2014 IEEE 28th International Parallel & Distributed Processing Symposium

Balancing CPU-GPU Collaborative High-order CFD Simulations on the Tianhe-1A Supercomputer

Chuanfu Xu*, Lilun Zhang*, Xiaogang Deng*, Jianbin Fang†, Guangxue Wang‡,
Wei Cao*, Yonggang Che*, Yongxian Wang* and Wei Liu*
*College of Computer Science, National University of Defense Technology, Changsha 410073, P.R. China
†Parallel and Distributed Systems Group, Delft University of Technology, Delft 2628CD, The Netherlands
‡State Key Laboratory of Aerodynamics, P.O. Box 211, Mianyang 621000, P.R. China

***Abstract*—HOSTA is an in-house high-order CFD software that can simulate complex flows with complex geometries. Large-scale high-order CFD simulations using HOSTA require massive HPC resources, thus motivating us to port it onto modern GPU-accelerated supercomputers like Tianhe-1A. To achieve a greater speedup and fully tap the potential of Tianhe-1A, we *collaborate* CPU and GPU for HOSTA instead of using a naive *GPU-only* approach. We present multiple novel techniques to balance the loads between the *store-poor* GPU and the *store-rich* CPU, and overlap the collaborative computation and communication as far as possible. Taking CPU and GPU load balance into account, we improve the maximum simulation problem size per Tianhe-1A node for HOSTA by 2.3X, meanwhile the collaborative approach can improve the performance by around 45% compared to the GPU-only approach. Scalability tests show that HOSTA can achieve a parallel efficiency of above 60% on 1024 Tianhe-1A nodes. With our method, we have successfully simulated China's large civil airplane configuration *C919* containing 150M grid cells. To our best knowledge, this is the first paper that reports a CPU-GPU collaborative high-order accurate aerodynamic simulation result with such a complex grid geometry.**

I. Introduction

The solution of the Navier-Stokes equations for CFD (Computational Fluid Dynamics) involves reasonably complex numerical methods and algorithms which are used in computational science today. In the past several decades, low-order (order of accuracy ≤ 2) CFD schemes have been widely used in engineering applications, but they are insufficient in simulation resolution and fidelity for complex flows containing sharp gradients and small disturbances. An effective approach to overcome the obstacle of accurate numerical simulations is to employ high-order methods [1]. Over the past 30 years, various high-order schemes have been developed and applied in CFD. Among them, *compact high-order finite difference schemes* are one of the most promising schemes [2]. Based on a given compact stencil, compact schemes can achieve relative higher order of accuracy with good spectral resolution and flexibility over traditional explicit finite difference schemes. Recently, to capture discontinuities in flows with shock wave, nonlinear formulation is added and various non-linear compact schemes such as Weighted Essentially Non Oscillatory (WENO) [3] and Weighted Compact Non-linear Scheme (WCNS) [4] were developed. They have been extensively studied and successfully used for many complex flows, and are also very attractive for multi-scale flows (e.g., aero-acoustics, electro-magnetics and turbulence).

Although many researchers claim that they have successfully applied high-order schemes in complex geometries, the grids in their tests are fairly simple compared to those in low-order scheme applications. Problems such as robustness and grid-quality sensitivity [1], [5] still hinder the wide application of high-order schemes on complex meshes. Recently, we have shown that these problems can be largely mitigated by treating the Geometric Conservation Law (GCL) [6] and block-interface conditions carefully when employing high-order schemes on complex configurations. To this end, we have developed a Conservative Metric Method (CMM) [7] to fulfill GCL by computing grid derivatives in a conservative form, with the same scheme used for fluxes. Further, we have employed a Characteristic-Based Interface Condition (CBIC) [8] to fulfill high-order multi-block computing by directly exchanging the spatial derivatives on each side of an interface. Based on CMM and CBIC, we have successfully simulated a wide range of flow problems with complex grids using WCNS. The WCNS scheme, along with CMM and CBIC, have been implemented in our in-house high-order CFD software HOSTA (High-Order SimulaTor for Aerodynamics) for 3D multi-block structured grids.

Some recent applications of HOSTA and WCNS can be found in [7]–[10]. Meanwhile, running HOSTA with a fairly large grid on a local machine often takes several weeks or even months. Thus, it is essential and practical to port it onto modern supercomputers, often featuring many-core accelerators/co-processors (GPUs [11], MIC [12], or specialized ones [13]). These heterogeneous processors can dramatically enhance the overall performance of HPC systems with remarkably low total cost of ownership and power consumption, but the development and optimization of large-scale applications are also becoming exceptionally difficult. Researchers often need to use various programming models/tools (e.g., CUDA [14] for NVIDIA's GPUs, OpenMP and MPI for intra-/inter- node parallelization), and map a hierarchy of parallelism in problem domains to heterogeneous devices with different processing capabilities, memory availability, and communication latencies. Hence, for simplicity developers often choose a naive GPU-only approach in a GPU-accelerated systems: letting GPUs compute efficiently and CPUs only manage GPU computation and communication. This is obviously a vast waste of CPU capacity, especially for top supercomputers like Tianhe-1A [15] equipped with powerful multi-core CPUs. Our previous GPU-only implementation [16] shows that on one Tianhe-1A node (containing two Intel Xeon X5670 CPUs and one NVIDIA Tesla M2050 GPU) the achievable performance of HOSTA on X5670s is around 80% of that on a M2050. In other words, we can potentially obtain an 80% performance improvement by using both the CPUs and the GPU. Therefore, to tap the full potential of heterogeneous systems and maximize application performance, it is immensely important to collaborate CPU and GPU for a *CPU-GPU collaborative simulation* instead of a GPU-only simulation.

Nevertheless, making an efficient and large-scale collaboration for real-world complex CFD applications on heterogeneous supercomputers has been described as a problem as hard as any that computer science has faced. Besides the aforementioned programming challenge, we need to balance the use of unbalanced hardware resources: GPUs are relatively rich in compute capacity but poor in memory capacity and the opposite holds for CPUs; this is generally true for most current heterogeneous supercomputers. On each Tianhe-1A node the M2050 contributes about 515 GFlops with less than 3GB device memory, while the X5670s, with 32 GB shared memory, contribute only 140 GFlops. Our previous results [16] show that HOSTA achieves a speedup of about 1.3 when comparing one M2050 with two hexa-core X5670s, but it can hold a maximum of 2M grid cells on the M2050. Thus, taking CPU-GPU load balance into account, the maximum amount of cells on one Tianhe-1A node is around 3.5M, with 2M cells on the GPU and 1.5M ($\approx \frac{2M}{1.3}$) cells on the CPUs. Therefore, HOSTA's simulation capacity on a single Tianhe-1A node is bounded by the relatively small device memory on GPUs. For large-scale grids, we can either use more compute nodes to keep the amount of cells per node below 3.5M, or perform an inefficient and unbalanced collaboration. Both cases severely limit the efficiency of heterogeneous systems. Hence, achieving load balance is one of the most serious challenges we face when collaborating *store-poor* GPU with *store-rich* CPU for HOSTA on Tianhe-1A. Another practical challenge is that HOSTA performs more extensive data exchanges (specifically four exchanges in each iteration, see II-B) among neighboring grid blocks than traditional low order scheme applications to ensure the robustness of high-order simulations. Thus, we also need to design an effective mechanism for data transfers among different parallel levels of Tianhe-1A when scaling HOSTA.

In this paper, we tackle these challenges by implementing a novel balancing scheme for intra-node CPU-GPU collaboration and overlapping CPU-GPU computation with communication for massively parallel simulations. On a single Tianhe-1A node we collaborate the CPUs and the GPU using nested OpenMP and CUDA. The key idea is to surmount the GPU memory limit by a dynamic device memory use policy for selected flow variables, coupled with grouping and pipelining GPU-computed grid blocks using CUDA streams. The balanced collaborative approach can top up HOSTA's maximum simulation problem size on a single Tianhe-1A node from 3.5M cells to 8M cells (2.3X), and meanwhile has up to 45% performance advantage over the GPU-only approach. To scale HOSTA on large-scale compute nodes, we use non-blocking MPI, CUDA events and CUDA streams to perform inter-block data exchanges as early as possible and overlap multiple levels of computation and communication. Despite of the extensive data exchanges required in HOSTA, we achieve a parallel efficiency of about 60% on 1024 nodes. As a case study, we simulate a very complex grid (China's large civil airplane configuration *C919*) containing 150M cells. Our balanced and scalable collaborative approach can obtain the same performance, but uses 50% less compute nodes than the CPU-/GPU-only approach on Tianhe-1A. The reduction in compute nodes, with the reduction in inter-node communication and HPC network investment, leads to a decrease in power consumption and expenditure for large-scale high-order CFD simulations on heterogeneous systems. To our best knowledge, this is the first paper that reports a CPU-GPU collaborative high-order aerodynamic simulation result with such a complex grid geometry.

II. Numerical methods and HOSTA implementation

In this section, we briefly introduce numerical methods used in this work and then give the implementation of HOSTA.

A. Numerical methods

In Cartesian coordinates the governing equations (Euler or Navier-Stokes) in strong conservative form are

$$\frac{\partial Q}{\partial t} + \frac{\partial F}{\partial x} + \frac{\partial G}{\partial y} + \frac{\partial H}{\partial z} = 0 \qquad (1)$$

the equations are transformed into curvilinear coordinates by introducing the transformation $(x, y, z, t) \longrightarrow (\xi, \eta, \zeta, \tau)$

$$\frac{\partial \tilde{Q}}{\partial \tau} + \frac{\partial \tilde{F}}{\partial \xi} + \frac{\partial \tilde{G}}{\partial \eta} + \frac{\partial \tilde{H}}{\partial \zeta} = 0 \qquad (2)$$

where

$$\tilde{F} = \tilde{\xi}_t Q + \tilde{\xi}_x F + \tilde{\xi}_y G + \tilde{\xi}_z H, \tilde{\xi}_x = J^{-1}\xi_x \qquad (3)$$

and with similar relations for the other terms.

The fifth-order WCNS scheme WCNS-E-5 was used in this work. Here we only consider the discretization of the inviscid flux derivative along the ξ direction. The discretization along the other directions can be dealt in a similar manner. The interior scheme of WCNS-E-5 can be expressed as

$$\begin{aligned} \frac{\partial \tilde{F}_i}{\partial \xi} &= \frac{75}{64h}(\tilde{F}_{i+1/2} - \tilde{F}_{i-1/2}) \\ &-\frac{25}{384h}(\tilde{F}_{i+3/2} - \tilde{F}_{i-3/2}) + \frac{3}{640h}(\tilde{F}_{i+5/2} - \tilde{F}_{i-5/2}) \end{aligned} \qquad (4)$$

where h is the grid size, and $\tilde{F}_{i+1/2} = \tilde{F}(U^L_{i+1/2}, U^R_{i+1/2}, \tilde{\xi}_{x,i+1/2}, \tilde{\xi}_{y,i+1/2}, \tilde{\xi}_{z,i+1/2})$ is the cell-edge flux computed by some flux-splitting methods that can be found in [4]. $U_i = U(\xi_i, t)$ is the flow variables. $U^L_{i+1/2}$ and $U^R_{i+1/2}$ are the left-hand and right-hand cell-edge variables. They can be obtained by a high-order nonlinear weighted interpolation of cell-node variables. The boundary and near-boundary schemes of WCNS-E-5 can be found in [4]. For the viscous fluxes, we also use the same sixth order central difference scheme as described in [4].

The GCL includes the volume conservation law (VCL) and the surface conservation law (SCL). For stationary grids, VCL naturally holds

$$I_t = (1/J)_\tau + (\tilde{\xi}_t)_\xi + (\tilde{\eta}_t)_\eta + (\tilde{\zeta}_t)_\zeta = 0 \qquad (5)$$

if SCL is satisfied, then

$$\begin{aligned} I_x &= (\tilde{\xi}_x)_\xi + (\tilde{\eta}_x)_\eta + (\tilde{\zeta}_x)_\zeta = 0 \\ I_y &= (\tilde{\xi}_y)_\xi + (\tilde{\eta}_y)_\eta + (\tilde{\zeta}_y)_\zeta = 0 \\ I_z &= (\tilde{\xi}_z)_\xi + (\tilde{\eta}_z)_\eta + (\tilde{\zeta}_z)_\zeta = 0 \end{aligned} \qquad (6)$$

High-order schemes with their low dissipative property usually bear more risks from the SCL-related errors than that of low-order schemes. The CMM ensures the SCL in high-order difference schemes at the following two aspects:

- First, metrics are acquired through the "conservative forms"

$$\tilde{\xi}_x = (y_\eta z)_\zeta - (y_\zeta z)_\eta, \tilde{\eta}_x = (y_\zeta z)_\xi - (y_\xi z)_\zeta, \quad \tilde{\zeta}_x = (y_\xi z)_\eta - (y_\eta z)_\xi \tag{7}$$

  and with similar relations for the remain items.

- Second, on each grid direction, the algorithm of the derivatives in Eq.(7) shall be identical to that of flow fluxes where the metrics are re-discretized in combination with the flow fluxes. For example, the re-discretization scheme of WCNS-E-5 is

$$\frac{\partial a_i}{\partial \xi} = \frac{75}{64h}(a_{i+1/2} - a_{i-1/2}) - \frac{25}{384h}(a_{i+3/2} - a_{i-3/2}) + \frac{3}{640h}(a_{i+5/2} - a_{i-5/2}) \tag{8}$$

  where the cell-edge values can be computed by high-order linear interpolation.

Finally, we briefly introduce the CBIC. We define the transformation matrix P_{QVC} in terms of conservative variables Q and characteristic variables V_C

$$P_{QV_C} = \frac{\partial Q}{\partial V_C} \tag{9}$$

the CBIC are

$$\begin{aligned} \frac{\partial Q}{\partial t}|_L &= (A_s^+)|_L(RHS)|_L + (A_s^-)|_L(RHS)|_R \\ \frac{\partial Q}{\partial t}|_R &= (A_s^+)|_R(RHS)|_R + (A_s^-)|_R(RHS)|_L \end{aligned} \tag{10}$$

where $(A_s^+) = P_{QV_c}diag[(1 + sign(\lambda_i))/2]P_{QV_c}^{-1}$, $(A_s^-) = P_{QV_c}diag[(1 - sign(\lambda_i))/2]P_{QV_c}^{-1}$. $(A_s^+)|_{L,R}$ are calculated by variables on the interface, and $(RHS)|_{L,R}$ can be calculated by one-side differencing or other methods. Here, the subscripts "L" and "R" denote the variables and terms on an interface but belong to the left-hand-side sub-block and the right-hand-side sub-block, respectively. One can verify that $\frac{\partial Q}{\partial t}|_L = \frac{\partial Q}{\partial t}|_R$. Please refer to [8] for more details about the CBIC.

B. HOSTA Implementation

HOSTA is implemented in Fortran 90 and we present its main flowchart in Fig.1. At the beginning of each iteration (a time step or a sub-iteration of unsteady simulations), boundary conditions are applied. Then data exchange for primitive variables of ghost/singularity cells is performed. Singularity cells are those cells that belong to several grid blocks. They are often unavoidable in complex multi-block structured grids. HOSTA calculates the delta of spectral radius (`calc_RDT`) and time-step (`calc_DT`) before it calculates and exchanges the gradient of primitive variables(`calc_DPV`). WCNS is implemented when calculating viscous (`calc_vis`) and inviscid (`calc_invis`) fluxes for RHS. The RHS is also exchanged. Implicit methods including LU-SGS, PR-SGS, Jacobi and explicit Runge-Kutta method are implemented for time-marching. In this paper, we use the Jacobi iterative method (`sol_jacobi`) on GPUs. After HOSTA exchanges the delta of conservative variables, it updates the primitive variables and residuals and then completes the iteration. Flow

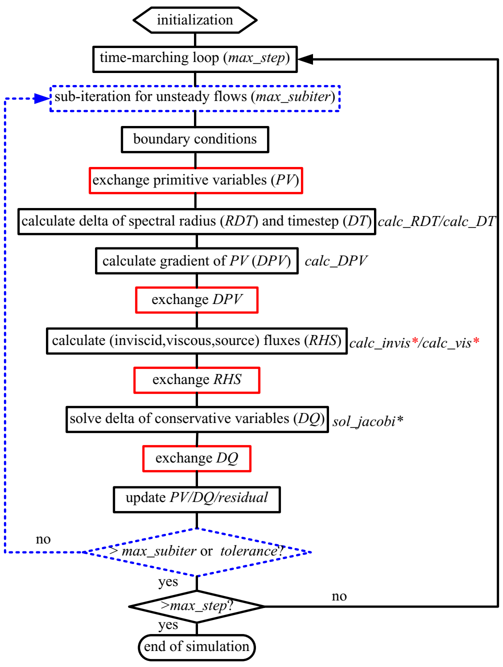
initialization
time-marching loop (*max_step*)
sub-iteration for unsteady flows (*max_subiter*)
boundary conditions
exchange primitive variables (*PV*)
calculate delta of spectral radius (*RDT*) and timestep (*DT*)
calc_RDT/calc_DT
calculate gradient of *PV* (*DPV*)
calc_DPV
exchange *DPV*
calculate (inviscid,viscous,source) fluxes (*RHS*)
*calc_invis**/*calc_vis**
exchange *RHS*
solve delta of conservative variables (*DQ*)
*sol_jacobi**
exchange *DQ*
update *PV/DQ/residual*
no
> *max_subiter* or *tolerance*?
yes
>*max_step*?
no
yes
end of simulation

Fig. 1. Flowchart of HOSTA. Some subroutines implementing the steps are also shown. Time-consuming subroutines are indicated by asterisk signs. WCNS are implemented in `calc_vis` and `calc_invis`, indicated by red asterisk signs. Note that each iteration has four exchanges (denoted by red rectangular) for ghost/singularity data to ensure the robustness of high-order schemes.

variables are declared as global variables and thus can be used in several subroutines conveniently. Those variables are allocated and initialized prior to the main time-marching loop, and are deallocated at the end of the simulation. In the MPI implementation, generally a large multi-block complex 3D grid is repartitioned, along three grid directions, into many grid blocks and those blocks are then grouped and distributed among MPI processes according to a load balance model [17]. Non-blocking MPI is used to overlap message passing and computation.

Generally, HOSTA organizes its multi-block calculation in a four-level loop: an outer *block-loop* specifying block index and a three-level inner *cell-loop* specifying cell index along the three directions. In [16], we present a dual-level GPU parallelization scheme to fully exploit parallelism in a multi-block loop. A cell-loop is implemented as a CUDA kernel, with each cell calculated by a CUDA thread, and a block-loop is mapped to a *stream-loop*, with grid blocks pipelined by multiple CUDA streams (*multi-stream*). We also perform particular optimizations for the time-consuming high-order flux calculating kernels. As a result, HOSTA achieves a speedup of about 1.3 when comparing one M2050 with two hexa-core X5670s (see [16] for more details).

III. The Collaborative Approach

A. Overall Domain Decomposition on Tianhe-1A

This work extends a MPI-CUDA implementation of HOSTA on Tianhe-1A [16] using MPI+OpenMP+CUDA.

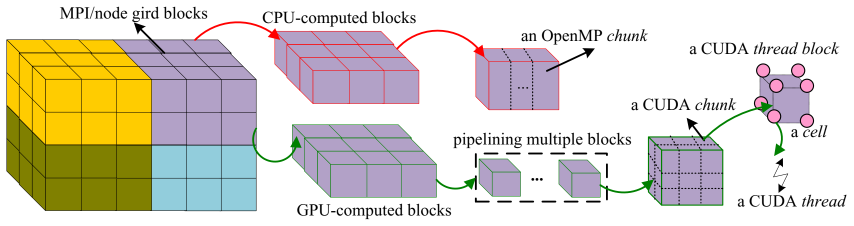

Fig. 2. Overall domain decomposition for MPI+OpenMP+CUDA parallelization. Grid blocks are first distributed among MPI processes/nodes and then on each node are further allocated between CPU and GPU.

```
streams(:) : CUDA stream arrays
gpu_blk=ceiling(gpu_ratio*nblk)
 ...
!$omp parallel num_threads(2)
 If (omp_get_thread_num().eq.0) then
    If  GBMS is defined  then    !GBMS implementation
        allocate device mem of num_grpstream blocks for mb_var
        Do stream_grp_id=1,num_grp
            Do  stream_id=1,num_grpstream
                nc=(stream_grp_id-1)*num_grpstream+stream_id
                If (nc<gpu_blk) then
                    H2D copy the nc-th mb_var using streams(stream_id)
                    Call calc_invis_gpu(…) using streams(stream_id)
                    D2H copy the nc-th mb_var using streams(stream_id)
                End If
            End Do
            synchronize streams
        End Do
        deallocate mb_var
    Else  !multi-stream implementation
        Do  stream_id=1,gpu_blk
            Call calc_invis_gpu(…) using streams(stream_id)
        End Do
    End If
 End If
 If (omp_get_thread_num().eq.1) then
        Call calc_invis_cpu(…)
 End If
!$omp end  parallel
 ...
```

Fig. 3. Pseudocode illustrating collaborative programming and GBMS in `calc_invis`. `mb_var` is a selected flow variable. The host and device kernels are encapsulated in `calc_invis_cpu` and `calc_invis_gpu` respectively

Tianhe-1A is a GPU-accelerated supercomputer developed by National University of Defense Technology (NUDT), China, and was ranked No.1 in the 36th Top500 list [18] for HPC systems in Nov, 2010. Fig.2 shows the overall domain decomposition for MPI+OpenMP+CUDA parallelization. The domain decomposition for MPI parallelization is same as described in Sect.II-B. We create one MPI process per node (of Tianhe-1A) to manage the two CPUs and the GPU. The *nblk* grid blocks on a MPI process/node is further allocated between the intra-node CPUs and GPU (see Sect.III-B). The OpenMP directives are added over the outer-most cell-loop, i.e., each CPU-computed grid block is logically partitioned to many *OpenMP chunks* and each chunk is calculated by an OpenMP thread. The single GPU parallelization is same as [16]. GPU-computed grid blocks are issued simultaneously by multiple CUDA streams, and each grid block is logically partitioned into many *CUDA chunks*; each CUDA chunk is updated by a CUDA thread block and finally, each cell in the chunk is calculated by a CUDA thread.

B. Intra-node collaborative programming

Since most OpenMP compilers support nested parallelism, we use this feature to coordinate the OpenMP parallelized CPU code and the CUDA parallelized GPU code. Fig.3 illustrates the collaborative programming as well as the balancing scheme (see Sect.III-C) for calculating the inviscid fluxes (`calc_invis`). The other calculating procedures can be implemented in a similar manner. To implement a collaborative calculation, we develop two different versions of `calc_invis`: the GPU version `calc_invis_gpu` wrapping CUDA kernels and the CPU version `calc_invis_cpu` parallelized with OpenMP. We create two OpenMP threads at the first-level parallel region. One thread calls `calc_invis_gpu` to launch CUDA kernels, dealing with the GPU-computed grid blocks. The other thread calls `calc_invis_cpu`, dealing with the CPU-computed grid blocks. In `calc_invis_cpu`, for each block we fork a second-level nested OpenMP threads (11 threads) on the multi-core CPUs. We call `cudaDeviceSynchronize` to perform a CPU-GPU synchronization before exchanging ghost/singularity data. During collaboration, heterogeneous devices are concurrently working on their own grid blocks.

We employ a static task/load partition for the intra-node CPUs and GPU: on each side all kernels always calculate the same data objects (specifically grid blocks in HOSTA) until the end of a simulation. This is based on an observation: in HOSTA (and also many other heterogeneous applications) the GPU code employs a static but efficient memory use policy similar to the CPU code. All flow variables of GPU-computed grid blocks are allocated and initialized before the time-marching loop, and deallocated until the end of the loop (see Sect.II-B). This can avoid unnecessary PCI-e data transfers during iterations, and we only need to transfer four flow variables (i.e., `RHS`, `DQ`, `PV` and `DPV`) when their ghost/singularity data exchanges are needed. The partition is implemented at the granularity of grid blocks. We use a parameter *gpu_ratio* to adjust the number of grid blocks simulated by each side

$$gpu\_blk = ceiling(gpu\_ratio \times nblk) \tag{11}$$

The GPU calculates $[1, gpu\_blk]$ blocks and the others are calculated on the CPUs. *gpu_ratio* is configured by profiling HOSTA's sustainable performance of one iteration on different devices. We set *gpu_ratio*=0 to perform a CPU-only simulation and then set *gpu_ratio*=1.0 to perform a GPU-only simulation. Taking the CPU performance as the baseline, *gpu_ratio* is evaluated as $\frac{SP_{GPU}}{(1.0+SP_{GPU})}$ (suppose the achieved speedup of GPU to CPU is SP_{GPU}).

Note that it is possible to implement a more fine-grained or dynamic partition like the approaches presented in [19], [20]. For example, in HOSTA we can precisely partition specific number of cells to different devices, or even assign different

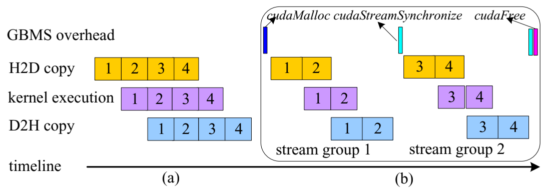

Fig. 4. Different schemes when pipelining 4 grid blocks: (a)multi-stream uses 4 streams; (b)GBMS uses 2 stream groups and each group contains 2 streams. It calls `cudaStreamSynchronize` to synchronize streams in each group and also calls `cudaMalloc/cudaFree` (and may also perform H2D/D2H copy) for selected flow variables.

number of cells for different kernels according to their execution characteristics on different devices. Those approaches generally require developers to reconstruct data structures, and meanwhile carefully manage data dependence among kernels and data transfers among devices. However, HOSTA has so many data objects/kernels implemented by different programming models on different devices that the changes will definitely make programming more difficult and may result in uncertain overall performance benefits. Therefore, the approaches in [19], [20] are often tested and applied in heterogeneous code implemented using a portable and unified language such as OpenCL [21] on different devices. Our partition approach in Eq.(11) is fairly simple but generic for multi-block CFD applications. Furthermore, we can adjust the partition granularity (e.g., grid block size) by independent mesh repartition tools [22].

C. Balancing intra-node CPU and GPU

Our results show that HOSTA runs about 1.3X faster on the M2050 than on the two X5670s ($gpu\_ratio = \frac{1.3}{2.3} \approx 0.57$), and thus we tend to distribute more grid blocks on the GPU. Suppose we have a large grid and each node is expected to simulate 6M cells, ideally we let the GPU simulate $6M * 0.57 \approx 3.4M$ cells and the CPUs simulate $2.6M (= 6M - 3.4M)$ cells according to their achievable performance. However, the M2050 can only hold 2M cells and the other 4M cells have to be distributed to the X5670s. Consequently, without a load-balancing scheme, for this case there is a significant imbalance of resource utilization: the GPU stays idle for most of the time while the CPUs are busy. Therefore, HOSTA's performance on a single Tianhe-1A node is bounded by the relatively small device memory on GPUs.

To mitigate the limit, we need to reduce the maximum device memory requirement of GPU-computed grid blocks. For this end, we extend the multi-stream scheme [16] in two steps. As a first step, we group multiple CUDA streams in multi-stream to form a Group-Based Multiple Streams (GBMS) scheme. In multi-stream, we use *gpu_blk* streams to issue the same number of grid blocks (see Fig.3 and Fig.4(a)). In GBMS, we divide *gpu_blk* streams into *num_grp* stream groups and each group contains *num_grpstream* streams (see Fig.3 and Fig.4(b)), i.e.,$gpu\_blk = num\_grp \times num\_grpstream$. As a second step, we add a dynamic device memory use policy in GBMS for particularly *selected flow variables* (e.g., `mb_var` in Fig.3). For the GPU-computed grid blocks, `mb_var` is also stored in the host main memory. The device memory of `mb_var` is allocated/deallocated dynamically before/after a CUDA kernel referencing it. Before executing the kernel, we copy it from the host memory to the device memory. After executing the kernel, we copy it back to the host memory if it is changed in the kernel. Fig.3 shows the pseudocode illustrating how GBMS is implemented in a collaborative simulation.

Compared with the multi-stream scheme, using GBMS can significantly reduce the requirement of device memory. Suppose for each block, `mb_var` requires D device memory, thus we need $gpu\_blk \times D$ device memory in multi-stream since those blocks are simultaneously executed on the GPU. With GBMS, the device memory requirement of `mb_var` is reduced to $num\_grpstream \times D$. Furthermore, in multi-stream the device memory of `mb_var` is persistent until the end of a simulation, while in GBMS the device memory of `mb_var` is only needed during the execution of a kernel and thus is temporary. In other words, the device memory of `mb_var` is deallocated immediately after the kernel finishes its execution; if the to-be-executed kernel do not use `mb_var`, we do not need to keep its device memory. We can use different $(num\_grp, num\_grpstream)$ configurations to further adjust the device memory requirement. For example, when pipelining 8 grid blocks on the GPU, $(num\_grp, num\_grpstream)$ can be set to (2,4) or (4,2). Obviously, the (4,2) configuration needs less device memory. For a large grid, users can adjust the configuration to check if the device memory is enough.

We specify selected flow variables based on the following principles:

- Variables used by a small number of CUDA kernels
- Variables that have to be transferred for ghost/singularity data exchanges
- Variables that occupy relatively large memory spaces
- Variables used in the CUDA kernels that are far faster than their counterpart Fortran subroutines

Those principles allow us to specify fewer selected flow variables, avoid significantly changing HOSTA code, and possibly minimize GBMS overhead (see next paragraph). The more selected flow variables we specify, the less device memory we require. In our implementation, we carefully specify 9 flow variables as selected flow variables, which accounts for about 36% of the total 25 flow variables and about 50% of the total device memory requirement. As a result, with GBMS we increase the maximum simulation problem size of HOSTA on the M2050 from 2M cells to about 4M cells.

At the same time, GBMS incurs an overhead on the GPU, due to dynamic device memory allocation/deallocation, H2D/D2H copy and stream synchronization between stream groups to ensure the completeness of asynchronous H2D/D2H copy. As Sect.IV-C1 shows, the relative speedup of one M2050 to two X5670s drops from 1.3 to around 1.0 in GBMS. GBMS also needs additional host memory to hold selected flow variables for GPU-computed grid blocks. The overhead is reasonable, mainly attributed to the employment of CUDA streams and carefully specifying selected flow variables, both of which can hide or minimize the GBMS overhead properly. GBMS provides a solid base for more flexible task partition and better load balance in a collaborative simulation. In the previous example where each node is expected to simulate 6M

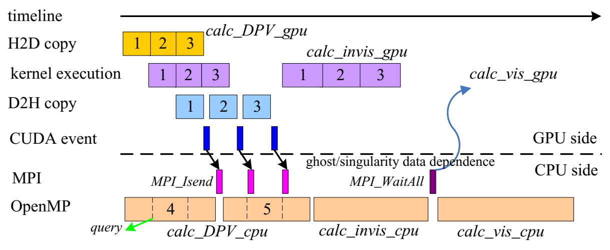

Fig. 5. Overlapping the collaborative CPU-GPU computation and communication for `calc_DPV`, `calc_invis` and `calc_vis`. Grid block *1*, *2* and *3* are calculated by the GPU. Grid block *4* and *5* are calculated by the CPUs.

cells, we can perform a well balanced collaborative simulation using GBMS: both the CPUs and the GPU simulates roughly 3M cells (i.e.,gpu_ratio=0.5).

D. Overlapping Computation and Communication

HOSTA has already implemented the overlapping of CPU computation and MPI communication by non-blocking MPI. For example, after calculating `DPV`, HOSTA calls `MPI_Isend` and `MPI_Irecv` to exchange `DPV` and the non-blocking MPI communication will be overlapped with the following `calc_invis`. Before using `DPV` in `calc_vis`, HOSTA calls `MPI_Waitall` to ensure that `MPI_Irecv` has finished receiving `DPV`. In [16], we also implemented a gather/scatter optimization to minimize CPU-GPU data transfers for ghost/singularity data of 3D grid blocks which are discontinuously stored in the device memory.

In this paper, using non-blocking MPI, CUDA multiple streams and CUDA events, we implemented an *as early as possible* non-blocking communication mechanism to further overlap the collaborative computation with PCI-e data transfers and MPI communication. Fig.5 illustrates the overlapping in `calc_DPV`, `calc_invis` and `calc_vis`. On the GPU side, when a stream finishes its operations of the associated grid block for `calc_DPV`, a CUDA event with the same stream ID is recorded to represent the data dependence between MPI communication and ghost/singularity data calculated by the stream. On the CPU side, we enable a counter in `calc_DPV`. During the execution of `calc_DPV`, for each CPU-computed gird block we query CUDA events several times (e.g., 3 times) to check if there are any GPU-computed grid blocks that have finished their calculations and intra-node data transfers, and also check the state of CPU-computed grid blocks. Then we call `MPI_Isend` or `MPI_Irecv` for ready grid blocks on both sides. Hence, we perform non-blocking communication for each grid block as early as possible, and this helps us better overlap the collaborative computation and the extensive high-order inter-block communication.

IV. Performance Results and Validation

A. HOSTA Implementation on Tianhe-1A

The GPU code is implemented using CUDA C. All CPU subroutines in the time-marching loop except those implementing MPI communications are rewritten as CUDA C kernels. Those kernels are wrapped in C functions to be called from Fortran code. We extend the main program loop to glue the GPU code with the CPU code for collaborative calculations. Since HOSTA is often used for high-resolution flow simulations, we always use a double precision implementation with ECC on. In total, we (4 full-time researchers and several part-time researchers) spent roughly 8 months for the MPI-CUDA version and another 3 months for the collaborative version. As we have mentioned, it is because we need to use diverse programming models/tools. Another important reason is that HOSTA is rather than a *toy* project, but a large-scale software containing more than 25000 lines of code. Totally we have added more than 16000 lines of CUDA C/C code and Fortran wrapper code for the final collaborative version. We believe it is also due to the time spent on finding and reducing bugs. By direct comparison of corresponding data fields, we make sure that the GPU-only version and the collaborative version always produces the same results as the CPU version up to machine accuracy. Debugging tends to be harder when various components/routines are tightly coupled, as changes in one may cause bugs to emerge in another. Therefore, making parallelization and optimization on heterogeneous supercomputers is still an exceptionally time-consuming and challenging job.

B. Experimental Setup and Metrics

We use CUDA of version 5.0, Intel icc11.1 for C code and Intel ifort11.1 for Fortran code. All code is compiled with the -O3 option. We use MPICH2-GLEX for MPI communication. We use automatically generated 3D airfoil configurations to facilitate performance evaluation. For each airfoil grid, users can specify the numbers of cells and partition parameters on three grid directions and obtain grid blocks of equal size. This can save a lot of grid generation time for complex geometries. We also present detailed timing results of *C919*. The total double precision floating point operation count on X5670s is measured by Intel Vtune amplifier xe 2011. Since we have no tools to count the GPU operations, we use the CPU operation count as a reference to estimate the GPU and the CPU+GPU machine efficiencies. To collect performance data, we simulate 10 time steps (*max_step*=10), with each having 50 sub-iterations (*max_subiter*=50), and average the execution time. We define the collaborative efficiency (*CE*) to evaluate the efficiency loss in CPU-GPU collaboration

$$CE = \frac{SP_{CPU+GPU}}{SP_{CPU} + SP_{GPU}} \times 100\% \qquad (12)$$

$SP_{CPU+GPU}$ is the achieved collaborative speedup, taking the performance of the two X5670s as the baseline. For example, if $SP_{CPU+GPU}$=1.8 and SP_{GPU}=1.3, then *CE* is $\frac{1.8}{1.0+1.3} \times 100\% \approx 78.3\%$, which represents an efficiency loss of around 22% in the collaboration.

Since we achieve a speedup of about 1.3/1.0 when comparing the M2050 to the X5670s without/with GBMS, gpu_ratio in Eq.(11) is set to 0.57 and 0.5 respectively. For simplicity, in default we fix the number of grid blocks ($\#block$) on each node to be 8[1]. Correspondingly, we set $(num\_grp, num\_grpstream)$=(2,2) in GMBS.[2]

[1]i.e., without GBMS, 3 blocks are calculated on the CPUs and the other 5 are calculated on the GPU; with GBMS, each side calculates 4 blocks.

[2]In practice, different $(num\_grp, num\_grpstream)$, gpu_ratio and $\#block$ may lead to different performance, but the maximum achieved performance is always within ±5% of the performance obtained with the default setting. Thus, we only present the results of the default setting.

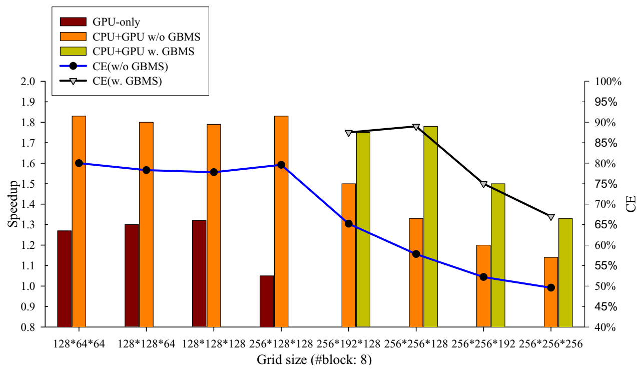

Fig. 6. Intra-node collaborative speedup and efficiency.

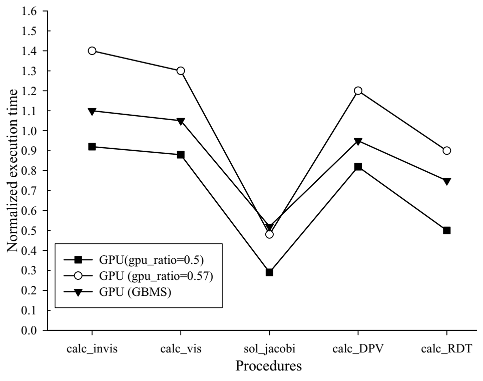

Fig. 7. Normalized execution times of different CPU and GPU procedures

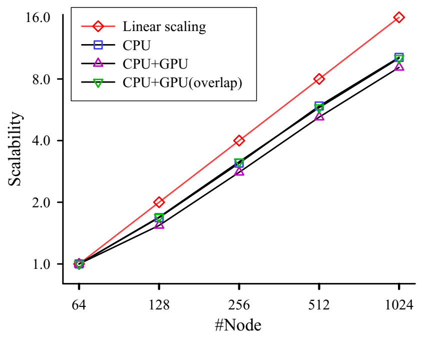

Fig. 8. Strong scalability results.

C. Performance Results

1) Node-level Performance: Fig.6 presents the intra-node speedup and *CE* with/without GBMS. The GPU-only results are only available for the four small grids, due to the device memory limit. For 256*128*128 (roughly 4M), we have to use GBMS and SP_{GPU} drops from 1.3 to 1.05. Although losing about 25% performance due to the GBMS overhead, one M2050 is still comparable to two X5670s. For those four small problems, we need not use GBMS in collaboration, and the average $SP_{CPU+GPU}$ and *CE* are 1.81 and 79% respectively. The collaborative approach has up to 45% performance advantage over the GPU-only approach. For larger problem sizes ($\geq$ 4M), GBMS plays a very important role for improving both the collaborative speedup and efficiency. Taking 256*256*128 (roughly 8M) as an example, without GBMS the GPU can only simulates 2M cells and the other 6M cells have to be simulated on the CPUs. Due to the severe load imbalance, $SP_{CPU+GPU}$ is only about 1.3 and *CE* is only about 57%. With GBMS each side simulates 4M cells, $SP_{CPU+GPU}$ and *CE* are increased to about 1.79 and 89% respectively. 8M cells is HOSTA's maximum simulation capacity on one Tianhe-1A node for a balanced collaboration, and this is an improvement of 2.3X compared to 3.5M cells without GBMS. As Fig.6 shows, further increase in problem size leads to a significant decrease in the collaborative speedup and efficiency. This is because the GPU simulation workload is still limited ($\leq$ 4M) even using GBMS. Despite of this, the GBMS balanced collaboration always has a notable advantage over the collaborative approach without GBMS.

We note that we lose around 20% speedup from ideal speedup (we get 1.8 of 2.3) in the intra-node collaboration. This is mainly due to the difference in sustainable performance between different kernels on CPU and GPU. As we mentioned, HOSTA has various complex numerical procedures consisting of many kernels, with each implemented in two versions. Those kernels may exhibit different characteristics on different devices. Some kernels run faster on the M2050, and others perform better on the X5670s. Fig.7 shows the normalized execution time of several time-consuming procedures in HOSTA, taking CPU procedure as the baseline. For an equal problem size on both devices (i.e., $gpu\_ratio = 0.5$), the difference is very significant. The GPU version `sol_jacobi` and `calc_RDT` far outperform their CPU counterparts (around 3.5X and 2X respectively); while for `calc_invis` and `calc_vis` the two versions almost have the same performance. Even the problem size is adjusted according to Eq.(11) (e.g., $gpu\_ratio = 0.57$ in Fig.7) to maximize HOSTA's performance of a whole iteration, the difference is still obvious. GBMS to some extent levels off the difference, because the GBMS overhead is carefully added to CUDA kernels (i.e., `sol_jacobi`) that run far faster than their counterpart Fortran subroutines.

The kernel characteristic difference is a practical bottleneck for efficient heterogeneous collaboration in multi-kernel applications [19], because kernels on different devices must wait for each other to finish. A possible optimization is to employ a more fine-grained (e.g., at the granularity of grid cells) partition or tailoring partition for different kernels, as described in III-B. To level off the performance difference, we may also focus on optimizing kernels with larger performance difference on different devices, or use a larger OpenMP parallel region and put the unbalanced and independent kernels together.

2) Large-scale Performance: In the following scalability tests, the 64 node results are used as the baseline. Fig.8 shows strong scalability results. The problem size of the baseline test is $256 \times 128 \times 128$ per node, and the total problem size is $64 \times 256 \times 128 \times 128$ (about 268M). Grids are evenly distributed among compute nodes. We achieve a speedup of about 10 when using 1024 nodes for CPU-only results, with an efficiency of around 64%. We use "*CPU+GPU*" to denote the col-

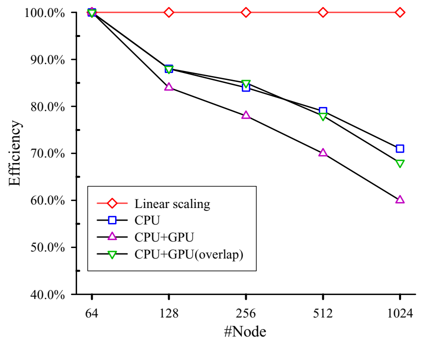

Fig. 9. Weak scalability results.

TABLE I. THE EXECUTION TIMES (IN SECOND), GFLOPS AND MACHINE EFFICIENCIES (E) FOR HOSTA WHEN SIMULATING *C919* ON TIANHE-1A.

#node	64	128	256	512
CPU	124	72	44	30
GPU	104	61	37	26
CPU+GPU(CE)	70(77%)	43(73%)	27(71%)	19(68%)
CPU GFlops(E)	780(8.7%)	1344(7.5%)	2186(6.1%)	3226(4.5%)
GPU GFlops(E)	928(2.8%)	1586(2.4%)	2580(2.0%)	3710(1.4%)
CPU+GPU GFlops(E)	1382(3.3%)	2250(2.7%)	3562(2.1%)	5094(1.5%)

laborative results without the overlapping optimization. Since the largest problem size per node (i.e., the 64 node test) is about 4M, we need not use GBMS. The collaborative speedup and efficiency decrease to about 9 and 56% respectively on 1024 nodes without the overlapping optimization. We turn on the overlapping optimization to evaluate its effectiveness. The optimized scalability results are similar to the CPU-only results. Fig.9 presents weak scalability results. The problem size per node is fixed to $256 \times 128 \times 128$. We lose about 27% efficiency from the perfect efficiency of 100% for CPU-only simulations when scaling to 1024 nodes. The efficiency loss is up to 32% (with the overlapping) and 40% (without the overlapping) for 1024 node collaboration, again demonstrating the effectiveness of the overlapping optimization.

To summarize, heterogeneous applications involve more complex and expensive data movements between devices at different levels than traditional CPU applications, which indicates that achieving a good parallel scalability and a high efficiency in large-scale systems is a tough challenge. Although the overlapping of communication and computation in HOSTA is fairly effective, we note that four data exchanges in one iteration is indeed a bottleneck in large-scale simulations. We are working on an improved implementation that can avoid some ghost/singularity communication.

Table I shows the detailed timing results of *C919*. *C919* is China's large civil airplane which currently is still under development. Since the grid contains about 150M cells, the 64 node GPU-only results are obtained with GBMS. For other node scales or collaborations, we need not use GBMS. The collaborative efficiency and the parallel efficiency are lower than that of the airfoil configuration, especially for large-number of nodes. This is because *C919* has a more complex geometry which is hard to be partitioned into grid blocks with a similar size. When distributing those blocks to large amount of Tianhe-1A nodes, and further allocating them between intra-

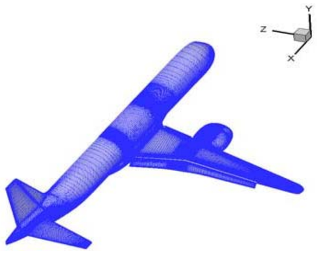

Fig. 10. The grid structure of *C919*

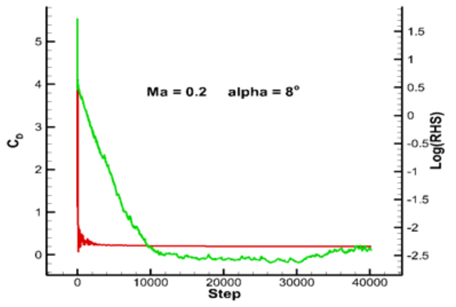

Fig. 11. The aerodynamic coefficient and the residual

node CPU and GPU, we can not achieve as good a load balance (of grid cells) as the less complex configurations such as airfoil. Nevertheless, the benefit of collaboration when simulating large grids with fairly complex geometries like *C919* is still reasonable: generally our collaborative approach can obtain the same performance, but uses only 50% less compute nodes than the CPU-/GPU-only approach on Tianhe-1A. This will significantly save the simulation cost of HOSTA.

Note that the machine efficiencies are not high: less than 10% for CPU and less than 3.0% for GPU. For HOSTA-like implicit CFD code, the flop per byte is quite low, but modern cache-based CPU architecture needs to execute large amount of operations per data item for high efficiency. This mismatch can explain the low efficiency (often less than 20% and even less than 10%) of real-world CFD applications [23]. For GPU, it is more difficult to exploit its full potential Flops and requires more tuning work to achieve a higher efficiency.

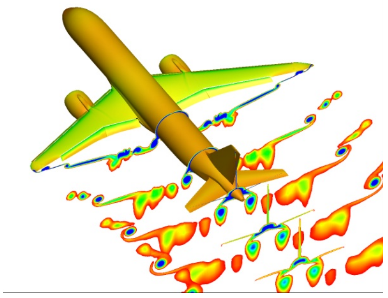

Fig. 12. The equivalent total pressure for different cross sections

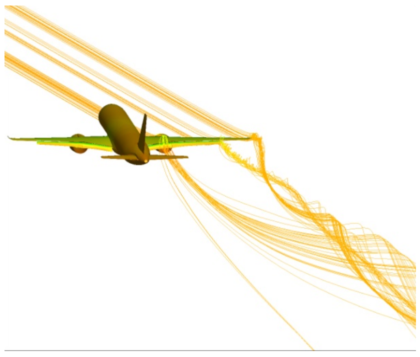

Fig. 13. Streamline of the wing tip, the side edge of wing flap and the nacelle

D. Validation

We perform high-order accurate aerodynamic simulations of *C919* to evaluate its takeoff state configuration. The body surface grid of the whole airplane is shown in Fig.10. The geometry is very complex and contains the wing, the body, the horizontal tail, the vertical fin, the pylon, the nacelle, the winglet, the leading edge slat and the trailing edge flap. The grid points near the leading and trailing wing edges, the post-wing, the horizontal and vertical tails are refined, with a $0.00001m$ first-level mesh spacing. The reference area is $120m^2$ and the distance between the reference point of torque and aircraft nose is $18.35m$. The incoming Mach number $Ma = 0.2$ and the chord length c=$4.8m$. The corresponding chord Reynolds number is 2.0×10^7 and the angle of attack is $8°$. We use the Roe flux-splitting scheme and the two-equation turbulence model in the simulation.

Fig.11 shows the convergence of the aerodynamic coefficient and the residual. The residual decreases about five order of magnitude and the drag coefficient keeps stable after 40 thousands step calculations, indicating the convergence of pressure and flow field in the simulation. In Fig.12, the nephogram for the equivalent total pressure of different cross sections is shown. Fig.13 shows the streamline of the wing tip, the side edge of wing flap and the nacelle. We clearly see the vortex at the wing tip, the slide and the trailing edge. The results indicate a very subtle numerical simulation based on the WCNS-E5 high-order scheme.

V. Related Work

There have been many studies in developing and applying high-order compact schemes in CFD since 1980s. Harten et al. [24] developed a fourth order accurate implicit finite difference scheme for shock capturing. Lele [2] developed several central compact schemes with spectral-like resolution. Recently, Lele's central compact schemes have been successfully applied by Rizzetta et al. [25] in the simulation of low speed flows. Adams et al. [26] developed a compact-ENO (Essentially Non-Oscillatory) scheme. Pirozzoli [27] developed a compact-WENO scheme. The WCNS scheme developed by us has been successfully applied in a wide range of flow simulations [7]–[10], [28]–[30]. In [31], Nonomura et al. shows the excellent freestream and vortex preservation properties of WCNS on curvilinear grids, compared with those of WENO.

Many prior work has shown the experiences of porting CFD codes of various models (e.g., Navier-Stokes equations and Lattice Boltzmann Method [32]) to GPUs, with impressive speed-ups. We only briefly review some CFD applications based on Navier-Stokes equations on multi-GPUs which are most related to this work. Brandvik et al. [33] implemented a Navier-Stokes solver for flows in turbomachines on multi-GPUs. They reported almost linear weak scaling using 16 GPUs. Ali et al. [34] parallelized an incompressible solver for turbulence with a second-order scheme and reported scalability results on 64 GPUs, but no validation is presented. Jacobsen et al. [35] parallelized a CFD solver for incompressible fluid flows. They demonstrated the large eddy simulation of a turbulent channel flow on 256 GPUs [36], but they only simulated a lid-driven cavity problem using 1D decomposition and low-order schemes. They also compare the scalability between the tri-level MPI+OpenMP+CUDA and the dual-level MPI+CUDA implementation [37], but they only use OpenMP to substitute intra-node MPI communication rather than collaborating CPU and GPU for computation as we have done in our work.

All the above studies use low-order CFD methods. Antoniou et al. [38] implemented a high-order solver on multi-GPUs for the compressible turbulence using WENO. But the solver can only run on a single node platform containing 4 GPUs for very simple domains like a 2D or 3D box. Their single-precision implementation achieve a speedup of 53 when comparing 4 Tesla C1070s with a single core of an Xeon X5450. Castonguay et al. [39] parallelized the first high-order, compressible viscous flow solver for mixed unstructured grids with MPI and CUDA, where the Vincent-Castonguay-Jameson-Huynh method is used. A flow over SD7003 airfoil and a flow over sphere is simulated using 32 GPUs. Zaspel et al. [40] implemented an incompressible double-precision two-phase solver on GPU clusters using a fifth-order WENO scheme. The test problem is a rising bubble of air inside a tank of water with surface tension effects, and parallel performance results are reported using up to 48 GPUs. In [16], we parallelize HOSTA on Tianhe-1A using MPI and CUDA, but no CPU-GPU collaboration as well as heterogeneous load balance were implemented. This work is a major extension of [16]. To summarize, the above GPU-enabled CFD simulations using high-order schemes are still preliminary as far as grid complexity, problem size and parallel scale are comprehensively concerned.

VI. Conclusion and Future Work

The employment of high-order CFD schemes is an effective approach to obtain high-resolution and high fidelity simulation results for complex flow problems. This work presented some novel techniques to achieve balanced and scalable high-order CPU-GPU collaborative simulations on the GPU-accelerated supercomputer Tianhe-1A. The approach and implementation of CPU-GPU collaboration as well as balancing scheme provide a fairly general experience of similar porting and optimizing efforts for multi-block CFD applications. We validate our work using China's large airplane configuration *C919* and reports the first CPU-GPU collaborative high-order accurate aerodynamic simulation result with a complex grid geometry.

For future work, besides fine tuning of the aforementioned bottlenecks in the collaborative code, we are planning to port HOSTA to China's new supercomputer Tianhe-2, which was ranked No.1 in Jun, 2013 for large-scale simulations such as direct numerical simulations of turbulence. Due to the usage of traditional programming models, we expect that the work would take us relatively less time. Since porting and optimizing CFD code often involves some common programming efforts such as restructuring kernel skeletons and data structures, in the long run we will encapsulate our experience learned in this paper and further work into an application programming and auto-tuning infrastructure for multi-block CFD simulations on heterogeneous systems.

VII. Acknowledgments

This paper was supported by the National Science Foundation of China under Grant No.11272352 and No.61379056, and the Open Research Program of China State Key Laboratory of Aerodynamics under Grant No.SKLA20130105.

References

[1] W. Z.J., "High-order methods for the euler and navier-stokes equations on unstructured grids," *Prog. Aerosp. Sci.*, vol. 43, pp. 1–41, 2007.

[2] L. S.K., "Compact finite difference schemes with spectral-like resolution," *J. Comput. Phys.*, vol. 103, pp. 16–42, November 1992.

[3] G. Jiang and C. Shu, "Efficient implementation of weighted eno schemes," *J. Comput. Phys.*, vol. 126, pp. 202–228, 1996.

[4] X. Deng and H. Zhang, "Developing high-order weighted compact nonlinear schemes," *J. Comput. Phys.*, vol. 165, no. 1, pp. 22–44, 2000.

[5] E. JA, "High-order accurate, low numerical diffusion methods for aerodynamics," *Prog. Aerosp. Sci.*, vol. 41, pp. 192–300, 2005.

[6] T. J.G. and T. K.R., *Numerical Solution of the One-dimensional Hydrodynamic Equations in an Arbitrary Time-dependent Coordinate System*. Technical Report UCLR-6522: University of California Lawrence Radiation Laboratory, 1961.

[7] X. Deng, M. Mao, G. Tu, H. Liu, and H. Zhang, "Geometric conservation law and applications to high-order finite difference schemes with stationary grids," *J. Comput. Phys.*, vol. 230, pp. 1100–1115, 2011.

[8] X. Deng, M. Mao, G. Tu, and et al., "Extending the fifth-order weighted compact nonlinear scheme to complex grids with characteristic-based interface conditions," *AIAA J.*, vol. 48, no. 12, pp. 2840–2851, 2010.

[9] F. K., N. T., and T. S., "Toward accurate simulation and analysis of strong acoustic wave phenomena-a rview fom the eperience of our study on rocket problems," *Int. J. Numer. Meth. Fluids*, vol. 64, pp. 1412–1432, 2010.

[10] X. Deng, M. Mao, G. Tu, and et al., "High-order and high accurate cfd methods and their applications for complex grid problems," *Commun. Comput. Phys.*, vol. 11, no. 4, pp. 1081–1102, 2012.

[11] GPGPU, "General-purpose computation on graphics hardware," 2013, http://gpgpu.org/.

[12] Intel, "Many integrated core (mic) architecture," 2013, http://www.intel.com/content/www/us/en/architecture-and-technology/many-integrated-core/intel-many-integrated-core-architecture.html.

[13] "Mont-blanc project home page," 2013, http://www.montblanc-project.eu.

[14] N. Corporation, "Cuda programming model," 2013, https://developer.nvidia.com/cuda-toolkit.

[15] X. Yang, X. Liao, K. Lu, and et al., "The tianhe-1a supercomputer: Its hardware and software," *J. Comput. Sci. Tech.*, vol. 26, no. 3, pp. 344–351, 2011.

[16] C. Xu, L. Zhang, X. Deng, Y. Jiang, W. C. J. Fang, Y. Che, Y. Wang, and W. Liu, "Parallelizing a high-order cfd software for 3d, multi-block, structural grids on the tianhe-1a supercomputer," in *International Supercomputing Conference*, 2013.

[17] A. Ytterstrom, "A tool for partitioning structured multiblock meshes for parallel computational mechanics," *Int. J. High Perform. Comput. Appl.*, vol. 11, pp. 336–343, 1997.

[18] "Top500 cite," 2013, http://top500.org/.

[19] K. Kofler, I. Grasso, B. Cosenza, and T. Fahringer, "An automatic input-sensitive approach for heterogeneous task partitioning," in *ICS*, 2013.

[20] E. A. et al., "Improving application behavior on heterogeneous many-core systems through kernel mapping," *Parallel Comput.*, 2013.

[21] "Khronos opencl working group," 2013, http://www.khronos.org/opencl.

[22] Y. Wang, L. Zhang, Y. Che, W. Liu, C. Xu, and H. Liu, "Th-meshsplit: A multi-block grid repartitioning tool for parallel cfd applications on heterogeneous cpu/gpu supercomputer," in *ParCFD*, 2012.

[23] W. D. Gropp, D. K. Kaushik, D. E. Keyes, and B. F. Smithdx, "Towards realistic performance bounds for implicit cfd codes," in *ParCFD*, 2000.

[24] A. Harten and H. Tal-Ezer, "On a fourth order accurate implicit finite difference scheme for hyperbolic conservation laws, ii. five-point schemes," *J. Comput. Phys.*, vol. 41, pp. 329–356, 1981.

[25] R. D., V. M., and M. P., "A high-order compact finite difference scheme for large-eddy simulation of active flow control," *Prog. Aerosp. Sci.*, vol. 44, pp. 397–426, 2008.

[26] A. N.A. and S. K., "A high-resolution hybrid compact-eno scheme for shock-turbulence interaction problems," *J. Comput. Phys.*, vol. 127, 1996.

[27] P. S., "Conservative hybrid compact-weno schemes for shock-turbulence interaction," *J. Comput. Phys.*, vol. 179, pp. 81–117, 2002.

[28] X. Deng, "High-order accurate dissipative weighted compact nonlinear schemes," *Science in China (Serial A)*, vol. 45, pp. 356–370, 2002.

[29] X. Deng, X. Liu, and M. Mao, "Advances in high-order accurate weighted compact nonlinear schemes," *Adv.Mech.*, vol. 37, pp. 417–427, 2007.

[30] X. Liu, X. Deng, and M. Mao, "High-order behaviors of weighted compact fifth-order nonlinear schemes," *AIAA Journal*, vol. 45, pp. 2093–2097, 2007.

[31] e. a. T. Nonomura, "Free-stream and vortex preservation properties of high-order weno and wcns on curvilinear grids," *Comput. Fluids*, vol. 39, pp. 197–214, 2010.

[32] T. Aoki, "A peta-scale les (large-eddy simulation) for turbulent flows based on lattice boltzmann method," in *GPU Technology Conference 2012*, 2012.

[33] T. Brandvik and G. Pullan, "An accelerated 3d navier-stokes solver for flows in turbomachines," in *ASME Turbo Expo 2009: Power for Land, Sea and Air*, 2009.

[34] A. Khajeh-Saeed and J. B. Perot, "Computational fluid dynamics simulations using many graphics processors," *Comput. Sci. Eng.*, 2012.

[35] D. A. Jacobsen, J. C. Thibault, and I. Senocak, "An mpi-cuda implementation for massively parallel incompressible flow computations on multi-gpu clusters," *AIAA paper2010-0522*, 2010.

[36] R. DeLeon, D. Jacobsen, and I. Senocak, "Large-eddy simulations of turbulent incompressible flows on gpu clusters," *Comput. Sci. Eng.*, vol. 15, pp. 26–33, 2013.

[37] D. A. Jacobsen. and I. Senocak, "Scalability of incompressible flow computations on multi-gpu clusters using dual-level and tri-level parallelism," *AIAA paper 2011-947*, 2011.

[38] A. S. Antoniou, K. I. Karantasis, and E. D. Polychronopoulos, "Acceleration of a finite-difference weno scheme for large-scale simulations on many-core architectures," *AIAA paper 2010-0525*, 2010.

[39] P. Castonguay, D. M.Williams, P. E. Vincent, M. Lopez, and A. Jameson, "On the development of a high-order, multi-gpu enabled, compressible viscous flow solver for mixed unstructured grids," *AIAA paper 2011-3229*, 2011.

[40] Z. P. and G. M., "Solving incompressible two-phase flows on multi-gpu clusters," *Comput. Fluids*, 2012.