1: Part 2: Algorithmic offsetting method.
2: $tmp1$ = 24*x[k,j,i]-(x[k,j,i+1]+x[k,j,i-1]+x[k,j+1,i]+x[k,j-1,i]+x[k+1,j,i]+x[k-1,j,i])+3*(x[k,j,i+1]+x[k,j,i-1]);
3: $tmp2$ = 2*(x[k,j,i+1]-x[k,j,i-1]);
4: $qe[0](\mathbf{t})$ =tmp1 - tmp2;
5: $qe[1](\mathbf{t})$ =tmp1(t+1) + tmp2(t+1);
6: $qw[1](\mathbf{t})$ =qe[0](t-1);
```

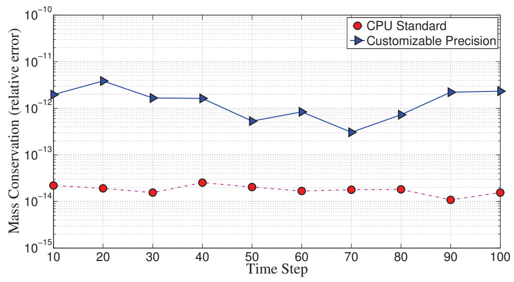

Figure 6. Mass conservation: relative error should be less than 10^{-11}.

6 MB on-chip fast memory that can provide a bandwidth of 14 TBytes/s. DFEs are connected with the CPU through PCI Express 2.0 interconnection.

In the hybrid design, we first decompose the total channel into eight $(130 \times 120 \times 114)$ subdomains. For each subdomain, the DFE processes the inner area, and the CPU processes the outer area. The halo exchange is done automatically by the neighboring communication functions from the framework of PETSC (Portable Extensible Toolkit for Scientific Computation) [14]. Constant streams will stay in the large on-board memory, and coordinate data will stay in the on-chip fast memory. We apply the algorithmic offsetting method to eliminate the unnecessary operations, and apply customizable precision to further reduce the resource usage. Based on the analysis in Section IV-C and the resources provided by Stratix5 D8 chip, we finally set the floating-point number to be 11 bits exponent and 32 bits mantissa.

Fig. 6 shows the accuracy validation on the basis of the mass conservation law. Mass conservation is an essential integral invariants in atmospheric simulation. Mathematically, the discretization scheme we employ leads to exact mass conservation. Due to the truncation error, the error of mass conservation is near to machine epsilon (around 10^{-14} in double precision). This conservation law can be further relaxed to 10^{-11}, indicating that at most 1% of total mass discrepancy is introduced after a billion time steps.

B. Reference Implementation

Our reference implementations are based on a hybrid rack with two 12-core Intel E5-2697 (Ivy Bridge) CPUs, and three Intel Xeon Phi 60-core 5120d (MIC) cards.

In the CPU implementation, we employ OpenMP multi-threading and vectorization to scale the performance over cores and vector units, and employ cache blocking to improve the efficiency of data reuse. After careful tuning, we manage to scale the CPU performance over 24 threads and achieve a speedup of 20x over a serial implementation.

In the CPU-MIC implementation, as the hybrid unit contains three MIC cards, we first decompose the whole data channel into three equal subdomains according to the right panel of Fig. 1. After applying the same hybrid methodology we proposed in Section III, MIC only needs to process the inner area of each subdomain, with CPU cores processing the outer area simultaneously. Both the CPU and MIC programs are fully optimized through OpenMP multi-threading, vectorization and cache blocking. The best performance is achieved with 24 threads for CPU and 236 threads for MIC in the offload mode.

VI. Results and Analysis

Table II shows the performance (number of mesh points processed per second) and power efficiency (performance per Watt) for different implementations. Note that the power consumption is measured with a power meter, and the FPGA chip works on a frequency of 180MHz. The CPU-MIC rack has the same physical size as the MPC-X FPGA unit, and can provide a more fair comparison based on performance per volume and power efficiency per volume.

The CPU-DFE unit with 8 FPGA chips is 18.5 times faster and 8.3 times more power efficient than the multi-core CPU implementation with two 12-core Intel E5-2697 (Ivy Bridge) CPUs, and is 6.2 times faster and 5.2 times more power efficient than a hybrid implementation with two 12-core Intel E5-2697 CPUs and three Intel Xeon Phi 5120d (MIC) cards.

The reference designs based on the most powerful multi-core and many-core architectures have been fully optimized through a series of sophisticated paralleling techniques. However, the overall performance is not scaled ideally, as we have to face the challenges from the complex stencil algorithm, and the bandwidth bottleneck caused by heavy data exchange. The discontinuous data access for the stencil computation leads to a higher rate of cache miss, and further restricts the computing efficiency. As for the DFE design, through applying hybrid mechanism, algorithmic

Table II
PERFORMANCE AND POWER EFFICIENCY FOR DIFFERENT PLATFORMS

Mesh size: $260 \times 240 \times 228$					
	performance (points/s)	speedup	power (Watt)	efficiency Perf/Watt	power efficiency
CPU 24-core	154K	1	427	0.36K	1
CPU-MIC unit	474K	3x	815	0.58K	1.6x
CPU-DFE unit	2.85M	**18.5x**	950	3K	**8.3x**

offsetting, fast memory table, and the customizable-precision arithmetic, we manage to map the complex Euler stencil kernel into a single FPGA chip (Fig. 4), and build a deep pipeline that can efficiently perform nearly 1000 floating-point operations per cycle. The inputting streams in addition form a cache-like data buffer that provides perfect data access [15]. All above contributions finally lead to the better performance of DFE over the reference designs.

In terms of the power efficiency, due to the low clock frequency, DFE generally consumes a lower energy usage over traditional platforms, and accordingly becomes a better alternative for studying the climate issues in a more environmentally-friendly and green way.

Note that the CPU-MIC unit used in the reference implementation can be considered as a replication of one computing node from Tianhe-2, the world's top supercomputer with a theoretical peak performance of 54.9 PFlops. As atmospheric modeling generally desires large-scale experiment to achieve better resolutions, we can project that our hybrid CPU-DFE design, if scaled to a large-scale supercomputer scenario, would demonstrate a similar speedup as archived in this paper, and greatly reduce the huge power consumptions.

VII. CONCLUSION

This paper proposes a highly-efficient and green DFE solver for the complex Euler atmospheric equations. The hybrid design achieves more balanced resource allocation and better data reuse, and can be generalized to any heterogeneous node equipped with both processors and accelerators. Optimizing approaches based on algorithmic offsetting, fast memory table and customizable-precision help us map the resource-demanding stencil kernel into a single FPGA chip and gain significant improvements in performance and power efficiency over multicore and manycore platforms. The experimental results show great potential on utilizing reconfigurable platforms in the atmospheric simulations.

ACKNOWLEDGMENT

This work was supported in part by National Natural Science Foundation of China (Grant No. 61303003, 41374113, 61170075 and 91130023), by National High-tech R&D (863) Program of China (Grant No. 2013AA01A208), by 973 Program of China (Grant No. 2011CB309701), by UK EPSRC, by the European Union Seventh Framework Programme (Grant No. 257906, 287804 and 318521), by the HiPEAC NoE, Maxeler University Program, and Altera.

REFERENCES

[1] M. D. M. Christopher B. Field, Vicente R. Barros *et al.*, *Climate Change 2014: Impacts, Adaptation, and Vulnerability, Summary For Policymakers*. United Nation, IPCC, 2014.

[2] C. Yang and X.-C. Cai, "A scalable fully implicit compressible Euler solver for mesoscale nonhydrostatic simulation of atmospheric flows," *SIAM J. Sci. Comput. To appear*, 2014.

[3] 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 the 18th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*. ACM, 2013, pp. 1–12.

[4] T. Shimokawabe, T. Aoki, and J. Ishida, "145 TFlops performance on 3990 GPUs of TSUBAME 2.0 supercomputer for an operational weather prediction," *Procedia Computer Science*, vol. 4, pp. 1535–1544, 2011.

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

[6] 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.

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

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

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

[10] L. Gan, H. Fu, W. Luk, C. Yang, W. Xue, X. Huang, Y. Zhang, and G. Yang, "Accelerating solvers for global atmospheric equations through mixed-precision data flow engine," in *FPL 2013*. IEEE, 2013, pp. 1–6.

[11] P. D. Düben, H. McNamara, and T. Palmer, "The use of imprecise processing to improve accuracy in weather & climate prediction," *Journal of Computational Physics*, 2013.

[12] P. H. Lauritzen, C. Jablonowski, M. A. Taylor, and R. D. Nair, Eds., *Numerical Techniques for Global Atmospheric Models*. Springer, 2011.

[13] P. Ullrich and C. Jablonowski, "Operator-split runge-kutta-rosenbrock methods for nonhydrostatic atmospheric models." *Monthly Weather Review*, vol. 140, no. 4, 2012.

[14] S. Balay, J. Brown, K. Buschelman, V. Eijkhout, W. Gropp, D. Kaushik, M. Knepley, L. C. McInnes, B. Smith, and H. Zhang, "PETSc users manual revision 3.4," 2013.

[15] H. Fu and R. G. Clapp, "Eliminating the memory bottleneck: an FPGA-based solution for 3D reverse time migration," in *FPGA 2011*. ACM, 2011, pp. 65–74.