2010 16th International Conference on Parallel and Distributed Systems

# Acceleration of a High Order Accurate Method for Compressible Flows on SDSM based GPU Clusters

Konstantinos I. Karantasis and Eleftherios D. Polychronopoulos
High Performance Infomation Systems Lab
School of Computer Engineering and Informatics
University of Patras
Rio, Greece 26500
Email: {kik, edp}@hpclab.ceid.upatras.gr

John A. Ekaterinaris
School of Mechanical and Aerospace Engineering
University of Patras
Rio, Greece 26500
and FORTH/IACM Heraclion, Greece, 71110
Email: ekaterin@iacm.forth.gr

***Abstract*—The recent advent of multicore processors, and especially the introduction of many-core GPUs, opens new horizons to large-scale, high-resolution, simulations for a broad range of scientific fields. Among them, the scientific area of CFD appears to be one of the candidates that could significantly benefit from the utilization of many-core GPUs. In order to investigate such a potential, we evaluate the performance of a high-order accurate method for the simulation of compressible flows. Current implementation is taking place on a GPU cluster. Nevertheless, a novel approach is followed concerning the utilization of GPU clusters that does not involve explicit message passing. Instead, the presented implementation resides on Software Distributed Shared Memory (SDSM) to propagate changes across the simulation phases. The first results prove to be emboldening and lay grounds for further research along the use of shared memory abstraction in order to utilize future GPU clusters.**

## I. Introduction

During the last decade, we have experienced a major shift on microprocessor design technology[1]. For the purpose of producing next generation microprocessors, that could exhibit respectable performance gains, preserving at the same time an acceptable rate of power consumption, in comparison to high-frequency processors of that time, the decision was to unfold the potential of parallelism utilization inside the chip. Along that process, that is often described as the multicore revolution[2], modern GPUs have - up to now - the lead concerning the number of cores that they deploy.

Due to their simplistic design that excludes several features and most notably memory coherence, out of order execution, and branch prediction, modern GPUs are able to encompass hundreds of cores in a single chip, while at the same time, the number of cores in general purpose CPUs reaches a few dozens at experimental level[3]. For instance, the NVIDIA Tesla GPU[4] consists of 30 multiprocessors, where each multiprocessor contains 8 cores, resulting on an aggregate number of 240 cores inside a single GPU.

Nevertheless, since the introduction of multi-core and many-core architectures, the burden of the utilization of the afforded resources has been transferred mostly to the software stack. In the case of parallel processing with GPUs, synergetic execution schemes have to be implemented between CPU and GPU threads, in order to benefit from the afforded computing power of many-core GPUs. That process has been facilitated by the introduction of simple programming environments, such as CUDA[5] and OpenCL[6], that rely on C/C++ programming languages and their respective multithreaded libraries. However, until now, there is no availability of well-established, sophisticated programming environments and optimization tools. Under these circumstances, the exploration of the performance capabilities of the newly introduced multiprocessors, has been relied mostly on the implementation, porting and hand-optimization of a wide range of applications that could benefit from the proliferation of computational resources. It is expected that such a thorough study at the application level will drive the development of high performance compilers, runtime systems and respective middleware.

In the current paper, following the same practice, we present the implementation of a computationally intensive high order accurate method for the simulation of compressible flows in order to study the potential acceleration of that application category. In parallel we attempt to gain important insight concerning attainable programming models that target GPU based supercomputers.

Concerning the application part, it is well known that numerical simulation of turbulence in high-speed flows is daunting. This is due to the difficulty of ensuring high-resolution and fidelity in capturing small disturbances in an environment containing sharp gradients associated with shocks and relatively thin boundary layers. Even with the use of higher-order approaches many shock capturing methods introduce spurious (numerical) noise, which contaminates the solution beyond acceptable limits. Such distortions can lead to significant damping of turbulence fluctuations and may mask the effects of the subgrid-scale (SGS) models. Specification of boundary conditions ensuring that the numerical discretizations remain stable is also a critical issue. Robust, high-fidelity and accuracy methodologies that are capable of treating complex flows and are applicable for high-resolution numerical solutions in complex domains are therefore solemnly required.

As far as the presented implementation platform is concerned, clusters are expected to remain the main structure in

This work is supported by the Karatheodori Grant no. C-141 of the University of Patras

1521-9097/10 $26.00 © 2010 IEEE
DOI 10.1109/ICPADS.2010.107

the organization of supercomputers in the near future. The advent of many-cores is not expected to replace clusters. On the contrary, it has already began to enhance their architecture, and the result of such a transition is evident by the deployment of the first heterogeneous accelerator clusters. Following that anticipation, in the current paper we present an alternative approach in order to program high performance GPU clusters. Instead to the common practice, that employes MPI or a related message passing programming model, we are presenting an implementation and first results of the deployment of a well-established SDSM in order to accelerate the execution of the numerical simulation under study.

The rest of the paper is organized as follows. In section II we refer to the research efforts that relate to the presented implementation. Section III describes the numerical implementation and provides the basis for the described simulation process. Section IV describes in detail the implementation effort in the context of the GPU cluster using CUDA and Intel Cluster OpenMP. In Section V we provide the performance evaluation of the proposed schemem and finally in Section VI we draw our conclusions and refer to our future work.

## II. Related Work

Since the introduction of modern GPUs and their respective development tools, CUDA and OpenCL, several scientific applications and simulation software have been ported on these many-core architectures[7]. While most of the efforts concentrate on the context of a single GPU and study its interaction with the host CPU, there is an important part of research that has focused on the utilization of multiple GPUs for parallel simulations. In cases where two or more GPUs reside on the same board, usually a shared memory programming model is employed, such as OpenMP or Pthreads[8][9]. When a GPU cluster is available[10], then, to our knowledge, most efforts up to now have been using some kind of message passing protocol, mostly MPI, in order to distribute cooperating processes across the cluster[11][12][13].

The most active area of high performance middleware that pose an alternative to explicit message passing, is the area of Partitioned Global Address Space (PGAS) languages. However, the research efforts that aim at providing a programming environment that could utilize GPU clusters through a PGAS language, are still at a preliminary stage. Concerning DSM implementation at a library level, the work of Gelado et al.[14] is influenced by the principles of shared memory abstraction, however ADSM implements a distributed shared memory layer between host and device memories, which, at the moment, is not meant to utilize distinct GPU resources on a cluster. Strengert et al.[15] have also presented CUDASA, which provides a language extension to CUDA. CUDASA is supported by a source-to-source compiler, and in order to realize inter-node communication on a cluster it uses MPI calls.

In the context of CFD, there is an ongoing research effort concerning high performance simulations that operate in both structured[16][17][18][19] and unstructured meshes[20][21]. Cohen et al.[18] have performed 3D simulations on structured meshes using double precision computations. The method that they apply in order to enforce 3D domain decomposition of the structured mesh is similar to the approach that we follow in the current paper. Klöckner et al.[21] achieve an acceleration on nodal discontinuous Galerkin methods operating on unstructured meshes using a single GPU and report a speedup of a factor of 10 in comparison to CPU execution. However, all the CFD simulations that utilize GPU clusters are using, until now, exclusively MPI.

## III. Numerical Implementation

In this section the governing equations and the numerical scheme used in parallel version of the code are summarized.

### A. Governing equations

The Navier-Stokes equations are solved numerically. The Cartesian coordinate form of the continuity momentum and energy equations is:

$$\frac{\partial \rho}{\partial t} + \frac{\partial (\rho\, u_j)}{\partial x_j} = 0 \tag{1}$$

$$\frac{\partial (\rho\, u_i)}{\partial t} + \frac{\partial (\rho\, u_i\, u_j)}{\partial x_j} = -\frac{\partial p}{\partial x_i} + \frac{\partial \tau_{i\,j}}{\partial x_j} \tag{2}$$

$$\frac{\partial (\rho\, E)}{\partial t} + \frac{\partial (\rho\, E + p) u_j}{\partial x_j} = -\frac{\partial}{\partial x_j}\left(c_p \frac{\mu}{\mathrm{Pr}} \frac{\partial T}{\partial x_j}\right) + \frac{\partial}{\partial x_j}\left[u_i \tau_{i\,j}\right] \tag{3}$$

where $E = p/(\gamma - 1) + 0.5 \rho u_i u_i$ is the energy, $T$ is the temperature, $Pr$ is the Prandtl number, and the viscous stress tensor $\tau_{i\,j}$ is given by

$$\tau_{i\,j} = \mu \left(2 S_{i\,j} - \frac{2}{3} S_{mm} \delta_{i\,j}\right) \tag{4}$$

with $S_{i\,j} = (\partial u_i / \partial x_j + \partial u_j / \partial x_i)/2$. The molecular viscosity is obtained from the computed temperature using Sutherland's law

$$\mu(T) = T^{\frac{3}{2}} \left[\frac{1 + 0.76}{T + 0.76}\right] \tag{5}$$

### B. Numerical scheme

The Favre-averaged Navier-Stokes equations were written in generalized coordinates $\xi = \xi(x, y, z)$ etc. The discretization of the inviscid fluxes is based on standard finite difference WENO scheme[22][23][24]. The numerical code includes options for 5th, 7th and 9th order accurate discretizations of the inviscid fluxes. The viscous fluxes were evaluated with a 4th order accurate explicit, central difference scheme by evaluating the second derivatives with repeated evaluation of the first derivative, first at half points and then at the nodes of the finite difference mesh. High order accurate discretization of the viscous fluxes requires large computational time. It was found that for a fourth order accurate explicit discretization, significant portion of computational time per time step is spend for the evaluation of the viscous fluxes and that higher order explicit or compact discretizations are prohibitively

expensive computationally. The classical tree stage third order accurate TVD Runge-Kutta method of Osher and Shu[25] is used for time marching. For Large Eddy Simulation (LES) computations the Smagorinsky subgrid scale model is used.

The essential details of the finite difference WENO discretization for an equally spaced mesh are given next. The reconstruction by WENO uses a convex combination of $k$ candidate ENO stencils, $S_q(j) = \{x_{j-q},\ ..., x_{j-q+k-1}\},\ q = 0, ..., k-1$. Each ENO stencil $S_q(j)$ produces a *k-th* order accurate ENO reconstruction $f^{(q)}_{j+1/2} = \sum_{m=0}^{k-1} c_{q\,j}\, \bar{f}_{j-q+m}$, where for the finite difference formulation is the nodal value $\bar{f}_{j-q+m}$. The convex sum of the WENO reconstruction

$$\tilde{f}_{j+1/2} = \sum_{q=0}^{k-1} \omega_q\, f^{(q)}_{j+1/2} = f_{j+1/2} + O(\Delta x^{2k-1}), \quad (6)$$

$$\omega_q \geq 0, \quad \textstyle\sum_{q=0}^{k-1} \omega_q = 1$$

produces a $(2k-1)^{\text{th}}$ order approximation at the cell boundaries of an equally space mesh, which is the transformed domain mesh. The nonlinear weights $\omega_q$ in Eq. (6) are given by:

$$\omega_q = \frac{\alpha_q}{\sum_{m=0}^{k-1} \alpha_m},\ \ \alpha_q = \frac{d_q}{(\varepsilon+\beta_q)^2}\ \ ,\ \ \sum_{q=0}^{k-1} d_q = 1 \quad (7)$$

where $d_q$ are positive constants (optimal weights) for the smooth stencils,[26][27][28] $\beta_q$ are the smoothness indicators of the stencil $S_q(j)$, and $\varepsilon > 0$ is a small constant to avoid division by zero. The value of this constant was taken $\varepsilon = 10^{-6}$ for all computations. The smoothness indicators $\beta_q$ were computed as sum of the squares of $L_2$ norms of the derivatives of the interpolation polynomial[22].

Smoothness measures for *k* = 3, 4, 5 or $(2k-1) - th$, 5th, 7th, and 9th order accurate WENO reconstructions are given in [24]. The reconstruction of the right state about *j+1/2* is symmetric.

High order finite difference WENO discretization of the inviscid fluxes clearly requires wide stencils. For example, the 5th order accurate WENO scheme requires a seven point wide stencil, while the 9th order accurate one involves an eleven point wide stencil. These wide stencils may present problems for the specification of boundary conditions and make less efficient the parallelization with domain decomposition techniques. In this work, boundary conditions were specified using the ghost cell approach for all boundaries. Furthermore, the simplest Lax-Friedrichs splitting $f^{\pm}(u) = 0.5[f(u) \pm a\,u],\ \ a = \max_u |\, f'(u)|$ is used for the evaluation of the numerical flux.

For the numerical solution of the three dimensional Euler or Navier-Stokes system with the finite difference WENO scheme it is necessary to define an average state of the left, $f_L$ and the right $f_R$ states at an interface. A Roe-type approximate Riemann solver is used. The Lax-Friedrichs numerical flux is used and the average state $\bar{U}$ is defined for the primitive variables $U = [\rho,\ u,\ \ v,\ \ w,\ \ p]^T$ using Roe's average.

Once the average state has been defined the conservative flux vectors are first projected on the characteristic space where the reconstruction is carried out and then the numerical flux is re-projected back to the conservative variables space and the derivatives are computed as

$$\frac{\partial f}{\partial x} = \frac{\hat{f}_{i+1/2} - \hat{f}_{i-1/2}}{\Delta x} \ \text{ or } \ \frac{\partial f}{\partial \xi} = \frac{\tilde{f}_{j+1/2} - \tilde{f}_{j-1/2}}{\Delta \xi} \quad (8)$$

for Cartesian or generalized coordinates, respectively. In Eq. (8), $\tilde{f}_{j+1/2}$ denotes the numerical flux which is reconstructed using high order accuracy. Therefore the evaluation of a proper set of right and left eigenvectors for the conservative flux vectors is a crucial step of the WENO solver.

A curvilinear coordinates transformation $(x, y, z) \rightarrow (\xi, \eta, \zeta)$ was applied and finite difference discretizations were applied in the equally space transformed domain. The metric quintiles were computed with the standard compact fourth order finite difference formulas. It was found that due to the simplicity of the mesh evaluation of metrics with sixth order accurate formulas does not make any difference.

## IV. Parallel Simulation

The presented application belongs to the broad category of scientific GPU codes that have been based on a former multithreaded or distributed version of the algorithm in order to achieve their acceleration on GPUs using software development environments such as CUDA and OpenCL. Particularly, the presented multi-GPU implementation is based on a multithreaded OpenMP version that is used to perform data parallel simulations on multicore CPUs[9]. Next, we describe the most important implementation challenges of the acceleration process.

### A. Domain decomposition

The presented method operates on structured meshes, which are particularly suited for domain decomposition. Nevertheless, the dimensions on which decomposition is applied differ between OpenMP-only version and multi-GPU cluster version. In the case of multithreaded execution, where solely OpenMP is used, the computational domain is fragmented in a band based fashion along the axial streamwise direction as depicted schematically in Fig. 1. The streamwise direction contains usually the largest number of points in most CFD simulations. However, without loss of generality, the directions can always be interchanged so that the largest number of points in the simulation is along the axial ($i$ or $\xi$) direction. The generalized coordinates form of the governing equations is solved and the global computation of dimension *Imax* x *Jmax* x *Kmax* is subdivided along the *i,* or $\xi$ direction, which is often the streamwise direction, in N subdomains of dimension ({*Imax/N*+2*m*} x *Jmax* x *Kmax*) where *m* is the number of the ghost points required for data transfer. The number of ghost points varies with the order of the base scheme and for the 5th order WENO is *m=3*, while for the 9th order WENO is *m=5.*

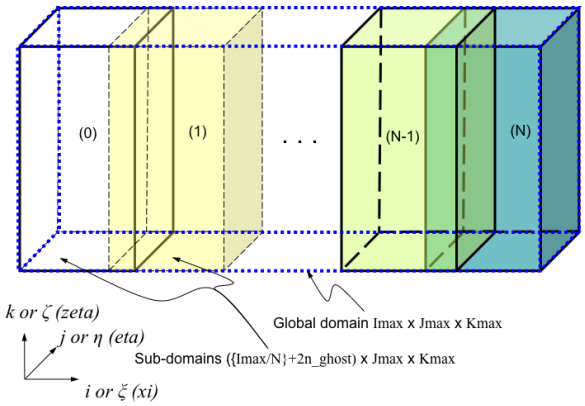


Fig. 1. Domain composition on the host side (CPU/Cluster level)

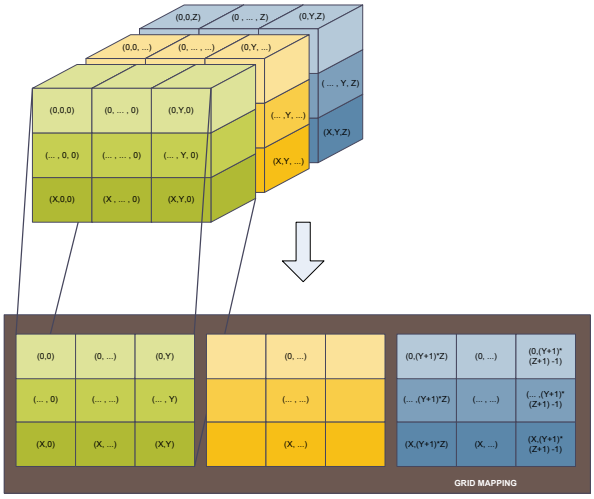


Fig. 2. Domain decomposition on the GPU device side and mapping of 3D blocks on a 2D grid

### B. Implementation on GPU Clusters

In order to pose a cooperative execution of the parallel simulation on a GPU cluster, a cluster enabled OpenMP implementation that is based on SDSM[29] is used as the middleware platform. In that way, the presented implementation follows a different approach than most practices that have been implemented up to now and involve the spawning of MPI communicating processes on every node of the cluster.

At a cluster wide level, on every OpenMP thread that is created we assign the control of a particular GPU device. Currently only a "1 to 1" assignment between OpenMP threads and GPU devices is supported. Rather than expressing explicitly the data communication between threads on distinct cluster nodes with messages, the changes are propagated transparently through the memory consistency mechanisms of the SDSM layer. These mechanisms are triggered on specific moments that involve the use of synchronization constructs. In our case, the execution pattern enforces solely the use of barrier synchronization.

In order to efficiently utilize multiple GPUs on a cluster, the band based domain decomposition that was described is not adequate. The domain has to be fragmented in a way that the resultant number of computational portions will be large enough to allow overlapping of thread blocks and efficient utilization of stream multiprocessors of every GPU. Therefore, a two-level hierarchical approach is followed to achieve fine-grain domain decomposition. At the top level a band based decomposition is also applied, with every band of the domain being assigned on each OpenMP thread. Subsequently, that band is imposed on a 3D decomposition that results on the formation of the computational thread-blocks that will be scheduled on each GPU. The implementation of that 3D scheme follows the practice that is described by Cohen et al. in [18]. Every data point of the mesh is represented by an aligned structure of floating-point, single precision, values and its manipulation is assigned on a single GPU thread. In order to achieve proper mapping of the 3D thread blocks on a 2D grid that is supplied on every kernel launch, the respective subdomain is decomposed on a two dimensional grid with dimensions $(\lceil I\_Size/BlockSize\_x \rceil, \lceil J\_Size/BlockSize\_y * Z\_Size/BlockSize\_z \rceil)$. A schematic representation of the applied scheme is shown in Fig. 2.

The above decomposition scheme leads to a quite coalescent data access pattern of the mutable data structures that correspond to the structured mesh. These data structures are placed in the global memory of the GPU and their respective parts that need to be communicated between multiple GPUs are placed at the sharable data region of the SDSM. On every simulation step these boundary points are transferred between host memory and device global memory with direct memory copies and among cluster nodes with propagations that correspond to barrier synchronization points. As far as immutable data in the GPU is concerned, plain variables are placed in constant memory and 3D read-only data structures that have been produced during initialization are placed in texture memory. In that way we are able to benefit from the caching mechanisms of texture memory which are optimized for spatial locality and less coalescent accesses.

Concerning the execution flow of the simulation (Fig. 3), all the computations that are required by the Runge-Kutta time stepping are taking place in the GPU. On every iteration, there are 5 distinct kernels that are launched and realize the simulation step, as it is depicted in Fig. 4. The kernels that discretize the right hand side part of the equations operate on the entire extent of the subdomain, while kernels *BC* and *UPDATE* perform updates that do not exhibit a high degree of parallelism. Still, their execution on the GPU side is more efficient than a potential execution on the CPU if we take into account the necessary memory copies that should take place.

Lastly, the necessary synchronization that is imposed by the computation scheme, it is restricted on barrier synchronization between the threads of the same common block inside each

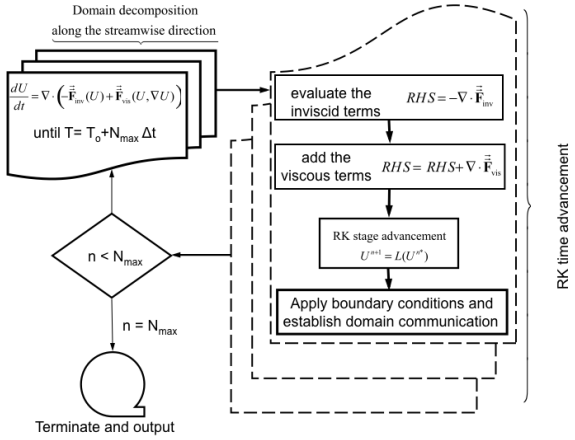


Fig. 3. Overall execution flow of the simulation

```
void RungeKutta() {
  for(int m = 0; m < ORDER; m++) {
    if(omp_get_num_threads() > 1){
      copyData(FROM_DEVICE, DIRECTION); //
      #pragma omp barrier
      copyData(TO_DEVICE, DIRECTION);
    }
    launchKernel(BC);          // Boundary Conditions Upd
    launchKernel(XI);          // Right hand side along I
    launchKernel(ETA);         // Right hand side along J
    launchKernel(ZETA);        // Right hand side along K
    launchKernel(UPDATE, m); // Update muttable data
  }
  copyData(FROM_DEVICE, DIRECTION);
}
```

Fig. 4. Execution flow of the Runge-Kutta time advancement in the GPU

CUDA kernel. At these points, barrier synchronization assures that the required updates of auxiliary local variables, such as Roe's averaging or right and left eigenvectors that are evaluated at this average state, have been accomplished. The need for mutual exclusion is minimized on the use of an atomic max operation that is used in order to compute the maximum eigenvalue required for the construction of the Lax-Friedrischs numerical flux.

## V. Benchmark Evaluation

In this section we describe the simulation settings and the experimental platform that were used for the validation of the parallel simulation on high performance GPU clusters. Next the performance is evaluated for the execution models that have been considered.

### A. Simulation test cases

The evaluation of the presented high order accurate method on GPU clusters refers to the simulation of Rayleigh-Taylor (R-T) instability[30][31]. The code is three dimensional and the computational domain for the simulations of the R-T instability is the box $(1 \times 0.25 \times 0.25)$ in three dimensions. The "heavy" fluid is on the left side and has density $\rho_L = 2$ while the "light" fluid on the right has density $\rho_R = 1$. The interface between the two fluids is at $x = 1/2$ and the variation

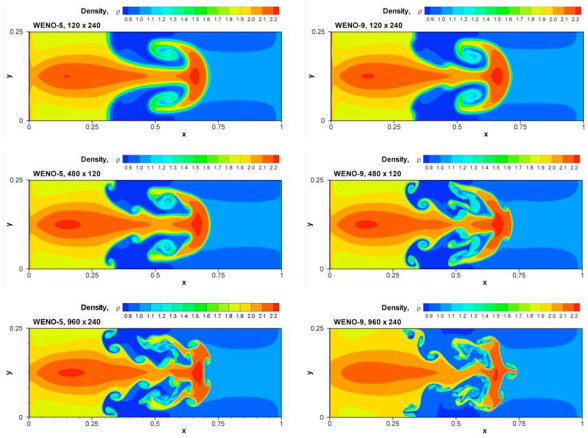


Fig. 5. Effect of the order of accuracy (WENO-5 versus WENO-9) and grid spacing on the development of Rayleigh-Taylor instability

of the initial pressure is linear throughout the domain. The initial pressure in the domain of the heavy fluid on the left is $p_L(x) = 1 + 2x$, while the variation of the pressure on the right is $p_R(x) = 1.5 + x$.

The initial perturbation of axial velocity $u(y) = -0.025\cos(8\pi y)$ is specified throughout the computational domain and the source term, $S = (0,\ \rho,\ 0,\ 0,\ \rho u)^T$ [30][31], is added for both viscous and inviscid simulations. The significant reductions of the computational time achieved through the use of GPUs, which are discussed in detail in the next section, made possible simulations of the R-T instability with different mesh densities and schemes of different order of accuracy. A comparison of different simulations obtained on a series of meshes is shown in Fig. 5. All simulations of Fig. 5 are carried out with the 3D code by enforcing periodic boundary conditions along the third dimension for the same final time. Clearly an increase on the order of accuracy yields the same effect as the doubling of the mesh density in both directions. Furthermore, the use of GPU cluster made possible three dimensional simulations. A three dimensional simulation obtained on a relatively coarse 240 x 60 x 60 point mesh showed similar structures with the corresponding 2D simulation of Fig. 5.

The R-T instability for inviscid flow develops a large number of small scales with the increase of resolution, either by the order of the scheme, or by the reduction of the mesh size. A 3D inviscid simulation with the WENO-9 scheme and a 480x120x120 mesh also closely resembled the corresponding 2D results of Fig. 5. A 3D viscous flow computation of the R-T instability for $R_e = 10^4$ is shown in Fig. 6. Two dimensional numerical experiments of viscous R-T instability (not shown here) have demonstrated that a mesh size of $h = 1/480$ yielded sufficient resolution since the computed results with meshes $h_1 = 1/480$ and $h_2 = 1/960$ are identical. The detail of the computed 3D flow field of the 3D R-T instability is also shown in Fig. 6.

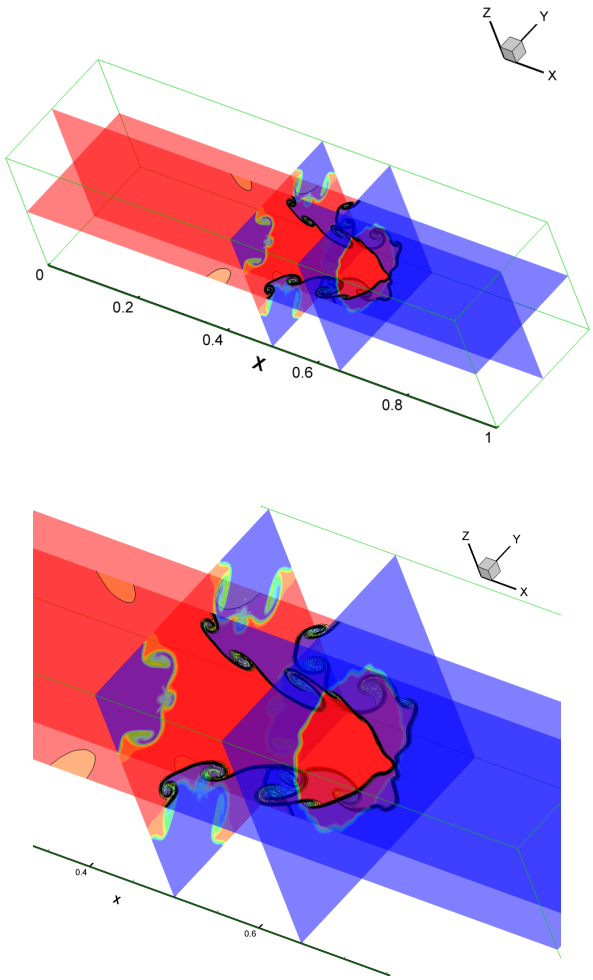


Fig. 6. Development of 3D Rayleigh-Taylor instability using WENO-9 scheme with a 480x120x120 mesh for viscous flow

### B. Experimental Platform

The experimental evaluation of the current implementation took place on a 4-node GPU cluster. The 4 nodes of the cluster were externally connected with 2 NVIDIA Tesla 1U S1070 computing blades, establishing one connection per node, that supplied each node with 2 Tesla C1060 graphics processors. Thus, each node was able to utilize 60 stream multiprocessors and a total number of 480 cores at the GPU side. In aggregate this specific configuration resulted on a GPU cluster with 240 stream multiprocessors and 1920 cores.

In terms of software middleware platforms, the Intel Cluster OpenMP[29] implementation was used for the realization of a SDSM that provided transparent shared memory abstraction over the cluster nodes. The specific implementation is supplied with the Intel C/C++ Compiler (*icc*) and has been based on the extension of the highly efficient Treadmarks DSM[32]. Accordingly, the NVIDIA CUDA software development environment was used to utilize the many-core GPUs per node.

Nevertheless, the use of the CUDA runtime environment was not possible, because the Intel C/C++ compiler is not fully supported at the moment by the NVCC CUDA compiler driver. Therefore, our implementation has been exclusively taken place at the low-level CUDA Driver API. This specific co-operation is feasible in the case where the source code that is destined to run on the host is compiled with *icc* and the source code of the CUDA kernels that will operate on the GPU device side is compiled with *nvcc*. Currently the only restriction, under that co-operation scheme, is that no direct memory copy (*memcpy*) operation can take part between GPU and a memory area that is sharable cluster-wide.

TABLE I
EXPERIMENTAL ENVIRONMENT AND SETTINGS

| | **CPU** | **GPU** |
|---|---|---|
| #Cluster nodes | 4 | |
| #units per node | 1 | 2 |
| #cores per node | 4 | 480 (60 SM) |
| Type | Intel®Xeon® E5504 @ 2.00GHz | Tesla C1060 @ 1.30 GHz S1070 1U system |
| Memory | 4096KB (Cache) 4 GB (Host Memory) | 4GB (Device Global) |
| Interconnect | Gigabit Ethernet via Gigabit Switch | PCI-E x16 |
| SDSM | Intel Cluster OpenMP (ICC 11.1) | |
| CUDA SDK | CUDA Driver API 2.2 | |

### C. Performance evaluation on GPU clusters

In order to evaluate the performance of our implementation we have conducted experiments using various mesh sizes that present equivalent results in terms of performance scaling. In the following graphs we present the results that concern executions on a 480x120x120 mesh. The execution times refer to the average execution time of 4 simulations, with each simulation performing 8000 iterations.

Four basic configurations are compared in terms of execution time (Table II and Fig. 7) and their respective speedup (Fig. 8). *OpenMP* refers to the multithreaded execution of the simulation that takes place completely on the CPU without involving GPU processing. *CUDA-LOCAL* refers to the local multi-GPU evaluation on a single node of the cluster. *CUDA-SDSM* refers to a cluster wide evaluation over SDSM, where on each node a separate process is spawned to drive the execution on a specific GPU. In that sense a single Cluster OpenMP thread corresponds to a single process yielding a "1 to 1" relation between OpenMP threads and local processes. Lastly the *CUDA-SDSM-MT* corresponds to an SDSM evaluation on top of the cluster, where a single process is started per node, and in the case of utilization of 2 GPUs per node, each process uses internal local multithreading. In that case, on the available cluster, the hardware resources signify the creation of maximum 2 threads per process.

Speedups are presented in comparison to the sequential execution of the OpenMP version. According to the results, sufficient speedup scaling is observed as long as there are

TABLE II
OVERALL EXECUTION TIME RESULTS IN MINUTES:SECONDS

| GPU Contexts | OpenMP | CUDA LOCAL | CUDA SDSM | CUDA SDSM-MT |
|---|---|---|---|---|
| 1 | 4243:10 | 137:05 | 136:59 | 137:02 |
| 2 | 2142:11 | 76:29 | 89:54 | 76:29 |
| 4 | 1128:27 | 134:24 | 61:07 | 60:39 |
| 6 | 1125:16 | 140:31 | 74:25 | 52:50 |
| 8 | 1126:30 | 156:09 | 68:50 | 47:32 |

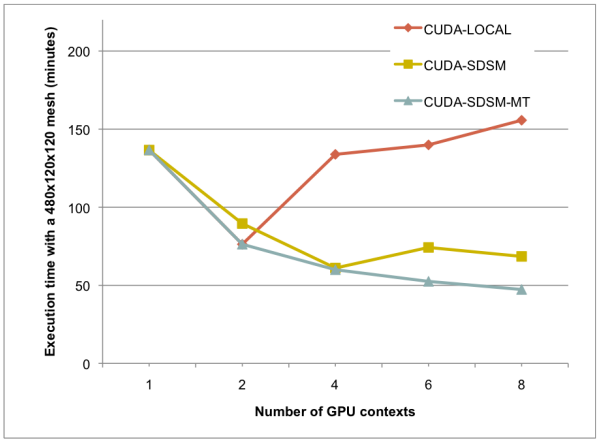


Fig. 7. Execution time results of the several multi-GPU schemes

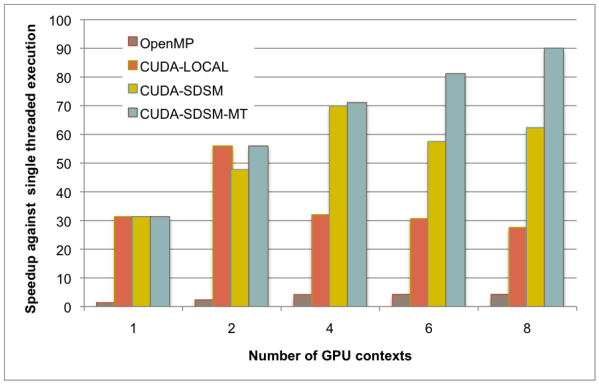


Fig. 8. Speedup in comparison with sequential execution

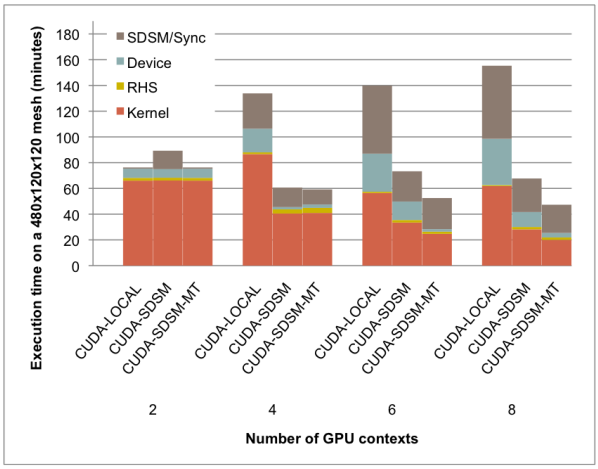


Fig. 9. Details of the execution time for the multi-GPU schemes

enough dedicated GPU devices and local multithreading is employed. In the current evaluation, this observation is valid for the *CUDA-SDSM-MT* execution mode. Under *CUDA-SDSM-MT* we observe adequate GPU utilization, which results to the highest speedup of an approximate factor of 90 compared with the sequential execution and a factor of 24x compared with the OpenMP local multithreaded execution on the CPU. An efficient execution is also achieved when multiple GPUs are used locally under *CUDA-LOCAL* and one context is created per GPU. The results that refer to *CUDA-LOCAL* under 4, 6 and 8 threads have been obtained by executing concurrently 2, 3 and 4 contexts respectively on each of the 2 GPUs that were locally available. As it was expected, this configuration does not exhibit a positive scaling, and it is presented here for the sake of completeness.

In contrast, we do not observe a positive scaling when multiple SDSM processes are spawned on each node to utilize the GPU devices. This fact is probably caused by the fact that interprocess communication - as it is realized on Intel Cluster OpenMP - is not able to perform efficiently at the local level. Therefore, the best strategy is to deploy a single SDSM process on each node of the cluster.

A detailed presentation of the time that is spent on the various phases of the simulation is shown in Fig. 9. These results show that a respectable decrease on the execution time of the kernels is observed as long as more GPUs are utilized. The kernel speedup is accompanied by a communication and synchronization cost that is increasing as the simulation incorporates more cluster nodes. Still, the cost at the SDSM layer is not increasing at the same pace that the kernel execution is decreasing. In that way the simulation is able to present a positive speedup when CUDA is combined with SDSM multithreading (*CUDA-SDSM-MT*). Finally, the cost that is depicted under the SDSM/Sync field in the case of *CUDA-LOCAL*, refers exclusively to the barrier synchronization. As no SDSM mechanisms is used, that cost is excessive due to the scheduling of more than 2 GPU contexts on just 2 GPU devices that are available on that node.

Overall, the divergence of *CUDA-SDSM-MT* from optimal speedup is mainly owed to the consecutive memory copies towards the memory hierarchy starting from the GPU to Host interface until the extension to the cluster level. Nevertheless, the execution follows a structured memory reference pattern and its impact does not forbid its acceleration.

## VI. CONCLUSIONS AND FUTURE WORK

In the current paper we have presented the implementation of a high-order accurate, computationally intensive, method for compressible flows in the context of GPU cluster computing. The implementation followed a novel approach in terms of

GPU cluster utilization that poses an interesting alternative to mainstream message passing. The multi-GPU computation on the cluster was carried out with the integration of our scheme in a modern Cluster OpenMP implementation over Software Distributed Shared Memory.

First results from the experimental evaluation show that, for specific applications, the presented approach is valid and can result in considerable acceleration of such simulations. The proposed implementation extends in a more consistent and transparent way the memory hierarchy of a GPU cluster, than message passing implementations do. Therefore, these preliminary results can encourage the implementation of appropriate middleware that will be based on the concept of SDSM and will be able also to offer an efficient programming model for heterogeneous GPU clusters.

Our future research will concentrate on providing such infrastructure at the middleware level. In parallel, we will aim to efficiently implement other also high-order accurate methods, that manipulate unstructured meshes and involve relatively more unbalanced parallel execution patterns.

## REFERENCES


[1] K. Olukotun and L. Hammond, "The Future of Microprocessors," *Queue*, vol. 3, no. 7, pp. 26–29, 2005.

[2] H. Sutter and J. Larus, "Software and the Concurrency Revolution," *Queue*, vol. 3, no. 7, pp. 54–62, 2005.

[3] J. Howard, S. Dighe, Y. Hoskote, S. Vangal, D. Finan, G. Ruhl, D. Jenkins, H. Wilson, N. Borkar, G. Schrom, F. Pailet, S. Jain, T. Jacob, S. Yada, S. Marella, P. Salihundam, V. Erraguntla, M. Konow, M. Riepen, G. Droege, J. Lindemann, M. Gries, T. Apel, K. Henriss, T. Lund-Larsen, S. Steibl, S. Borkar, V. De, R. V. D. Wijngaart, and T. Mattson, "A 48-Core IA-32 Message-Passing Processor with DVFS in 45nm CMOS," in *IEEE International Solid-State Circuits Conference*, San Francisco, California, USA, Feb 2010.

[4] E. Lindholm, J. Nickolls, S. Oberman, and J. Montrym, "NVIDIA Tesla: A Unified Graphics and Computing Architecture," *IEEE Micro*, vol. 28, no. 2, pp. 39–55, 2008.

[5] *NVIDIA CUDA Compute Unified Device Architecture, Programming Guide*, 2nd ed., NVIDIA Corporation, 2009. [Online]. Available: http://developer.download.nvidia.com/compute/cuda/2_2/toolkit/docs/

[6] *OpenCL - the Open Standard for Parallel Programming of Heterogeneous Systems*, Khronos Group, 2009. [Online]. Available: http://www.khronos.org/opencl/

[7] M. Pharr and R. Fernando, *GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation*. Addison-Wesley Professional, 2005.

[8] J. C. Thibault and I. Senocak., "CUDA Implementation of a Navier Stokes Solver on Multi-GPU Desktop Platforms for Incompressible Flows," in *47th AIAA Aerospace Sciences Meeting*, Orlando, Florida, USA, jan 2009.

[9] A. S. Antoniou, K. I. Karantasis, E. D. Polychronopoulos, and J. A. Ekaterinaris., "Acceleration of a Finite-Difference WENO Scheme for Large-Scale Simulations on Many-Core Architectures." in *AIAA Paper 2010-0525, 48th AIAA Aerospace Sciences Meeting*, Orlando, Florida, USA, jan 2010.

[10] Z. Fan, F. Qiu, A. Kaufman, and S. Yoakum-Stover, "GPU Cluster for High Performance Computing," in *SC '04: Proceedings of the 2004 ACM/IEEE conference on Supercomputing*. Washington, DC, USA: IEEE Computer Society, 2004, p. 47.

[11] D. A. Jacobsen, J. C. Thibault, and I. Senocak, "An MPI-CUDA Implementation for Massively Parallel Incompressible Flow Computations on Multi-GPU Clusters," in *48th AIAA Aerospace Sciences Meeting*, Orlando, Florida, USA, jan 2010.

[12] D. Göddeke, R. Strzodka, J. Mohd-Yusof, P. S. McCormick, H. Wobker, C. Becker, and S. Turek, "Using GPUs to Improve Multigrid Solver Performance on a Cluster," *International Journal of Computational Science and Engineering*, vol. 4, no. 1, pp. 36–55, Nov. 2008.

[13] J. C. Phillips, J. E. Stone, and K. Schulten, "Adapting a message-driven parallel application to GPU-accelerated clusters," in *SC '08: Proceedings of the 2008 ACM/IEEE conference on Supercomputing*. Piscataway, NJ, USA: IEEE Press, 2008, pp. 1–9.

[14] I. Gelado, J. E. Stone, J. Cabezas, S. Patel, N. Navarro, and W.-m. W. Hwu, "An asymmetric distributed shared memory model for heterogeneous parallel systems," in *ASPLOS '10: Proceedings of the fifteenth edition of ASPLOS on Architectural support for programming languages and operating systems*. New York, NY, USA: ACM, 2010, pp. 347–358.

[15] M. Strengert, C. Mller, C. Dachsbacher, and T. Ertl, "CUDASA: Compute Unified Device and Systems Architecture," in *Eurographics 2008 Symposium on Parallel Graphics and Visualization (EGPGV'08)*, K.-L. M. Daniel Weiskopf, Jean M. Favre, Ed. Eurographics Association, 2008, pp. 49–56.

[16] E. Elsen, P. LeGresley, and E. Darve, "Large calculation of the flow over a hypersonic vehicle using a GPU," *J. Comput. Phys.*, vol. 227, no. 24, pp. 10 148–10 161, 2008.

[17] P. Micikevicius, "3D finite difference computation on GPUs using CUDA," in *GPGPU-2: Proceedings of 2nd Workshop on General Purpose Processing on Graphics Processing Units*. New York, NY, USA: ACM, 2009, pp. 79–84.

[18] J. M. Cohen and J. Molemake, "A Fast Double Precision CFD Code Using CUDA," in *21st International Conference on Parallel Computational Fluid Dynamics (ParCFD2009)*, 2009. [Online]. Available: http://www.jcohen.name/papers/Cohen_Fast_2009_final.pdf

[19] T. Brandvik and G. Pullan, "An Accelerated 3D Navier-Stokes Solver for Flows in Turbomachines," *Accepted for publication in the ASME Transactions, Journal of Turbomachinery (2009)*, 2009.

[20] A. Corrigan, F. Camelli, R. Lhner, and J. Wallin, "Running Unstructured Grid CFD Solvers on Modern Graphics Hardware," *AIAA*, vol. 28, no. 4, pp. 13–27, 2008.

[21] A. Klöckner, T. Warburton, J. Bridge, and J. S. Hesthaven, "Nodal discontinuous Galerkin methods on graphics processors," *J. Comput. Phys.*, vol. 228, no. 21, pp. 7863–7882, 2009.

[22] G.-S. Jiang and C.-W. Shu, "Efficient implementation of weighted ENO schemes," *J. Comput. Phys.*, vol. 126, no. 1, pp. 202–228, 1996.

[23] D. S. Balsara and C.-W. Shu, "Monotonicity preserving weighted essentially non-oscillatory schemes with increasingly high order of accuracy," *J. Comput. Phys.*, vol. 160, no. 2, pp. 405–452, 2000.

[24] J. A. Ekaterinaris, "High-order accurate, low numerical diffusion methods for aerodynamics," *Progress in Aerospace Sciences*, vol. 41, no. 3-4, pp. 192 – 300, 2005. [Online]. Available: http://www.sciencedirect.com/science/article/B6V3V-4GPTDH6-1/2/997c3ccbed3881b7aec052328be1133a

[25] C.-W. Shu, "Total-Variation-Diminishing Time Discretizations," *SIAM Journal on Scientific and Statistical Computing*, vol. 9, no. 6, pp. 1073–1084, 1988. [Online]. Available: http://link.aip.org/link/?SCE/9/1073/1

[26] G. Erlebacher, M. Y. Hussaini, C. G. Speziale, and T. A. Zang, "Toward the large-eddy simulation of compressible turbulent flows," *Journal of Fluid Mechanics Digital Archive*, vol. 238, no. -1, pp. 155–185, 1992.

[27] F. Nicoud and F. Ducros, "Subgrid-scale stress modelling based on the square of the velocity gradient tensor," *Flow, Turbulence and Combustion*, vol. 62, no. 3, pp. 183–200, 09 1999. [Online]. Available: http://dx.doi.org/10.1023/A:1009995426001

[28] E. Lenormand, P. Comte, L. T. Phuoc, and P. Sagaut, "Subgrid-Scale Models for Large-Eddy Simulations of Compressible Wall Bounded Flows," *AIAA Journal*, vol. 38, pp. 1340–1350, Aug. 2000.

[29] J. P. Hoeflinger, "Programming with cluster openMP," in *PPoPP '07: Proceedings of the 12th ACM SIGPLAN symposium on Principles and practice of parallel programming*. New York, NY, USA: ACM, 2007, pp. 270–270.

[30] C. L. Gardner, J. Glimm, O. McBryan, R. Menikoff, and D. Sharp, "The dynamics of bubble growth for Rayleigh-Taylor unstable interfaces," *NASA STI/Recon Technical Report N*, vol. 88, pp. 12 027–+, May 1987.

[31] Y.-N. Young, H. Tufo, A. Dubey, and R. Rosner, "On the miscible Rayleigh-Taylor instability: two and three dimensions," *Journal of Fluid Mechanics*, vol. 447, no. -1, pp. 377–408, 2001.

[32] C. Amza, A. L. Cox, S. Dwarkadas, P. Keleher, H. Lu, R. Rajamony, W. Yu, and W. Zwaenepoel, "TreadMarks: Shared Memory Computing on Networks of Workstations," *IEEE Computer*, vol. 29, no. 2, pp. 18–28, 1996. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.52.4125