Solving Mesoscale Atmospheric Dynamics Using a Reconfigurable Dataflow Architecture

This article presents an efficient dataflow methodology for solving Euler atmospheric dynamic equations. The authors map a complex Euler stencil kernel into a single field-programmable gate array chip and develop a long streaming pipeline that can perform 956 mixed-precision operations per cycle. Their dataflow design outperforms traditional multicore and many-core counterparts in both time to solution and energy to solution.

Lin Gan, Haohuan Fu
Tsinghua University

Wayne Luk
Imperial College London

Chao Yang
Chinese Academy of Sciences

Wei Xue, Guangwen Yang
Tsinghua University

Numerical atmospheric simulation is an essential method to study the climate system and a key tool for verifying climate changing mechanisms, making predictions into the future, and providing guidance for protecting the planet from severe issues such as climate change and weather disasters. Due to atmospheric models' high demand for computing power, a lot of effort has been made to study and simulate the atmospheric model using the world's most powerful supercomputers. However, traditional high-performance computing (HPC) systems based on multicore and many-core architectures have to face architectural constraints from data representation, memory access patterns, communications, and bandwidth, as well as challenges from the extremely complex atmospheric algorithm. Thus, novel architectures and revolutionary optimization techniques are in great demand toward achieving better results in both time to solution and energy to solution.

In recent years, dataflow engines (DFEs) based on reconfigurable field-programmable gate arrays (FPGAs) have developed quickly and achieved many inspiring results in various applications such as deep learning.[1] Unlike traditional architectures, reconfigurable DFEs[2] obtain high performance through deploying a long pipeline of concurrent operations corresponding to the targeting algorithm, while maintaining high power efficiency due to the lower clock frequency. Their customizable features on the hardware circuit, data representation, and on-chip memory provide great flexibility and optimizing space to tackle complex numerical algorithms.

Published by the IEEE Computer Society
0272-1732/17/$33.00

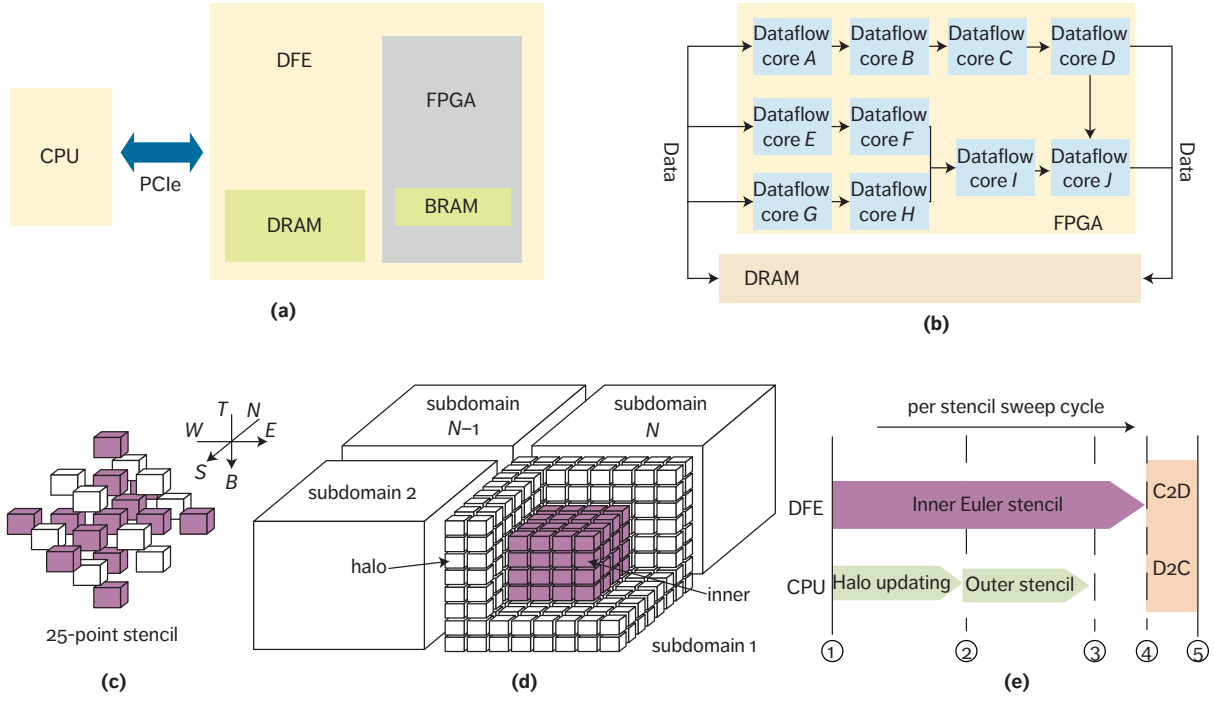

Figure 1. The dataflow engine (DFE) architectures and algorithmic computing mechanisms. (a) General architecture of the hybrid CPU-DFE system using PCI Express as a bridge. (b) Dataflow computing model. Each dataflow core refers to a specific operation. (c) The 25-point Euler stencil. (d) The hybrid domain decomposition methodology assigns inner stencil to DFE and outer area to CPU. (e) Workflow of the hybrid methodology, in which C2D and D2C refer to the data exchange between CPU and DFE.

In this article, we present an efficient dataflow computing model to solve the 3D Euler atmospheric equations, which is the essential step to describe mesoscale atmospheric dynamics. Through fully exploiting the customizable features of the selected FPGA platforms, we propose a set of novel optimizing techniques such as a customizable window buffer, algorithmic offsetting arithmetics, and a mixed-precision mechanism; manage to map the complex Euler kernel into a single FPGA chip; and build a long computing pipeline that can perform 956 mixed-precision operations per cycle. We also present our efforts to fully optimize the Euler algorithm over traditional platforms including CPU, Many Integrated Coprocessor (MIC), GPU, and SW26010 CPU.

Our dataflow design outperforms traditional multicore and many-core counterparts in terms of both time to solution and energy to solution. The performance comparisons with other major multicore and many-core processors prove dataflow computing to be a promising method in the post-Moore era to break the architectural constraints and bring inspiring achievements.

Background

This section introduces the hardware architecture, equations, and algorithm, and summarizes the existing efforts.

Dataflow Engines

Figure 1a shows the general architecture of a hybrid CPU-DFE system. Similar to the traditional hybrid system, it also contains CPUs to handle issues such as DFE initialization, task scheduling, and data exchange.

Compared with traditional multicore and many-core architectures such as CPU, MIC, and GPU, DFE achieves high performance through developing within a single FPGA chip a long streaming pipeline that comprises

```
1: for (k, j, i) ← (0, 0, 0) to
   (N_{k-1}, N_{j-1}, N_{i-1}) do
2:     if (k, j, i) ϵ Halo then
3:         Halo Updating
4:     end if
5:     3D Euler Stencil
6: end for
```

Figure 2. Euler algorithm per stencil sweep.

many concurrent operations. Different hardware resources, such as logic units and digital signal processors (DSPs), are customized to deploy a specific hardware circuit that corresponds exactly to the targeted algorithm (see Figure 1b). Data that is originally stored in the onboard memory (DRAM) will stream through different dataflow cores on the pipeline and get computed. By streaming different dataflows through different dataflow cores, we can execute multiple instructions onto multiple data (MIMD), and thus boost the overall performance.

Euler Equations

We can write the Euler equations[3] as the following set of conservation laws:

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

in which

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

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

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

in which $p_{00} = 1013{:}25\text{hPa}$ is the ground-level pressure, $R = 287{:}04\text{J}/(\text{kg} \cdot \text{K})$ is the gas constant for dry air, and $\gamma = 1{:}4$. We remark here that because of the conservative property of the finite volume scheme,[3] the numerical fluxes on a common edge of two consecutive mesh elements are identical.

We used a cell-centered finite volume scheme plus an explicit Runge-Kutta time-stepping method,[3] after which each time step in solving the Euler equations (that is, the Euler algorithm) required two Euler stencil sweeps applied at all mesh elements. Figure 1c is the Euler stencil in the 3D channel, where to update a mesh element (for example, the central one in Figure 1c), we need to access 25 elements. After the original channel is decomposed into different subdomains corresponding to different MPI processes (see Figure 1d), the halo area needs to exchange data information with neighboring subdomain halos, which is marked as halo updating.

Figure 2 shows the Euler algorithm that contains two major steps: halo updating and 3D Euler stencil computing. Unlike an ordinary stencil kernel, the 3D Euler stencil has to perform over 2,100 double-precision operations to update one element per sweep,[4] and it is more complex in the communication model, resulting in tough computational challenges for traditional HPC platforms.

Computational Challenges

Traditional HPC systems based on many-core and multicore architectures face tough challenges in part from the complex algorithmic kernel. The more realistic 3D model built on the Euler equations brings more complex algorithms and data communication patterns, and thus greatly strikes the memory bandwidth. The neighboring points of a 25-point stencil are stored in different rows, columns, or slices, and can be far apart in the memory space. Accessing all 25 points incurs a lot of cache misses.

Although complex atmospheric stencils have been widely studied on traditional HPC platforms, the achieved efficiency is relatively low (generally less than 10 percent). Therefore, novel architectures and revolutionary optimization techniques are in great demand to achieve better efficiency.

Existing Efforts

With increased computing power in the form of many-core processors and accelerators (for

example, the Sunway many-core processor, GPU, MIC, and so on), we see a lot of efforts at porting and refactoring atmospheric models on the most powerful supercomputers to achieve ultra-high scalability and resolution. A recent work[5] that won the 2016 ACM Gordon Bell Prize scales a fully implicit Euler solver to over 10.5 million cores of the Sunway TaihuLight supercomputer[6] and sustains a double-precision performance of 7.95 Pflops with 488-m horizontal resolution. Although these many-core architectures demonstrate potential for improving the performance of existing models, we still see a gap of several orders of magnitude to advance to the next level of science.

On the other hand, preliminary work through FPGAs has achieved promising results in recent years. Melissa Smith and her colleagues accelerated the Parallel Spectral Transform Shallow Water Model using ORNL's SRC Computers.[7] They managed to deploy and accelerate the key subroutines (such as fast Fourier transform or Laplace transform) on the FPGA clusters. Diego Oriato and his colleagues accelerated a realistic dynamic core of a meteorological limited area model (LAM)[8] using the Maxeler[2] DFE platform. It was a successful trial to reduce the resource usage through fixed-point arithmetic with satisfactory speedup over multiple CPU cores. Compared with these works, the Euler equations we study are more complex in the spatial discretization schemes and data layouts and more meaningful for achieving cloud-resolving simulation in future models.

There are also many successful works on porting stencil kernels onto FPGAs. Kentaro Sano and his colleagues employed nine FPGAs (Altera Stratix IV EP4SGX230 and Stratix V EP5SGSD8 FPGAs) to compute 2D and 3D Jacobi computations in a deep pipeline approach and obtain computing efficiencies of 87.4 and 83.9 percent, respectively.[9] Xinyu Niu and his colleagues propose an approach to exploit runtime reconfigurability of FPGA for optimizing the revise time migration (RTM) stencil.[10] The achieved performance on a Xilinx Virtex-6 SX475T FPGA is up to two orders of magnitude faster than the CPU reference designs. Hasitha Waidyasooriya and his colleagues proposed an OpenCL-based FPGA design and achieved better stencil performance over CPUs and GPUs.[11] Koji Okina and his colleagues selected two essential 3D stencil kernels from the heat conduction and electromagnetic fields to evaluate their performance on stream-oriented FPGAs.[12] Comparatively, the 3D Euler stencil in this work is much more challenging to solve, because more than 2,100 double-precision floating-point operations must be performed.

Dataflow-Oriented Euler Designs

This section focuses on dataflow designs, including the essential optimizations to boost the overall performance.

Design Overview

Efficiently mapping a given algorithm into a DFE generally requires consideration of several issues.

Hybrid domain decomposition. The first step is to focus on the heterogeneous architecture for the best utilization on both the CPU and the DFE. So, the original algorithm can be divided into different tasks being processed by either the CPU or DFE, separately but simultaneously.

Hardware resource analysis. Because on-chip resources such as flip-flops and the DSP are required to build the hardware circuit for the streaming pipeline, those limited resources might not be enough to fulfill the demand for complex algorithms that generally contain too many operations, such as the Euler algorithm that originally contained more than 2,100 double-precision floating-point operations. In the rest of this section, we discuss the key designs we proposed to decrease the Euler algorithm's resource demands.

Hybrid Domain Decomposition

Figure 1d shows the hybrid decomposition methodology. We decompose each subdomain into an inner cube (the shaded elements in Figure 1d) and the outer area (the blank elements). So, all halo elements go to the outer area, and the inner cube contains only stencil computation. We assign the DFE to process the inner cube stencil computations, and we assign the CPU to process the outer area stencil computation and all communication. Figure 1e shows the workflow of hybrid domain decomposition.

The CPU and DFE are now working simultaneously, and the CPU time for computation and communication can be hidden inside the DFE time for stencil computation, which results in efficient computation-communication overlapping. Moreover, DFE no longer needs to spend resources implementing the outer-part computation.

Memory Optimization

In this section, we focus on the optimizations based on the hierarchic memory.

Customizable window buffer. As we explained earlier, accessing the 25 points from the 3D Euler stencil always incurs a lot of cache misses for traditional architectures. On an FPGA, BRAM can be used to construct a customizable window buffer to accommodate the streaming data from the 3D Euler stencil. The window buffer has the size covering all 25 points required by current stencil computing, and thus performs perfect cache mechanism with the following behaviors:

- Data will stream into the window buffer when required for stencil computing.
- Data will stream out of the window buffer when no longer required for stencil computing.
- During stencil computing, all 25 points required are in the window buffer and can be accessed immediately.

Fast memory table. In the stencil loop, variables that rely only on the index coordinate can be precalculated during compile time by the CPU and stored as a fast memory table using the BRAM. Those variables can be accessed through looking up the table. If the BRAMs are not big enough to store all the coordinate variables, we can alternatively store them on the DRAM as constant variables. So, we use more memory in exchange of operations.

Data accommodation strategy. Different types of data from the Euler algorithm are stored on different memory on the DFE side. The on-board large DRAM is used to accommodate the wave propagation variables[3] that must be updated every step, whereas the on-chip BRAM is used to store the coordinate variables to form the fast memory table. Because the kernel computation is not bounded by the memory bandwidth, the data can be accessed in time according to the FPGA frequency (180 MHz in our case) and in a 2D pattern by using MaxCompiler,[2] a source-to-source compiler that provides a high-level hardware description language.

Algorithm Optimization and DFE Programming

The identical rule to compute the fluxes on a common edge of two consecutive mesh elements provides a big optimizing space to simplify the Euler stencil based on the dataflow computing model. For example, Figure 3a shows the C code to compute variables $y0$ and $y1$. We can learn that computing $y0$ and $y1$ is putting an identical rule on different elements from x, and the x elements for computing $y1$ are one time step ahead of the x elements for computing $y0$. In other words, the result of $y1$ at step t equals the result of $y0$ at step $t - 1$. So, in a dataflow computing model, all dataflow operations for computing stream dfe_$y1$ can be replaced by offsetting stream dfe_$y0$ one time step backward (see Figure 3c). Algorithmic offsetting lets us reduce a large amount of the redundant floating-point operations. Figure 3b shows how to transfer between the CPU and the DFE and how to pipeline the kernel for size cycles.

Figures 3b and 3c also illustrate the programming details based on MaxCompiler.[2] The programming complexity is reduced to the same level of software programming. Meanwhile, the flexibility to play with hardware design options is mostly retained.

In Figure 3, input stream x goes through different operations to construct a computing pipeline for output streams $y0$ and $y1$. Similarly, when x goes to other pipelines for different output streams simultaneously, a MIMD model that corresponds to what is shown in in Figure 1b is formed accordingly.

Precision Optimization

The customizable feature on data representations is another method to reduce the resource usage. By using fixed-point or reduced-precision floating-point data to replace the original double-precision data, we can greatly decrease the Euler algorithm's resource requirements. To determine the best data precision, we applied

```
// x, y0, and y1 are the variables in the original Euler code
1: double y0 = 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]);
2: double y1 = 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] + xk + 1, j, i – 1] + x[k – 1, j, i – 1]);
(a)

// nx and ny are the domain size on x and y dimensions, respectively
1: const int size = nx * ny;
2: int sizeBytes = size * sizeof(double);
3: Stencil_Compute_writeLMem (0, sizeBytes, x);     \\ CPU transfers array x to DFE DRAM as
                                                       stream data
4: Stencil_Compute (size, y0, y1);                  \\ The DFE Stencil kernel is executed
                                                       and pipelined for size cycle
(b)

// dfe_x, dfe_y0, and dfe_y1 refer to the dataflows that correspond to x, y0, and y1
// nx and ny are the domain size on x and y dimensions, respectively
// in Java class Stencil_Compute
1: DFEVar dfe_x = io. input ("x", hwFloat (11, 53));
2: DFEVar dfe_y0 = 24* dfe_x – (stream . offset (dfe_x , 1) + stream . offset (dfe_x , – 1)
   + stream . offset (dfe_x, nx) + stream . offset (dfe_x, –nx) + stream . offset (dfe_x,
   nx * ny) + stream . offset (dfe_x, –nx * ny));
3: DFEVar dfe_y1 = stream . offset (dfe_y0, –1); \\ by using algorithmic offset, original
   operations can be eliminated
4: io . output ("y0", dfe_y0, hwFloat (11, 53));
5: io . output ("y1", dfe_y1, hwFloat (11, 53));
(c)
```

Figure 3. An example of algorithmic offsetting and programming based on MaxCompiler.[2] (a) The original Euler code written in C programming language. Its corresponding DFE implementation on MaxCompiler is shown in (b) and (c). (b) The host CPU code that transfers *x* from the CPU to DFE (line 3) and that pipelines the kernel for size cycle (line 4). The signals in the dataflow are called *DFEVar variables*, each of which is associated with a datatype (such as `hwFloat(11,53)`). (c) The `stream.offset` utility automates the construction of the window buffer (line 2) and facilitates the implementation of the algorithmic offsetting method (line 3), so the algorithmic offsetting method is easier to implement on the DFE than on traditional counterparts. Lines 1, 4, and 5 in (c) show functions to transfer data to and from DFE. MaxCompiler provides tools for hardware simulation and resource analysis.

range analysis and precision analysis. The basic idea for range analysis is to track and record the range of all variables throughout the iteration and estimate the best data type and width range to represent the variables. Fixed-point data can be used for modules whose variables cover a small range, whereas reduced-precision floating-point data can be used for the remaining modules with a wide variable range. The width, such as the fractional part for fixed-precision and the exponent part for floating-point, can also be estimated based on the range of variables.

The basic idea for precision analysis is to dynamically trace the influence on the final accuracy with a set of different bit widths and find the least bit width that can still guarantee a satisfying accuracy. For example, 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 accordingly. The relative error of divergence can be used as an important indicator for a quick estimation of accuracy, and shall stay under 11 percent to ensure accuracy.

Selected Hardware and Implementation

The hybrid CPU-DFE node contains two six-core E5650 CPUs (hyperthreading enabled)

and eight DFE accelerator cards. Each DFE card has one Altera Stratix5 D8 FPGA chip and up to 48 Gbytes of on-board memory. There is also 6 Mbytes of on-chip BRAM that can provide a bandwidth of 14 TBytes per second. DFEs are connected with the CPU through PCI Express 2.0.

The channel is first divided into eight subdomains, and the hybrid domain decomposition is applied on each subdomain. So, each DFE processes only the inner-part stencil computing, while simultaneously, the CPU is processing the outer-part stencil computing and the halo updating. No direct communication is thereby required between DFE cards. The halo exchange is done automatically by using the neighboring communication functions from the framework of PETSC (Portable Extensible Toolkit for Scientific Computation).[3]

For the inner-part stencil computation, we applied memory and algorithm optimizations to reduce the total number of computations to 956 (see Table 1). We then applied precision optimization to reduce the resource requirement by roughly 30 percent, and to an appropriate amount (see Table 2) that can fit the whole Euler algorithm onto the selected FPGA. The relative error of divergence increases from 10^{-14} to 10^{-12}, but still stays under 10^{-11} to ensure the accuracy of numerical simulation, which is validated in previous work.[4]

Table 1. Number of floating-point operations per sweep.

Operations	+, –	×	÷	Pow, Sqrt	OFFSET
Original ALG	1,192	697	170	48	132*
ALG offsetting	619	549	76	21	30* + 140†
Look-up table	424	460	51	21	30* + 140†

*OFFSET operations on original input streams.
†New OFFSET operations generated after using ALG offsetting.

Table 2. Resource usage of mixed-precision algorithm.

Resource	Flip flops	Multipliers (18 × 18)	DSP blocks	Block memory
Stratix 5 D8	524,800	3,926	1,963	2,567
Mixed precision	382,599	652	326	1,041

Multicore and Many-Core Designs

In this section, we explain our design details based on the reference counterparts targeting the same Euler code. Table 3 shows the system specifications. A lot of effort, including the algorithmic optimizations proposed earlier, is made on the reference designs for fair comparisons.

Hybrid CPU-MIC Design

The hybrid CPU-MIC node contains two 12-core Intel E5-2697 (Ivy Bridge) CPUs and three Intel Xeon Phi 60-core 5120d (MIC) cards (see Table 3). The same hybrid methodology we proposed earlier is also applied here to fully utilize both CPU and MIC resources, so that the MIC needs only 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 multithreading and vectorization to improve parallelism from different levels, and through cache blocking to improve the data reuse. Other efforts also include loop splitting, prefetching, and array-to-structure to structure-to-array.

The fully optimized performance based on two 12-core Intel E5-2697 is the CPU version for later performance comparison.

Hybrid CPU-GPU Design

The hybrid CPU-GPU node contains two 12-core Intel E5-2697 (Ivy Bridge) CPUs and two Tesla K40 GPUs (see Table 3).

Besides the hybrid domain decomposition methodology and the general tuning techniques such as multithreading, shared memory, coalesced access, kernel splitting, we further propose a set of novel GPU tuning techniques specifically designed for the Euler algorithm, including a customizable data

Table 3. Multicore and many-core system specifications.

Architecture	Clock (GHz)	Peak (TFlops) (float | double)	Memory (Gbytes)	Bandwidth (Gbytes/s) (theoretical | measured)	Year*
CPU (E5-2697)	2.70	0.52 | 0.26	64	100 | 60	2013
MIC (5120d)	1.09	2.03 | 1.01	8	352 | 159	2013
GPU (K40)	0.75	4.29 | 1.43	12	288 | 100	2013
CPU (SW26010)	1.45	3.06 | 3.06	32	130 | 103	2016

*Selected processors were announced close to the year when the selected FPGA was announced, except for the SW26010.

Table 4. Performance and power efficiency. (Mesh size is 260 × 240 × 228.)

Platform*	Performance (Gflops)	Speedup (×)	Power (W)	Efficiency (performance/W)	Power efficiency (×)
CPU node	85	1	427	0.20	1
CPU-MIC node	255	3	815	0.31	1.6
CPU-GPU node	220	2.6	625	0.35	1.8
SW26010 CPU	520	6	380	1.36	6.8
CPU-DFE node	1,570	18	950	1.66	8.3

*Details of different platforms can be found in the "Multicore and Many-Core Designs" and "Selected Hardware and Implementation" sections.

caching mechanism and thread warp rescheduling scheme. Remarkable performance boost is achieved among mainstream GPUs, including Tesla Fermi C2050, K20x, K40, and K80. A speedup of 31.64 is obtained on the Tesla K80 over a 12-core E5-2697 CPU.

SW26010 Many-Core CPU

The SW26010 CPU[6] is the many-core processor used to construct the Sunway TaihuLight, the world's most powerful supercomputer. Each SW26010 contains four core groups; each core group contains 65 cores, including 1 management processing element (MPE) plus 64 computing processing elements (CPEs). CPE cores within a core group can communicate with each other through low-latency register communication (P2P and collective communications). MPE has 32 Kbytes of L1 cache and 256 Kbytes of L2 cache, whereas CPE has 64 Kbyte scratchpad memory.

As for the optimization for the Euler algorithm, on the parallelism level, the algorithm is scaled among the 64 CPE cores first and then among the vector units inside each CPE core. On the memory level, a customized data-sharing scheme through register communication and on-the-fly array transposition are also proposed, while the basic mechanism is to improve the bandwidth utilization and decrease

the bandwidth requirement. A speedup of more than 100 times is achieved over the MPE-only version. Optimizing details can be found in our previous work.[5]

Performance Evaluation

This section shows the performance results and comparison, with corresponding analysis.

Performance and Power Efficiency

Table 4 summarizes the performance and power efficiency. The power consumption is measured with a power meter, and the FPGA works at a frequency of 180 MHz. The dataflow design based on a hybrid CPU-DFE node outperforms traditional multicore and many-core counterparts in both time to solution and energy to solution. The achieved performance on a hybrid CPU-FPGA node is 18 times faster and 8 times more power efficient than a CPU node, 6 times faster and 5 times more power efficient than a CPU-MIC node, 7 times faster and 4.7 times more power efficient than a CPU-GPU node, and 3 times faster and 1.2 times more power efficient than a Sunway SW26010 CPU.

Discussion

The customizable feature on on-chip hardware resources, memory, and data representations offers us significant optimizing space to overcome major constraints confronted by multicore and many-core architectures.

For example, compared with the imperfections in data presentation of traditional HPC platforms, the mixed-precision arithmetic that enables flexibility in data format and precision can greatly improve the computing efficiency of logic units and decrease consumption of on-chip resources. Algorithmic offsetting based on the streaming model can eliminate a large amount of redundant operations. Therefore, we greatly decrease the resource demands of the Euler algorithm and manage to build a long streaming pipeline that can perform 956 mixed-precision operations per cycle. The customizable window buffer performs better caching behaviors, whereas the on-chip fast memory decreases both the number of operations and the data transfers. Therefore, we can push data closer to the computing side and break the memory wall restriction.

Moreover, the FPGA chip's low clock frequency leads to better power efficiency compared with other platforms. The above reasons contribute to the inspiring results achieved in this work. Note that selected multicore and many-core processors are announced close to the year when the selected FPGA is announced, so we can provide comparisons of different systems from the same era.

As for the programming model, new methods such as OpenCL and MaxCompiler can support high-level programming and a user-friendly development environment, so the programming complexity for reconfigurable hardware is reduced to a level similar to that of software programming.

Compared with traditional multicore and many-core counterparts, the FPGA-based dataflow architecture provides a novel computing methodology through mapping parallelism into the increased number of transistors, and can greatly improve its performance and power efficiency. With the reconfigurable hardware becoming a component in future computing systems, we see it as a promising candidate to provide highly efficient and green HPC solutions in the post-Moore era. Future work will include using more powerful FPGAs and testing the scalability among multiple nodes.

Acknowledgments

This work was supported in part by the National Key R&D Program of China (grant nos. 2016YFA0602200 and 2016YFA0602103), the National Natural Science Foundation of China (grant nos. 41374113 and 91530323), the National High-Tech R&D (863) Program of China (grant no. 2015AA01A302), China Postdoctoral Science Foundation (grant no. 2016M601031), Tsinghua University Initiative Scientific Research Program (grant no. 20131089356), Major Science and Technology Foundation Program of Ministry of Education (special support for NSCC-Guangzhou, Sun Yat-sen University), China Special Fund for Meteorological Research in the Public Interest (grant no. GYHY201306062), Key Laboratory of Data Analysis and Application of State Oceanic Administration of China, the European Union Horizon 2020 Research and Innovation Programme (grant no. 671653), the UK EPSRC (EP/I012036/1, EP/L00058X/1, EP/L016796/1,

and EP/N031768/1), the Maxeler University Programme, and the Intel Programmable Solutions Group.

References

1. H. Sharma et al., "From High-Level Deep Neural Models to FPGAs," *Proc. 49th Ann. IEEE/ACM Int'l Symp. Microarchitecture*, 2016, pp. 1–12.
2. O. Pell and V. Averbukh, "Maximum Performance Computing with Dataflow Engines," *Computing in Science & Eng.*, vol. 14, no. 4, 2012, pp. 98–103.
3. C. Yang and X.-C. Cai, "A Scalable Fully Implicit Compressible Euler Solver for Mesoscale Nonhydrostatic Simulation of Atmospheric Flows," *SIAM J. Scientific Computing*, vol. 36, no. 5, 2014, pp. S23–S47.
4. L. Gan et al., "A Highly-Efficient and Green Data Flow Engine for Solving Euler Atmospheric Equations," *Proc. 24th Int'l Conf. Field Programmable Logic and Applications*, 2014, pp. 1–6.
5. C. Yang et al., "10m-Core Scalable Fully-Implicit Solver for Nonhydrostatic Atmospheric Dynamics," *Proc. Int'l Conf. High Performance Computing, Networking, Storage and Analysis*, 2016, p. 6.
6. H. Fu et al., "The Sunway TaihuLight Supercomputer: System and Applications," *Science China Information Sciences*, vol. 59, no. 7, 2016; doi:10.1007/s11432-016-5588-7.
7. M.C. Smith, J.S. Vetter, and X. Liang, "Accelerating Scientific Applications with the SRC-6 Reconfigurable Computer: Methodologies and Analysis," *Proc. 19th IEEE Int'l Parallel and Distributed Processing Symp.*, 2005; doi:10.1109/IPDPS.2005.75.
8. D. Oriato et al., "Acceleration of a Meteorological Limited Area Model with Dataflow Engines," *Proc. Symp. Application Accelerators in High Performance Computing*, 2012, pp. 129–132.
9. K. Sano, Y. Hatsuda, and S. Yamamoto, "Multi-FPGA Accelerator for Scalable Stencil Computation with Constant Memory Bandwidth," *IEEE Trans. Parallel and Distributed Systems*, vol. 25, no. 3, 2014, pp. 695–705.
10. X. Niu et al., "Exploiting Run-time Reconfiguration in Stencil Computation," *Proc. 22nd Int'l Conf. Field Programmable Logic and Applications*, 2012, pp. 173–180.
11. H.M. Waidyasooriya et al., "OpenCL-Based FPGA-Platform for Stencil Computation and Its Optimization Methodology," *IEEE Trans. Parallel and Distributed Systems*, vol. 28, no. 5, 2017, pp. 1390–1402.
12. K. Okina et al., "Power Performance Profiling of 3D Stencil Computation on an FPGA Accelerator for Efficient Pipeline Optimization," *ACM SIGARCH Computer Architecture News*, vol. 43, no. 4, 2016, pp. 9–14.

Lin Gan is a postdoctoral research fellow in the Department of Computer Science and Technology at Tsinghua University and the assistant director of the National Supercomputing Center in Wuxi. His research interests include high-performance computing solutions to geoscience applications based on hybrid platforms such as CPUs, FPGAs, and GPUs. Gan received a PhD in computer science from Tsinghua University. He has received the 2016 ACM Gordon Bell Prize, Tsinghua-Inspur Computational Geosciences Youth Talent Award, and the FPL Significant Paper award. He is a member of IEEE. Contact him at lingan@tsinghua.edu.cn.

Haohuan Fu is an associate professor in the Ministry of Education Key Laboratory for Earth System Modeling and in the Department of Earth System Science at Tsinghua University, and the deputy director of the National Supercomputing Center in Wuxi. He is the affiliated researcher in the Laboratory for Regional Oceanography and Numerical Modeling, Qingdao National Laboratory for Marine Science and Technology. His research interests include high-performance computing in earth and environmental sciences, computer architectures, performance optimizations, and programming tools in parallel computing. Fu received a PhD in computing from Imperial College London. He has received the 2016 ACM Gordon Bell Prize, Tsinghua-Inspur Computational Geosciences Youth Talent Award, and the FPL Significant Paper Award. He is a member of IEEE. He is the corresponding author of this article. Contact him at haohuan@tsinghua.edu.cn.

Wayne Luk is a professor of computer engineering at Imperial College London and the director of the EPSRC Centre for Doctoral Training in High Performance Embedded and Distributed Systems. His research focuses on theory and practice of customizing hardware and software for specific application domains, such as genomic data analysis, climate modeling, and computational finance. Luk received a doctorate in engineering and computing science from the University of Oxford. He is a Fellow of the Royal Academy of Engineering, IEEE, and the British Computer Society. Contact him at w.luk@imperial.ac.uk.

Chao Yang is a professor in and vice director of the Laboratory of Parallel Software and Computational Sciences, Institute of Software, Chinese Academy Sciences. His research interests include numerical analysis and modeling, large-scale scientific computing, and parallel numerical software. Yang received a PhD in computer science from the Institute of Software, Chinese Academy of Sciences. He has received the 2016 ACM Gordon Bell Prize. He is a member of IEEE and ACM. Contact him at yangchao@iscas.ac.cn.

Wei Xue is an associate professor in the Department of Computer Science and Technology at Tsinghua University. His research interests include scientific computing and uncertainty quantification. Xue received a PhD in electrical engineering from Tsinghua University. He has received the 2016 ACM Gordon Bell Prize and the Tsinghua-Inspur Computational Geosciences Youth Talent Award. He is a member of IEEE. Contact him at xuewei@tsinghua.edu.cn.

Guangwen Yang is a professor in the Department of Computer Science and Technology at Tsinghua University and the director of the National Supercomputing Center in Wuxi. His research interests include parallel algorithms, cloud computing, and the earth system model. Yang received a PhD in computer science from Tsinghua University. He has received the 2016 ACM Gordon Bell Prize. He is a member of IEEE. Contact him at ygw@tsinghua.edu.cn.