TSINGHUA SCIENCE AND TECHNOLOGY
ISSN 1007-0214 14/18 pp164–173
DOI: 10.26599/TST.2020.9010026
Volume 27, Number 1, February 2022

An MPI+OpenACC-Based PRM Scalar Advection Scheme in the GRAPES Model over a Cluster with Multiple CPUs and GPUs

Huadong Xiao*, Yang Lu, Jianqiang Huang, and Wei Xue

Abstract: A moisture advection scheme is an essential module of a numerical weather/climate model representing the horizontal transport of water vapor. The Piecewise Rational Method (PRM) scalar advection scheme in the Global/Regional Assimilation and Prediction System (GRAPES) solves the moisture flux advection equation based on PRM. Computation of the scalar advection involves boundary exchange, and computation of higher bandwidth requirements is complicated and time-consuming in GRAPES. Recently, Graphics Processing Units (GPUs) have been widely used to solve scientific and engineering computing problems owing to advancements in GPU hardware and related programming models such as CUDA/OpenCL and Open Accelerator (OpenACC). Herein, we present an accelerated PRM scalar advection scheme with Message Passing Interface (MPI) and OpenACC to fully exploit GPUs' power over a cluster with multiple Central Processing Units (CPUs) and GPUs, together with optimization of various parameters such as minimizing data transfer, memory coalescing, exposing more parallelism, and overlapping computation with data transfers. Results show that about 3.5 times speedup is obtained for the entire model running at medium resolution with double precision when comparing the scheme's elapsed time on a node with two GPUs (NVIDIA P100) and two 16-core CPUs (Intel Gold 6142). Further, results obtained from experiments of a higher resolution model with multiple GPUs show excellent scalability.

Key words: Graphics Processing Unit (GPU) computing; Open Accelerator (OpenACC); Message Passing Interface (MPI); Global/Regional Assimilation and Prediction System (GRAPES); Piecewise Rational Method (PRM) scalar advection scheme

1 Introduction

Moisture advection describes the transport of water vapor by wind, which plays a crucial role in the weather/climate model[1]. The Global/Regional Assimilation and Prediction System (GRAPES) Piecewise Rational Method (PRM) scalar advection scheme simulates the dynamic process of water vapor, which is an import part of the dynamic core in a Numerical Weather Prediction (NWP) model[2]. GRAPES is a multiscale hydrostatic and nonhydrostatic unified NWP system developed by the Numerical Weather Prediction Centre of China Meteorological Administration (CMA)[3]. It is a fundamental system for CMA's operational model sets, including global, regional, and ensemble models, and widely used in

• Huadong Xiao and Yang Lu are with the Institute of Geodesy and Geophysics, Chinese Academy of Sciences, Wuhan 430074, China, and also with the University of Chinese Academy of Sciences, Beijing 100049, China. E-mail: xiaohd@cma.gov.cn; luyang@whigg.ac.cn.
• Huadong Xiao is also with the National Meteorological Information Center, Beijing 100081, China.
• Jianqiang Huang and Wei Xue are with the Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China. E-mail: hjqxaly@163.com; xuewei@tsinghua.edu.cn.
• Jianqiang Huang is also with the Department of Computer Technology and Application, Qinghai University, Xining 810016, China.
∗ To whom correspondence should be addressed.
Manuscript received: 2020-07-30; accepted: 2020-08-18

China.

The future weather and climate model is advancing toward higher resolution, higher precision, more complicated physics processes, more coupled components, and more ensembles[4], so large amounts of computing power are required. This is a big challenge in terms of both the hardware and software and can be achieved using powerful and low-energy-consuming architectures. With the emergence of Graphics Processing Units (GPUs) as general-purpose computational architectures[5], the computation time of weather/climate forecasting models can be shortened by making full use of the power of GPUs[6].

Several examples that partially adapted GPUs in weather and climate prediction codes showed performance gains[7–17]. Especially, GPU acceleration of scalar or tracer advection modules using Compute Unified Device Architecture (CUDA) C/Fortran achieves an approximately three-fold speedup[12, 18, 19]. Most of these GPU adaptations use CUDA, a parallel computing platform for GPU computing and Application Programming Interface (API) model created by NVIDIA. However, porting of legacy Central Processing Units (CPU) based applications with CUDA often requires rewriting the code with a low-level API, which means making significant changes to the original system. Fortunately, Open Accelerator (OpenACC), a directive-based programming model such as Open Multi-Processing (OpenMP), provides a set of standards, high-level directives that enable C/C++ and Fortran programmers to utilize accelerators without significant programming effort[20]. With OpenACC directives, one can easily express the offloading of both computation and data from a host CPU device to an accelerator device GPU and obtain a single source code that can be run on a range of devices to achieve excellent performance.

OpenACC can be combined with Message Passing Interface (MPI) to exploit coarse- and fined-grained parallelism for computations. By combining it with MPI, multiple GPUs can be employed to solve a complex large-scale problem, in which MPI acts as a bridger for inter-GPU communication.

The PRM scalar advection scheme is a time-consuming module in GRAPES. When GRAPES at 25-km resolution is run with 512 parallel tasks on a supercomputer, the system consumes about 7% of the total run time, making it one of the most computationally time-consuming modules in GRAPES. Computation of the scalar advection scheme involves the repeated update of values of associated points on a multi-dimensional grid using only the benefits at a set of neighboring points. Most of the computations are independent, except for the data sharing between neighbors, and this typical characteristic can be exploited to accelerate computation using GPU or other accelerators.

In this paper, a parallel implementation of the PRM scalar advection scheme is proposed based on the combination of OpenACC and MPI. Several optimization strategies such as minimizing data transfers, loop restructuring to improve memory access and exposing more parallelism, and overlapping computation and transfers are applied for boosting performance. The experimental results for a medium workload and a 6-hour simulation by running the entire model in Double Precision (DP) demonstrate that when two NVIDIA P100 GPUs are used, a 3.5-fold speedup can be achieved compared with two Intel Gold 6142 CPUs. The computational performance of the accelerated scheme with multiple GPUs scales well with increasing number of GPUs.

The remainder of this paper is organized as follows. Section 2 presents the description of the PRM scalar advection scheme and its computation characteristics. Section 3 presents our computation method with CPUs and GPUs. Further, Section 4 illustrates the OpenACC implementation and optimization of the scheme. The results are analyzed and discussed in detail in Section 5. Finally, Section 6 concludes this paper.

2 Background

The governing prognostic equation for moisture in GRAPES[21], is given by

$$\frac{\mathrm{d}q}{\mathrm{d}t} = S(q) \tag{1}$$

where t denotes time, and $S(q)$ represents the source and sink term of water content q. Neglecting the source and sink term, Eq. (1) can be expressed in flux form in the 3D Cartesian coordinate system.

$$\frac{\partial q}{\partial t} + \frac{\partial qu}{\partial x} + \frac{\partial qv}{\partial y} + \frac{\partial qw}{\partial z} = q\Big(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z}\Big) \tag{2}$$

where u, v, and w are the velocity components in x, y, and z directions, respectively. Moreover, in the spherical terrain-following coordinate system, the flux form equation for water vapor is written as follows:

$$\frac{\partial q}{\partial t} + \frac{1}{r}\frac{\partial}{\partial \lambda}\Big(\frac{uq}{\cos\varphi}\Big) + \frac{1}{r}\frac{\partial v\cos\varphi q}{\partial \sin\varphi} + \frac{\partial q\hat{w}}{\partial \hat{z}} = q\Big(\frac{1}{r}\frac{\partial}{\partial \lambda}\Big(\frac{u}{\cos\varphi}\Big) + \frac{1}{r}\frac{\partial v\cos\varphi}{\partial \sin\varphi} + \frac{\partial \hat{w}}{\partial \hat{z}}\Big) \tag{3}$$

where $\hat{z}$ and $\hat{w}$ are the height and vertical velocity in the height-based terrain-following coordinate system, respectively, λ and φ represent the longitude and latitude, respectively, and r is the Earth's radius in the thin layer approximation. Using the time-splitting algorithm, Eq. (3) can be solved in three directions.

$$\begin{cases} \dfrac{\partial q}{\partial t}+\dfrac{1}{r}\dfrac{\partial}{\partial \lambda}\Big(\dfrac{uq}{\cos\varphi}\Big)=q\dfrac{1}{r}\dfrac{\partial}{\partial \lambda}\Big(\dfrac{u}{\cos\varphi}\Big), \\ \dfrac{\partial q}{\partial t}+\dfrac{1}{r}\dfrac{\partial v\cos\varphi q}{\partial \sin\varphi}=q\dfrac{1}{r}\dfrac{\partial v\cos\varphi}{\partial \sin\varphi}, \\ \dfrac{\partial q}{\partial t}+\dfrac{\partial q\hat{w}}{\partial \hat{z}}=q\dfrac{\partial \hat{w}}{\partial \hat{z}} \end{cases} \tag{4}$$

The terms on the left side of Eq. (4) can be calculated using the 1D advection equation, and those right-side can be calculated by correction of divergence after advection.

For brevity, the flux form of the 1D advection equation is expressed as

$$\frac{\partial f}{\partial t}+\frac{\partial f u}{\partial x}=0 \tag{5}$$

where f is the scalars of advection, time, coordinate in a space, and velocity, respectively. By integrating the equation in the control grid cell $[x_{i-1/2}, x_{i+1/2}]$, Eq. (5) can be discretized as

$$\bar{f}_i^{n+1}=\bar{f}_i^{n}-\frac{1}{\Delta x}(g_{i+1/2}-g_{i-1/2}) \tag{6}$$

where n is the time-level, $\Delta x = x_{i+1/2}-x_{i-1/2}$, $g_{i-1/2}$ and $g_{i+1/2}$ represent the flux at the left and right boundary of grid i, respectively, and $\bar{f}_i^n$ represents the average value of integration at grid cell i.

$$\bar{f}_i^{n}=\frac{1}{\Delta x_i}=\frac{1}{\Delta x}\int_{x_{i-1/2}}^{x_{i+1/2}} F_i(x)\mathrm{d}x \tag{7}$$

where $F_i(x)$ is the function obtained using the fitted algorithm in the i-th grid cell. Then the PRM scalar advection algorithm is defined as the rational function to fit the distribution of scalars in the domain.

$$F_i(x)\equiv R_i(x)=\frac{a_i+2b_i(x-x_{i-1/2})+\beta_i b_i(x-x_{i-1/2})^2}{[1+\beta_i(x-x_{i-1/2})]^2}, \quad x\in[x_{i-1/2}, x_{i+1/2}] \tag{8}$$

The function $F_i(x)$ is under the limitation of the average value of integration and boundary value for grid cell i.

$$\begin{cases} F_i(x_{i-1/2})=f_{i-1/2}, \\ F_i(x_{i+1/2})=f_{i+1/2}, \\ \dfrac{1}{\Delta x_i}\int_{x_{i-1/2}}^{x_{i+1/2}} F_i(x)\mathrm{d}x=\bar{f}_i \end{cases} \tag{9}$$

where $f_{i-1/2}$ and $f_{i+1/2}$ are the left and right boundary values of grid cell i, respectively. The parameters in $R_i(x)$ can be obtained by solving Eqs. (8) and (9).

$$\begin{cases} a_i=f_{i-1/2}, \\ b_i=\beta \bar{f}_i+\dfrac{1}{\Delta x_i}(\bar{f}_i-f_{i-1/2}), \\ \beta_i=\dfrac{1}{\Delta x_i}\left(\dfrac{f_{i-1/2}-\bar{f}_i}{\bar{f}_i-f_{i+1/2}}\right) \end{cases} \tag{10}$$

As long as the boundary values of $f_{i\pm1/2}$ are determined, the function $F(x)$ can be calculated. In the PRM scalar advection scheme, the boundary value is obtained by interpolating the average value of integration of neighboring points[22].

In this way, the rational function of $R_i(x)$ is constructed, and $\bar{f}_i^{n+1}$ can be calculated from Eqs. (6) and (7). In Eq. (6), the boundary flux $g_{i+1/2}$ can be expressed as

$$g_{i+1/2}=\int_{t^n}^{t^{n+1}}\{\min(0,u_{i+1/2})R_{i+1}^{+}[x_{i+1/2}-u_{i+1/2}(t-t^n)]-\max(0,u_{i+1/2})R_i^{-}[x_{i+1/2}-u_{i+1/2}(t-t^n)]\}\mathrm{d}t \tag{11}$$

Then,

$$g_{i+1/2}=\begin{cases} -\dfrac{a_{i+1}^{+}\xi+b_{i+1}^{+}\xi^2}{1+\beta_{i+1}^{+}\xi}, & u_{i+1/2}\leqslant 0; \\ -\dfrac{a_{i+1}^{-}\xi+b_{i+1}^{-}\xi^2}{1+\beta_{i+1}^{-}\xi}, & u_{i+1/2}>0 \end{cases} \tag{12}$$

where $\xi=\int_{t^n}^{t^{n+1}} u(t)_{i+1/2}\mathrm{d}t$.

To solve the computational stability and simulate the moisture distribution better over the polar region, a polar mixing technique and a time-splitting advection algorithm with corrections are introduced. The modified polar mixing technique with a transition stage to solve the transition problem between mixed and unmixed regions is used in GRAPES.

$$\begin{cases} c_{i,1}=c_{\text{ave1}}, \\ c_{i,2}=0.7c_{\text{ave2}}+0.3c_{i,2}, \\ c_{i,3}=0.4c_{\text{ave3}}+0.6c_{i,3}, \\ c_{i,4}=0.1c_{\text{ave4}}+0.9c_{i,4} \end{cases} \tag{13}$$

where $c_{\text{ave}j}$ is the average value of c at the j-th latitude circle, $c_{\text{ave}j}=\frac{1}{I}\sum_{i=1}^{I} c_{i,j}$, i is the total number of counts of a grid cell along the latitude circle, $j=1,2,3$, and 4 are the corresponding latitude circles for transfering water content from the polar region to the equator, and c is the integration value of the concentration of water content over the grid cell.

From the aforementioned mathematical model, we can see that for each grid point on a 3D grid, the computation of the PRM scalar advection scheme updates its value with a function of the value at its six neighboring points over a number of time steps. This problem is a type of stencil computation. In the calculation of the scheme, the 7-point stencil in 3D space is transformed into a 3-point stencil in 1D space along the three dimensions of i, j, k (Fig. 1). The computing advection for a grid point depends on itself and its neighbor points. Thus, in the parallel computation with MPI for the scheme, different pieces of computation have to communicate with each other for boundary exchange, which is unlike the high parallelism in the parameterization schemes for physical processes. This data dependence of applying local value of neighboring points adds considerable complexity to achieving an effective parallel computation. A simplified 3-point stencil computation of a 1D space taken as an example to indicate the ratio of computation to memory access can be represented by Eq. (14):

$$h_{\text{new}}(i) = \alpha \times h(i) + \gamma \times [h(i-1) + h(i+1)] \quad (14)$$

where h and h_{new} are the 1D array at the current/next time and α and γ are two scaled factors, respectively. In Eq. (14), there are three reading operations (ops), one writing ops, two adding ops, and two multiplying ops. For this double-precision computation, the ratio of computation to memory access is defined as FLOP/byte i.e., $4/[(3+1)\times 8] = 0.125$, which is a very low FLOP/byte ratio, where FLOP represents floating-point operations. Thus, it is difficult to optimize the computation with this low value, especially for multiple GPUs. Moreover, the parallel computation of the PRM scalar advection scheme with MPI and GPU is not easy, especially for accelerating performance. Therefore, an appropriate method should be considered.

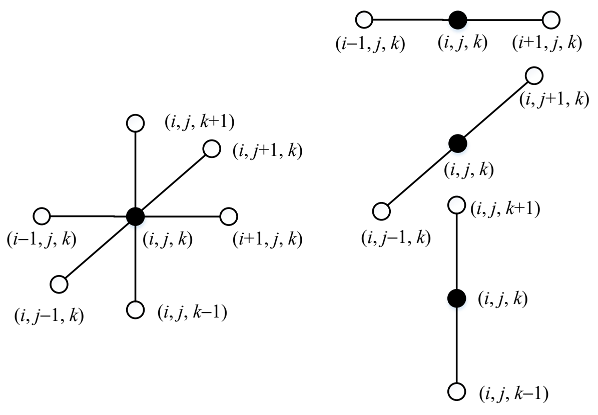

Fig. 1 7-point stencil of a 3D space and 3-point of 1D space along *i*, *j*, and *k* dimensions. The computation on the grid with filled circle depends on that of the grid with the surround hollow circles.

3 Computation with Multiple CPUs and GPUs: Our Method

3.1 Algorithm

The entire GRAPES code is used for the GPU porting of the PRM scalar advection scheme only by adding or changing code relevant to the system. GRAPES uses MPI to decompose the problem space across the latitude and longitude dimensions. A GPU programming model, such as CUDA, OpenCL, or OpenACC, is used for the computation and data offloaded by the MPI task. In this hybrid MPI and GPU programming model, MPI is used across CPU cores, and a local "X" programming model of choice is used within the GPU attached to each core. Further, the code needs fewer changes because of the high-level programming model of OpenACC and is kept understandable and more uncomplicated by the programmer; moreover, single source code can be used to target both CPU only and hybrid CPU-GPU architectures, as the directives can be ignored when no GPU is available in the system. Several investigations have shown that the performance loss with OpenACC is due to the lack of support of sharing memory usage and banking conflicts, which is often not substantial, particularly on the most recent GPUs[11, 15]. We employed the hybrid MPI and OpenACC programming method. First, we added some codes to set up the running environment for GPUs. Second, OpenACC directives were inserted into the computation-intensive portions of the PRM scalar advection scheme, and the compiler automatically mapped it to the GPU. Then the code was added to the main program of GRAPES to obtain the number of available GPUs and assign a GPU to each MPI process. Each GPU was controlled by its dedicated MPI process(es). The way for an MPI process to manage a GPU can be implemented by attaching the identification (ID) of the GPU device to the ID of the MPI process. The implementation of hybrid OpenACC with MPI programming for the PRM scalar advection scheme is given by Algorithm 1.

3.2 Overlapping computation and data transfers

The technique of overlapping computation and data transfers was employed in the GPU implementation of the PRM scalar advection scheme to reduce the cost

Algorithm 1 Hybrid OpenACC and MPI computation of the PRM advection scheme

```
1: MPI environment initialization
2: $nproc \leftarrow$ number of processes
3: $myid \leftarrow$ process id
4: OpenACC initialization
5: $ngpus \leftarrow$ number of GPU devices on a node
6: $gid \leftarrow myid\%ngpus$    ▷ Attach each GPU with one or more processes
7: Set a desired GPU to perform computation
8: for $i \leftarrow 1, nstep$ do
9:     #pragma acc data copy($model\_data(:,:,:,:), i, ...$)    ▷ Set OpenACC data region for data movement
10:    {
11:    PRM_3D($model\_data(:,:,:,:), i, ...$)    ▷ Perform PRM scalar advection scheme computation on GPU
12:    }
13: End MPI environment

14: procedure PRM_3D($model\_data(:,:,:,:), i, ...$)
15:    #pragma acc data present($model\_data$) create($\cdots$)    ▷ Set OpenACC data region to indicate date resident on GPU
16:    {
17:    #pramga acc kernels loop    ▷ Set OpenACC compute region
18:    for $j \leftarrow jts, jte$ do    ▷ jts/jte, the start/end index along $j$ direciton for an MPI process
19:        #pragma acc loop
20:        for $k \leftarrow kts, kte$ do
21:            #pragma acc loop
22:            for $i \leftarrow its, ite$ do
23:                $model\_data(i,k,j) \leftarrow \cdots$
24:    UPDATEHALO($\cdots$) ▷ GPU data exchange between halos with MPI
25:    }
```

of data transfers and achieve excellent performance. The basic idea is that independent datasets can provide opportunities for overlapping computation with data transfers by asynchronous data movement and computation by exploiting the pipeline parallelism[23].

This overlapping technique exploits inter-variable independence. When one variable in the numerical weather models can be computed independently from another, the boundary exchange for this variable can be overlapped with the computation of another variable, as shown in Fig. 2. As the advection of water content

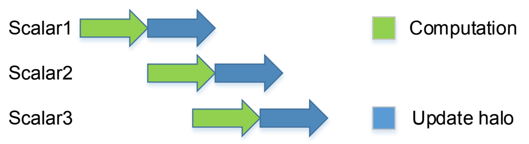

Fig. 2 Overlapping computation and data exchange (transfers) for different variables.

can be computed independently in GRAPES, the following overlapping techniques were applied to these computations.

4 OpenACC Implementation and Optimization

4.1 Expressing parallelism with OpenACC directives

Several compute regions are added around the parallel loops by inserting OpenACC directives. The directive of ACC KERNELS or ACC PARALLEL is inserted at the top of the nested loop inside each kernel, and the ACC LOOP directive is added right before each loop. Then the ACC END KERNELS or ACC END PARALLEL directive is inserted at the bottom of each kernel. Here, the kernel means the construct of a nested loop annotated with OpenACC instructions. With these added directives, numerous indications are given to the compiler where code needs to be processed in parallel. Data regions with OpenACC data directives are combined to manage data movement effectively. The ACC DATA COPYIN/COPYOUT/COPY directives are added just before calling the actual routine for the computation of the PRM scalar advection scheme and the ACC END DATA directive after calling the subroutine. These directives indicate movement of the input and output to the GPU and CPU, respectively. Furthermore, some other code of OpenACC data directive such as ACC DATA PRESENT combined with ACC CREATE directive is added before the first region of the actual computing code. In the actual computing code, the added present clause indicates that the data have been already available on the GPU, and thus data movement is not required. The create clause means that the data are only created on the GPU and not needed to be copied to or from the GPU. Optimization of data locality and minimization of data transfer between the CPU host and GPU device through PCIe can be archived by adding data regions at the outermost place of the relevant code of the PRM scalar advection scheme.

The array variables required in the ACC DATA PRESENT directive can be obtained from the intent attribute of the array definition. Contrarily, variables required for ACC CREATE directive are obtained from the definition of local automatic arrays. As the computation of the PRM scalar advection scheme involves exchanging boundary data, the halo-update functions from GRAPES Parallel Programming Interface

(PPI) enveloped with many MPI functions are called at several places of the code. OpenACC directives such as ACC UPDATE HOST/DEVICE or ACC HOST_DATA USE_DEVICE are added to maintain data coherence between CPU and GPU in PPI. A GPU-based accelerated version with OpenACC can be produced by the steps described below.

4.2 Optimization

For analyzing and improving performance, the NVIDIA profiler tool "nvprof" was used for hotspot and bottleneck analysis of the routine. The screenshot in Fig. 3 shows NVIDIA visual profiler for the initial implementation running at a horizontal resolution of 0.25°. There were no unexpected GPU-CPU transfers and little overhead.

Systematic optimizations were performed in terms of data transfers, loop parallelism, and data sharing. We used different optimization techniques focused on the computation and memory access to maximize the throughput for the essential kernels.

4.2.1 Minimizing data transfers

To compute on a GPU, it is necessary to shift data from CPU/host memory to GPU memory and move back necessary data from GPU memory to CPU/host memory. These processes consume a lot of time, which is of great importance in improving performance. During the initial implementation, we added a data region surrounding the part of the code where the actual computing subrouinte is called for the PRM scalar advection scheme to perform data movement between CPU and GPU once. The other place of code, where the data transfers are related to the handling of the halo regions. Two allocatable arrays were used to pack and unpack the halo data for arrays with dimensions of 2, 3, and 4 in the original version, enabling the efficient transfer of contiguous ranges between different CPUs. We used ACC DECLARE CREATE directive right after the declaration of the two allocatable arrays, thus avoiding duplicate memory requests, as only few places are involved in code array allocation and deallocation for different dimensions. Then we explicitly wrote a loop around the code with Fortran array syntax that was used to copy from the original arrays, and the COLLAPSE clause was added to the existing ACC LOOP directive to transform the subsequent tightly nested loops into a single loop. This restructuring and collapsing of loops can expose more parallelism and realize better memory coalescing.

Now the code obtained by minimizing data transfers was considered our baseline version. For the case of running with a horizontal resolution of 0.25°, the most time-consuming kernel of the 1D stencil computation in the latitude direction per step spends about 47 ms (see Fig. 4), which occupies 20.9% of the total GPU computation time as shown in Fig. 3. Optimizing this kernel is significant for improving the performance of the entire GPU implementation. Therefore, the roofline model is employed to optimize the accelerated PRM advection scheme code. The performance was plotted

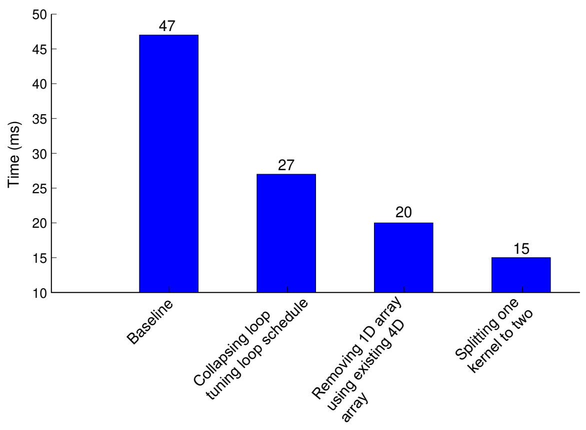

Fig. 4 Time required for different optimization techniques used in the kernel of 1D stencil computation in the latitude direction.

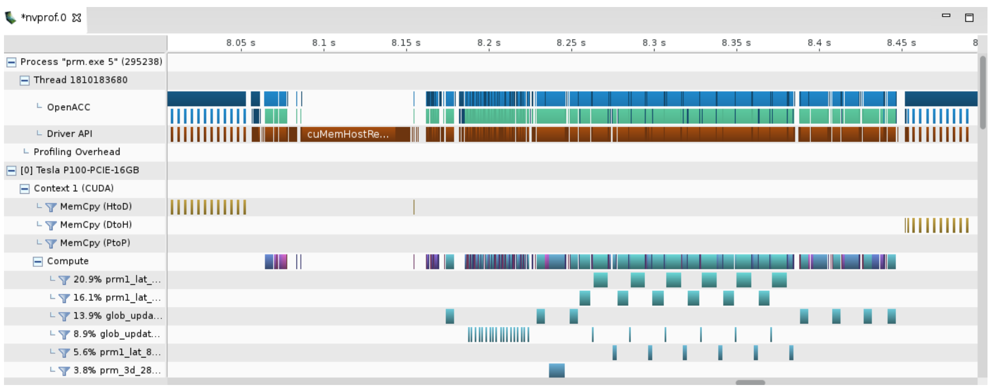

Fig. 3 Visual profiler for the timeline of the initial GPU implementation of the PRM scalar advection scheme.

against the Operational Intensity (OI) and compared with the theoretical limitations estimated from the peak sustained memory bandwidth and the maximum sustained performance[24]. The OI of this kernel in the initial version was 4.10, plotted as a vertical blue line. In Fig. 5, the blue line hits the slanted part of the roof (red line), indicating that the kernel of 1D stencil computation in the latitude direction is memory bound. Then memory access was mainly considered for optimizing the kernel.

4.2.2 Loop optimization

For most key kernels, optimizations such as loop restructuring, scalar replacement, and loop scheduling are employed to improve performance. Merging adjacent loops and replacing temporary arrays with scalars to allocate on registers is an effective way to boost the performance in terms of calculation time and data transfer overhead. This can be achieved by reducing the number of memory accesses and using a larger kernel to minimize the kernel launch overhead. Changing loop order and collapsing independent loops to increase parallelism and enable coalesced memory access also reduce calculation time. Other methods, such as loop fusion to form an appropriate size of the loop, can reduce the number of kernels and increase computational intensity. This contributes to the reduction in kernel launch overhead and synchronization and further improvement of performance. In some cases, performance can be improved using the manual setting gang, worker, and vector clause around loops to change the kernel scheduling during parallelism mapping to the GPU.

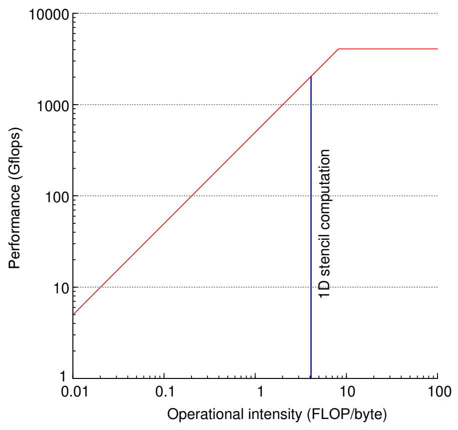

Fig. 5 Roofline plot for the 1D stencil kernel of the latitude direction on the NVIDIA P100 GPU. The blue line is plotted according to its operational intensity: line hitting the slanted part of the roof is limited by available memory bandwidth, and line hitting the flat part of the roof is limited by peak computational performance.

4.2.3 Data sharing

Computing advection for a single point requires data for some neighbor points. For the PRM scalar advection scheme, a 3-point stencil is required in each direction. For example, to execute the kernel of the 1D stencil computation along the latitude slice, each element in the slab is required for several threads; therefore, using shared memory for data sharing is beneficial. The OpenACC tile clause is adopted for this purpose.

Different optimizations of time for the critical kernel of 1D stencil computation in the latitude direction are shown in Fig. 4 as an example. When we collapsed two subsequent loops and set the gang worker clause for loop scheduling, the kernel time was reduced from 47 to 27 ms. Then a 1D array was removed for caching an existing 4D array and the 4D array was used directly, which exposed more parallelism and reduced memory usage. Accordingly, the time was further decreased to 20 ms. Finally, a large kernel was split into two; the time was reduced to 15 ms, which was about one-third of the time obtained in the baseline version.

The optimization procedure presented in Table 1 illustrates the above results. The ratio achieved for the DP peak performance of GPU and a few other metrics (measured by NVIDIA profiler tool "nvprof") are also given in Table 1 to provide a better understanding of the present performance gains. Altogether, a performance of more than 22.3% of NVIDIA P100 theoretical DP peak performance was achieved for this kernel; the achievable occupancy increased from 0.37 to 0.42. The kernel time in the third version was the second shortest, but with the smallest achievable occupancy. The attainable performance of a kernel is related to not only the achievable occupancy but also other factors[25].

Table 1 An illustrative example of the optimization procedure for the 1D stencil computation kernel in the latitude direction.

Version	FLOP efficiency (ratio of achieved occupancy to the DP peak performance) (%)	Achieved occupancy	Register per thread
Baseline	7.46	0.37	26
Collapsing loop & tuning loop schedule	11.66	0.37	26
Removing 1D array & using existing 4D array	16.05	0.25	26
Splitting one kernel to two	22.30	0.42	28

During all these attempts for optimization, the incremental method was applied to make sure that the correct result was obtained.

5 Result and Discussion

In this section, performance results using GPUs with PGI Accelerator compilers are discussed. Four cases with a horizontal resolution of 2.5°, 2.0°, 1.0°, and 0.5° and 60 vertical layers in the GRAPES global model with 144×72×60, 180×90×60, 360×180×60, and 720×360×60 grid points were run on a single GPU node, respectively. Two other cases with a horizontal resolution of 0.5° and 0.25° and the same 60 vertical layers in GRAPES with 720×360×60 and 1440×7200×60 grid points, respectively, were run on multiple GPU nodes. The tests were performed on a Linux GPU cluster system with each node of dual 16-core 2.6-GHz Intel Xeon Gold 6142 CPUs and two 16-GB NVIDIA P100 GPUs. Each node was equipped with 128-GB memory and a 100-Gbps Intel Omni-Path adaptor. The cluster ran on Red Hat Enterprise Linux Server release 7.2. All the results were obtained using the PGI Accelerator compiler 18.4 with the compiling option, which supports fast, vectorizing, binding, and target architecture optimization. Time reported was obtained by averaging five separate runs. Regarding the correctness of the results, the relative error was in the range of the tolerance of machine precision.

5.1 Single node

A 6-hour simulation with the initial conditions set by the system initialization component of GRAPES was performed. The time per step for both CPU and GPU computation on a single node is shown in Table 2.

The total CPU time was obtained by running the model with 32 MPI processes using all the CPU cores of a node in parallel mode. The total GPU run time was divided into two parts: actual GPU calculation time and data transfer time. Here one GPU was controlled by one MPI task. The speedup was expressed in terms of the total time and defined by the total CPU time of computation with 2 CPUs (32 cores) over the GPU time of that with 2 GPUs. The values obtained were 1.76, 1.94, 2.89, and 3.51 for the 2.5°, 2.0°, 1.0°, and 0.5° cases, respectively. Especially for the 0.5° case, the GPU time was less than 30% of the CPU time. When the GRAPES model resolution for the running example was increased, the GPU computation showed much greater advantage over the CPU computation. This can be attributed to the computation of a much greater number of grid points by the GPU, which can be performed concurrently and efficiently. A small workload running with low grid resolution of the GRAPES model did not saturate a GPU. Furthermore, if the GPU data transfer overhead was not considered, for the actual calculation time and the same problem size, the GPU computing performance was much better. Because the current exact grid resolution of the GRAPES global model system in operation is no less than 0.5°, over three times speedup can be obtained for the accelerated PRM-based scalar advection scheme with OpenACC and MPI using two GPUs over two CPUs within a single node.

Table 2 Time on CPU and GPU platform for the case with 2.5°, 2.0°, 1.0°, and 0.5° horizontal resolution.

Resolution (°)	CPU core	GPU	Time (ms)			Speedup
			Calculation	Transfer	Total	
2.5	32	–	–	–	65	–
	2	2	27	10	37	1.76
2.0	32	–	–	–	95	–
	2	2	33	16	49	1.94
1.0	32	–	–	–	358	–
	2	2	69	55	124	2.89
0.5	32	–	–	–	1356	–
	2	2	173	213	386	3.51

5.2 Multiple nodes

A larger grid size can be used to saturate the GPUs when more GPUs are used. The GRAPES domains covered with 720×360×60 and 1440×720×60 grid points were adopted to test the performance of multiple GPU nodes. Each node was set with two GPUs as mentioned above. The total GPU time per step on multiple GPUs is shown in Fig. 6. When the number of GPUs used on multiple nodes was increased, the

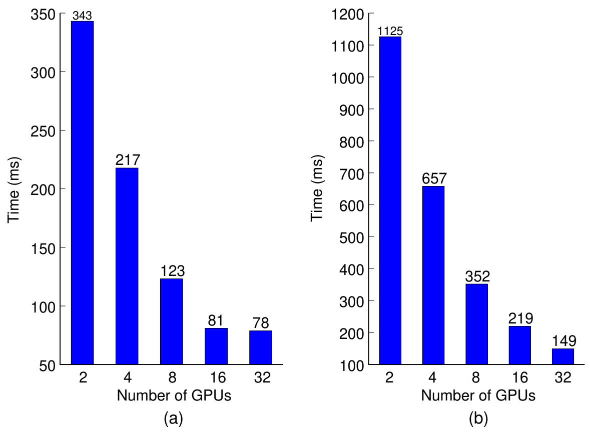

Fig. 6 GPU time per step for the running case with (a) 0.5° horizontal resolution and (b) 0.25° horizontal resolution.

total GPU time deceased considerably. In addition, the magnitude of time reduction decreased with increasing number of GPUs, especially for the smaller case with 0.5° horizontal resolution. It is particularly obvious for the 0.5° case that the total time was almost the same when the number of used GPUs was increased from 16 to 32. This can be explained by the fact that the workload assigned to each GPU is not sufficient for exploiting the power of GPUs with increasing number of GPUs and decreasing grid points for calculation. For a proper-sized problem, such as the high resolution of the 0.25° case, the accelerated computation of PRM-based scalar advection scheme scales well with increasing number of GPUs.

6 Conclusion

In this paper, we present an MPI+OpenACC-based PRM scalar advection scheme over a cluster with multiple CPUs and GPUs to exploit the computational capacity of multiple GPUs. In our proposed scheme, the total run time of a computing node with two GPUs is about 70% less than the whole run time of a computing node with two CPUs (with 32 cores each) for a medium-sized problem. Comparing the performance of the two GPUs over two CPUs (all cores used) in the same generation, the obtained speedup factor is 3.51. Moreover, the GPU implementation scales well with increasing number of GPUs for large-sized problems. Overall, these results are encouraging because in an adequately adapted GPU code, there will be overhead only when I/O is requested. Therefore, the overhead as a percent of the computation time will be minimal. Using an MPI+OpenACC directive-based programming method not only is the parallel development of code simplified but also achieves considerable performance, which gives a valuable reference for the future development of other such systems.

Acknowledgment

This work was partially supported by the decision support project of response to climate change of China, the National Natural Science Foundation of China (Nos. 41674085, 41604009, and 41621091), the Natural Science Foundation of Qinghai Province (No. 2019-ZJ-7034), and the Open Project of State Key Laboratory of Plateau Ecology and Agriculture, Qinghai University (No. 2020-zz-03).

References

[1] W. H. Raymond, Moisture advection using relative humidity, *J. Appl. Meteor.*, vol. 39, no. 12, pp. 2397–2408, 2000.

[2] Y. Su, X. S. Shen, X. D. Peng, X. L. Li, X. J. Wu, S. Zhang, and X. Chen, Application of PRM scalar advection scheme in GRAPES global forecast system, (in Chinese), *Chinese Journal of Atmospheric Sciences*, vol. 37, no. 6, pp. 1309–1325, 2013.

[3] D. H. Chen, J. S. Xue, X. S. Yang, H. L. Zhang, X. S. Shen, J. L. Hu, Y. Wang, L. R. Ji, and J. B. Che, New generation of multi-scale NWP system (GRAPES): General scientific design, *Chin. Sci. Bull.*, vol. 53, no. 22, pp. 3433–3445, 2008.

[4] P. Bauer, A. Thorpe, and G. Brunet, The quiet revolution of numerical weather prediction, *Nature*, vol. 525, no. 7567, pp. 47–55, 2015.

[5] J. D. Owens, M. Houston, D. Luebke, S. Green, J. E. Stone, and J. C. Phillips, GPU computing, *Proc. IEEE*, vol. 96, no. 5, pp. 879–899, 2008.

[6] M. W. Govett, J. Middlecoff, and T. Henderson, Running the NIM next-generation weather model on GPUs, in *Proc. 2010 10th IEEE/ACM Int. Conf. Cluster*, *Cloud and Grid Computing*, Melbourne, Australia, 2010, pp. 792–796.

[7] J. Michalakes and M. Vachharajani, GPU acceleration of numerical weather prediction, *Parallel Process. Lett.*, vol. 18, no. 4, pp. 531–548, 2008.

[8] T. Henderson, J. Middlecoff, J. Rosinski, M. Govett, and P. Madden, Experience applying fortran GPU compilers to numerical weather prediction, in *Proc. 2011 Symp. Application Accelerators in High-Performance Computing*, Knoxville, TN, USA, 2011, pp. 34–41.

[9] I. Demeshko, N. Maruyama, H. Tomita, and S. Matsuoka, Multi-GPU implementation of the NICAM atmospheric model, in *Proc. 18th Int. Conf. Parallel Processing Workshops*, Rhodes Island, Greece, 2012, pp. 175–184.

[10] C. Yang, W. Xue, H. H. Fu, L. Gan, L. F. Li, Y. T. Xu, Y. T. Lu, J. C. Sun, G. W. Yang, and W. M. Zheng, A peta-scalable CPU-GPU algorithm for global atmospheric simulations, in *Proc. 18th ACM SIGPLAN Symp. Principles and Practice of Parallel Programming* (*PPoPP'13*), Shenzhen, China, 2013, vol. 48, pp. 1–12.

[11] X. Lapillonne and O. Fuhrer, Using compiler directives to port large scientific applications to GPUs: An example from atmospheric science, *Parallel Process. Lett.*, vol. 24, no. 1, p. 1450003, 2014.

[12] M. Norman, J. Larkin, A. Vose, and K. J. Evans, A case study of CUDA Fortran and OpenACC for an atmospheric climate kernel, *J. Comput. Sci.*, vol. 9, pp. 1–6, 2015.

[13] M. Huang, J. Mielikainen, B. Huang, H. Chen, H. L. A. Huang, and M. D. Goldberg, Development of efficient GPU parallelization of WRF Yonsei University planetary boundary layer scheme, *Geosci. Model Dev.*, vol. 8, pp. 2977–2990, 2015.

[14] O. Fuhrer, T. Chadha, T. Hoefler, G. Kwasniewski, X. Lapillonne, D. Leutwyler, D. Lüthi, C. Osuna, C. Schär, T. C. Schulthess, et al., Near-global climate simulation at 1km resolution: Establishing a performance baseline on 4888 GPUs with COSMO 5.0, *Geosci. Model Dev.*, vol. 11, pp. 1665–1681, 2018.

[15] J. Q. Huang, W. T. Han, X. Y. Wang, and W. G. Chen, Heterogeneous parallel algorithm design and performance optimization for WENO on the Sunway Taihulight supercomputer, *Tsinghua Science and Technology*, vol. 25, no. 1, pp. 56–67, 2020.

[16] R. Kelly, GPU computing for atmospheric modeling, *Comput. Sci. Eng.*, vol. 12, no. 4, pp. 26–33, 2010.

[17] S. Saarinen, D. Salmond, and R. Forbes, Preparation of IFS physics for future architectures, presented at the 16th ECMWF Workshop on High Performance Computing in Meteorology, Reading, UK, 2014.

[18] I. Carpenter, R. K. Archibald, K. J. Evans, J. Larkin, P. Micikevicius, M. Norman, J. Rosinski, J. Schwarzmeier, and M. A. Taylor, Progress towards accelerating HOMME on hybrid multi-core systems, *International Journal of High Performance Computing Applications*, vol. 27, no. 3, pp. 335–347, 2013.

[19] W. Vanderbauwhede and T. Takemi, An investigation into the feasibility and benefits of GPU/Multicore acceleration of the weather research and forecasting model, in *Proc. 2013 Int. Conf. High Performance Computing & Simulation* (*HPCS*), Helsinki, Finland, 2013, pp. 482–489.

[20] R. Farber, *Parallel Programming with OpenACC*. Cambridge, MA, USA: Morgan Kaufmann Publishers, 2016.

[21] J. S. Xue and D. H. Chen, *Scientific Design and Application of Numerical Prediction System GRAPES*, (in Chinese). Beijing, China: Science Press, 2008.

[22] P. Colella and P. R. Woodward, The piecewise parabolic method (PPM) for gas-dynamical simulations, *J. Comput. Phys.*, vol. 54, no. 1, pp. 174–201, 1984.

[23] T. Shimokawabe, T. Aoki, C. Muroi, J. Ishida, K. Kawano, T. Endo, A. Nukada, N. Maruyama, and S. Matsuoka, An 80-Fold speedup, 15.0 TFlops full GPU acceleration of non-hydrostatic weather model ASUCA production code, in *Proc. 2010 ACM/IEEE Int. Conf. High Performance Computing, Networking, Storage and Analysis*, New Orleans, LA, USA, 2010, pp. 1–11.

[24] S. W. Williams, A. Waterman, and D. A. Patterson, Roofline: An insightful visual performance model for multicore architectures, *Communications of the ACM*, vol. 52, no. 4, pp. 65–76, 2009.

[25] V. Volkov, Better performance at lower occupancy, presented at GPU Technology Conf. 2010, San Jose, CA, USA, 2010.

Huadong Xiao is currently pursuing the PhD degree in geodesy and surveying engineering at the Institute of Geodesy and Geophysics, Chinese Academy of Sciences. He is a senior engineer at National Meteorological Information Center of Chinese Meteorological Administration. His research interest includes high performance computing in meteorology and geodesy.

Yang Lu received the PhD degree from the Institute of Geodesy and Geophysics, Chinese Academy of Sciences in 1997. He is a professor at the same institute. His research interest includes geodesy, geophysics, polar glaciology, and data processing.

Jiangqiang Huang is an associate professor at Qinghai University, China. He is currently a PhD candidate in the Department of Computer Science and Technology, Tsinghua University. His research interest includes high performance computing.

Wei Xue received the BEng and PhD degrees in electrical engineering from Tsinghua University in 1998 and 2003, respectively. He is an associate professor and the director of High Performance Computing Institute in the Department of Computer Science and Technology, Tsinghua University. His research interest includes scientific computing and uncertainty quantification.