Parallel Computing and Dynamic Load Balancing Techniques for DG-WENO schemes

Shyam Sundar * K. S. Yeo †

October 16, 2009

1 Introduction

In the last few decades, Computational Fluid Dynamics (CFD) has evolved to a fairly mature state. With the available high performance computers, many realistic simulations such as flow around an aircraft, combustion in internal combustion engines etc. are now realizable. However, obtaining an accurate solution representing all the flow characteristics is still a challenge in the field of CFD. A fluid flow is characterized by the presence of viscous diffusion, turbulence, shocks and discontinuities, generation of acoustic waves etc.. These flow characteristics can result in complex interactions (acoustic-shock, shock-boundary layer, acoustic-boundary layer etc.) which, in most cases, are impractical to solve analytically. The conventional CFD schemes like JST schemes are not accurate enough to capture these interactions accurately. In recent years there has been a renowned interest to devise a single scheme to accurately represent all the flow characteristics. Such flow solvers should be able to solve problems of different scales of fluctuations, ranging from strong shocks, to acoustic noise generation and propagation. In this aspect, the higher order Weighted Essentially Non Oscillatory (WENO) schemes and the higher order Discontinuous Galerkin (DG) methods for flow problems were developed and have been successfully applied for flow problems.

The high resolution, high order schemes require large computational cost and hence computational time. Reducing the computational cost is in itself a challenging problem. The computational time, however, can be reduced by utilizing more than one CPU for arriving at a solution. This process is in general denoted as parallel computing. In a parallelized solver, the job is split in to smaller problems which communicate among each other. The additional cost incurred due to this communication determines the efficiency of the parallelization process. An efficient parallel solver should have very low communication costs and reduce the idle times of the processors. In problems such as adaptive meshing, or high resolution shock capturing schemes, the nature of the solution and the resulting computational cost vary. Due to the difference in computational cost, few processors might be over-loaded while the other processors are idling. For such cases, load balancing techniques can be applied for distributing the load from a over-loaded CPU to the least loaded CPU.

In this work, a higher order DG scheme is formulated with a new higher order compact WENO scheme for problems involving strong shocks. For the first time compact WENO computations

*PhD student, Mechanical Engineering, NUS (g0501014@nus.edu.sg)
†Associate Proffessor, Mechanical Engineering, NUS (mpeyeoks@nus.edu.sg)

of orders higher than 3 are reported. The existing literature reports only orders upto 3. The DG-WENO code developed with this scheme is parallelized using MPI libraries. A load balancing technique is formulated to address the difference in computational cost arising due to the compact WENO reconstructions.

2 Runge Kutta - Discontinuous Galerkin Method

Consider a general hyperbolic system given by

$$\frac{\partial Q}{\partial t} + \nabla \cdot \vec{F} = 0 \tag{1}$$

where Q is the solution and $\vec{F}$ is the corresponding numerical flux vector. For the Euler equations, the conserved variables and the corresponding fluxes are given by

$$Q = \begin{pmatrix} \rho \\ \rho u_i \\ \rho e \end{pmatrix} \qquad F_j = \begin{pmatrix} \rho u_j \\ \rho u_i u_j + \boldsymbol{p}\delta_{ij} \\ u_j \left(\rho e + \boldsymbol{p}\right) \end{pmatrix} \tag{2}$$

where ρ, u_i $\boldsymbol{p}$ and e are the density, velocity, pressure and internal energy of the fluid respectively, and δ_{ij} is the Kronecker delta function. The Euler equation is closed with the equation of state given by

$$\boldsymbol{p} = (\gamma - 1)\,\rho \left(e - \frac{1}{2}U^2\right)$$

where γ is the specific heat ratio (1.4 for air) and U is the velocity magnitude. To solve the governing equation set given in equation 1, the standard Runge Kutta Discontinuous Galerkin (RK-DG) method [3] is used to solve the system of governing equations. We describe the RK-DG method in brief in the following sections.

2.1 Spatial Discretization

To obtain the Discontinuous Galerkin scheme formulation, the equation 1 is expressed in a weak form by multiplying the terms with a trial function ψ. This results in

$$\psi\frac{\partial Q}{\partial t} + \psi\nabla \cdot \vec{F} = 0 \tag{3}$$

The computational domain is divided into smaller elements Ω. On integrating the equation 3 over the element Ω, we obtain

$$\int_\Omega \psi\frac{\partial Q}{\partial t}d\Omega + \int_{\partial\Omega} \psi\vec{H}\cdot\vec{n}d\Gamma - \int_\Omega (\nabla\psi)\cdot\vec{F}d\Omega = 0 \tag{4}$$

where $\partial\Omega$ is the boundary of the cell Ω, $\vec{n}$ is the unit normal vector of the boundary and $\vec{H}$ is the Riemann flux at the boundary. The element is taken to be a linear triangle with vertices (0,0), (1,0) and (0,1) constructed in a reference plane (ξ, η). Appropriate transformations are applied to transform the equations from the physical plane (x, y) to the reference plane (ξ, η). The solution Q is expanded with a set of basis function $\phi(\xi, \eta)$ and corresponding coefficients α

as

$$Q \equiv \alpha_i \phi_i \tag{5}$$

For simplicity, we use the tensor notation, implying the summation of index appearing twice. If the trial function ψ is expanded with the same basis polynomial ϕ, the equation 4 is written as

$$\int_\Omega \phi_j \phi_i d\Omega \frac{d\alpha_i}{dt} + \int_\Gamma \phi_j \vec{H} \cdot \vec{n} d\Gamma - \int_\Omega (\nabla \phi_j) \cdot \vec{F} d\Omega = 0 \tag{6}$$

Note that the time derivative of α is the total derivative as α is invariant in space.

2.2 Time discretization

Equation 6 is marched using a low storage m-stage m^{th} order Strong Stability Preserving Runge Kutta (SSP-RK) [1, 2] scheme given by

$$\alpha^{(0)} = \alpha^n \tag{7}$$

$$\alpha^{(i)} = \alpha^{(i-1)} + \Delta t \frac{d}{dt} \alpha^{(i-1)} \tag{8}$$

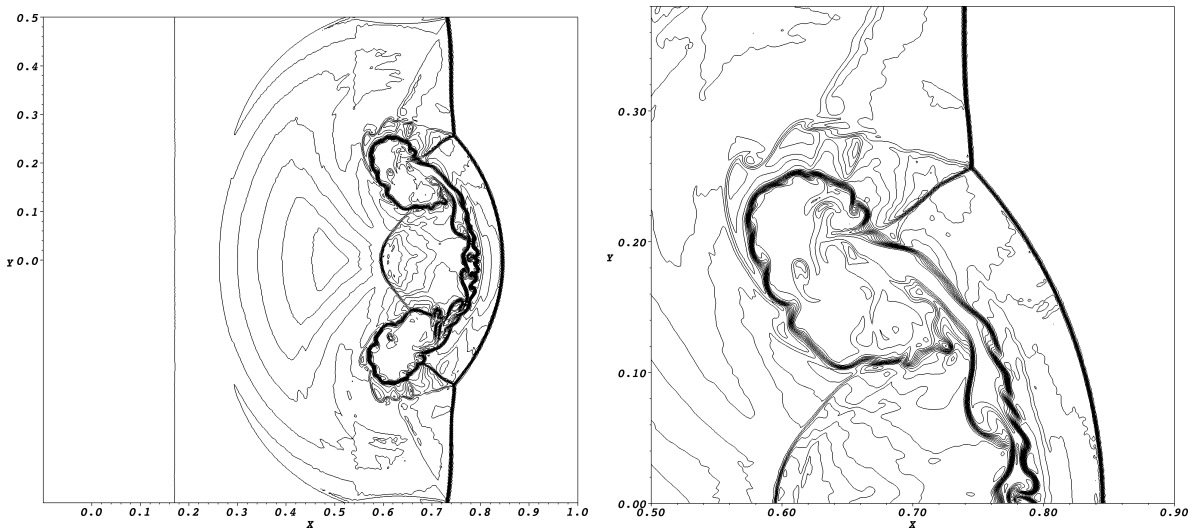
$$\alpha^{n+1} = \sum_{k=0}^{m} C_{m,k} \alpha^{(k)} \tag{9}$$

where $\alpha^{(k)}$ is the intermediate values of α at k^{th} RK stage and the coefficients $C_{m,k}$ of the m stage RK scheme are obtained as in [1]. The order of time integrator is matched with the spatial order of representation of the solution Q in equation 5.

3 WENO Shock Capturing Method

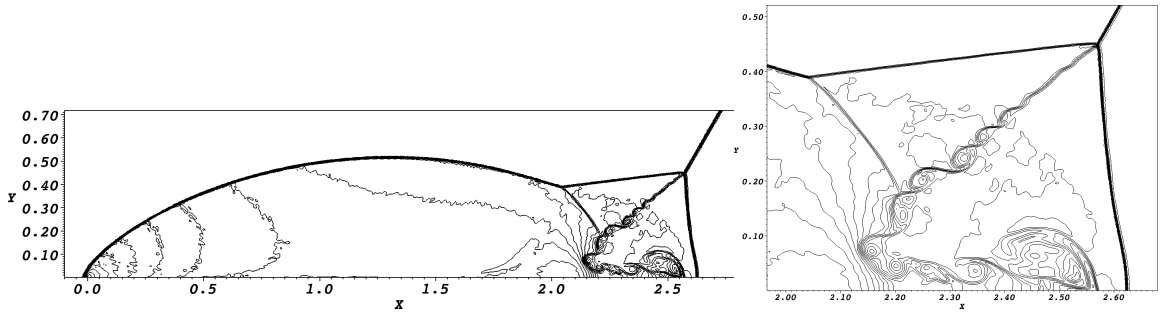
In the traditional WENO schemes, the solution reconstruction process should require a large stencil comprising of elements in the neighborhood. In the proposed formulation, the WENO scheme is compact, depending on only the immediate neighboring elements. This makes the WENO scheme more suitable for parallelization since the communication is restricted only to the immediate neighbors. The formulation of the WENO scheme and the test cases are detailed in the article attached along with this report. A few selected results are shown in Figures 1 and 2 to illustrate the advantages of high resolution schemes in the presence of complex shock interactions. The high resolution plots are obtained by subdividing the elements into smaller elements.

The details of these initial value problems can be found in the attached article. As mentioned earlier, a new oscillation detector is used to identify the troubled cells where the WENO schemes should be applied. Figure 3 shows the localized regions identified by the oscillation detector for the double mach reflection problem. Only 12% of the entire flow domain requires WENO reconstruction, and the computational cost is reduced to almost one third.



(a) Low resolution Plot (b) High resolution plot

Figure 1: 3^{rd} order solution of a shock colliding on a gas bubble: $t = 0.2$



(a) Low resolution plot (b) High resolution plot

Figure 2: 3^{rd} order solution of a Double Mach Reflection of Mach 10 shock:$t = 0.2$

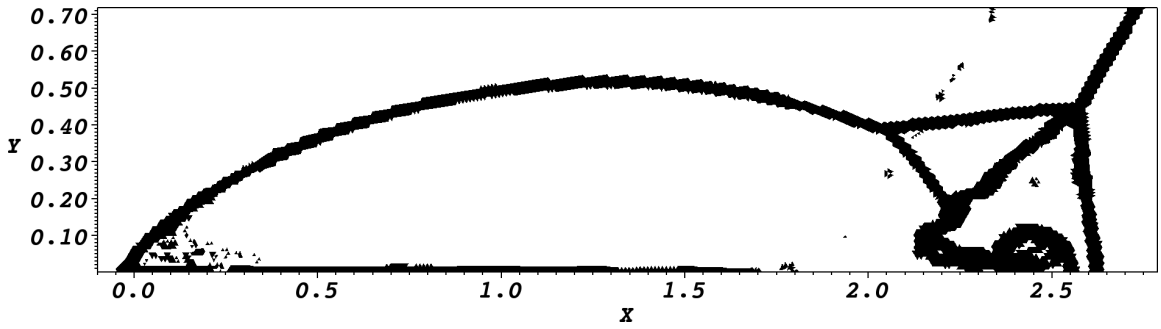


Figure 3: Troubled cells in a 3^{rd} order solution of a Double Mach Reflection

4 Parallelization of Flow Solver

As detailed in Section 2.2, an explicit low storage Runge Kutta scheme is used for temporal discretization. Each stage of the time stepping scheme is solved as

$$\alpha^{(i)} = \alpha^{(i-1)} + \Delta t \frac{d}{dt}\alpha^{(i-1)} \tag{10}$$

where the first order time derivative is obtained from the spatial scheme (DG-WENO). At each stage, the following steps are performed

1. Compute the inter-cellular fluxes
2. Compute the internal fluxes
3. Estimate the temporal derivative from equation 6
4. Estimate coefficients α^i for current stage
5. Remove spurious oscillations in α^i using WENO

It can be seen that the steps 1 and 5 are dependent on element neighbors while the other steps depend on the available internal values of the element. A single block solver can be parallelized with the modification of the steps 1 and 5 to take care of the partition boundaries. The paralleization strategy adopted in this work follows a similar concept. Using the Obeject Oriented Programing System (OOPS) in C++ the existing single block solver is wrapped with the MPI communication routines. This method is also useful in the case of dynamic load balancing, where the blocks are transferred between different processors.

To maintain brevity, the only the overview of paralleization process is detailed in the report. The different tasks involved in parallelizations are detailed below.

4.1 Partitioning the mesh

The open source codes GMSH and METIS are used for mesh generationand mesh partitioning. In the first stage of partitioning, the mesh is divided into N_b number of partitions called blocks. Each element is given a unique identification number, which does not vary across the blocks. In the second step, the graph containing the blocks and their connectivity is again partitioned into N_p groups, so that, each of the blocks are assigned to a processor. This stage requires block to block connectivity information. Two blocks are marked as neighbors if any element from a block is connected to an element belonging to other block. With this two stage partitioning, a processor can now contain more than one block. This architecture is useful for load balancing as the blocks can be transferred between the processors.

4.2 Inter block communication

The inter block communications occur at the face level of the bordering elements. A face is marked as block boundary when the two neighboring elements sharing the face belong to different blocks. For a given block a list of faces sharing a boundary with a neighboring block is matched to a similar list formed in the neighboring block. To illustrate it, if $\mathcal{B}(i,j)$ is the list in i^{th} mesh containing the faces that share the boundary with j^{th} mesh, we have the condition $\mathcal{B}(i,j) =$

$\mathcal{B}(j,i)$. For the interface fluxes, only the face value need to be obtained. However, for WENO reconstructions, the entire solution matrix of the neighboring element is required. Since WENO schemes are only applied in small region of the computational domain, only selected solutions are transferred across the blocks.

Communicating the neighboring face values for each face in the list increases the unwanted communication cost. To reduce this, values are lumped into a single block of memory which is then transferred and processed on the neighboring block. For the exchange of data, a non-blocking send and a blocking receive is used to exchange the data with the neighbors. A non-bocking send can improve the efficiency of a code parallelized with MPI, while maintaining simplicity in synchronization. In the current work, the first processor also acts as a server collecting the information (such as Δt) and broadcasting to other processors. This methods utilizes all the CPUs allocated for the current job.

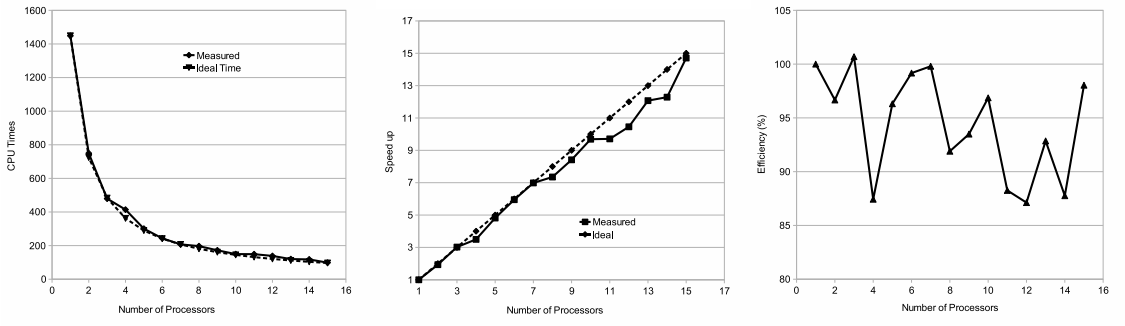
4.3 Dynamic Load Balancing

When the computational cost of each block vary with time, load balancing techniques can be applied to re-distribute the work load to the idle processors. This dynamic load balancing is performed in the second stage of mesh partitioning. Each block is weighted by its respective computational cost and a block-graph is generated using the block to block connectivity. This graph is partitioned using METIS library to N_P groups belonging to each processor. The blocks and their associated data are then moved to the newly assigned processor. To avoid additional computational cost, this step is performed at a specific interval. The number of blocks is taken to be a multiple of the number of processors. The relative size of the block with respect to processor region is defined by the granularity. The granularity of the load balancing is given by an integer value defined as

$$g = \frac{N_b}{N_P} \tag{11}$$

A larger value of g would result in decrease in the size of blocks. Smaller block sizes can aid in effective work load distribution among the processors. However, to avoid errors in measuring the computational cost, a block should have the computational cost significantly higher than the clock precision (usually in order of milli seconds) supported by the CPU architecture. Hence, it can not be too small. A larger number of blocks could also increase the processing time for block to block communication. Thus, an optimum value of g should be decided based on the nature of computation. Since we do not use multi-threading, a non-blocking send is required to enable all the blocks belonging to a single processor to send the data and a blocking receive is sufficient to receive the data from neighboring blocks.

In the application of load balancing for a DG-WENO scheme, a block containing the troubled cells (where WENO reconstruction is applied) will have a higher computational time compared to the normal blocks. A difference in computational cost of the processors can cause bottlenecks and increase the idle wait time of other processors. The current load balancing technique ensures an equal distribution of the computational cost to the processors by re-distributing the blocks. In a transient computation, since the shocks and the interactions change with time, this configuration would not be efficient over longer time intervals. Thus, a suitable interval should be chosen for load balancing.



(a) CPU Time (b) Speed up (c) Efficiency

Figure 4: Parallel computing of 3^{rd} order solution in Atlas Cluster

5 Results and Discussion

5.1 Effectiveness of Parallelization

In order to test the pure parallel computing efficiency, we disable the WENO reconstruction and solve the Double Mach problem with a time step $\Delta t = 0$ for specified number of iterations. The ideal computational time is given by

$$T_{Ideal} = \frac{T(1)}{N_P} \tag{12}$$

where $T(N_P)$ is the computational time for N_P processors. The parallel computing efficiency is calculated as

$$Efficiency = \frac{T(1)}{N_P T(N_P)} \tag{13}$$

and the corresponding speed-up achieved is given by

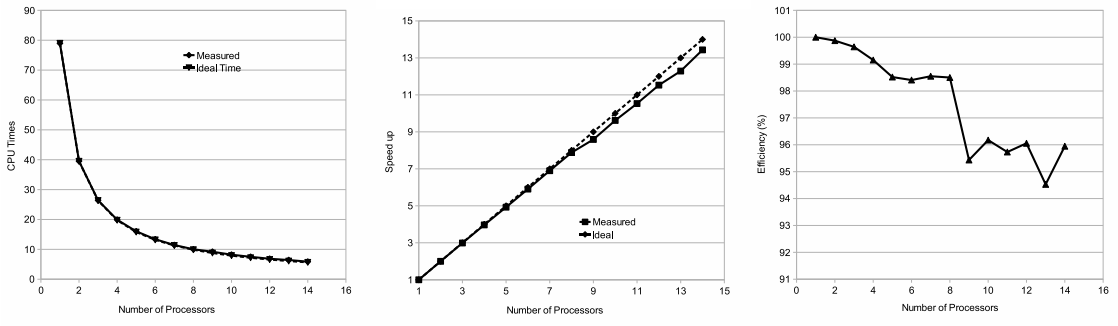
$$Speedup = \frac{T(1)}{T(N_P)} \tag{14}$$

For measuring the performance of the parallel code, the nodes should be free of any other CPU work load and the available bandwidth should be at optimum level. Here, we present the results obtained from the Atlas cluster of SVU lab as well as the ROCKS cluster at Temasek Laboratories. At the time of job submission, the Atlas cluster was fully loaded while the ROCKS cluster was completely idle. Due to this, a larger number of iterations are performed in Atlas cluster to obtain the performance characteristics. In all the cases, MPI_Wtime function is used to measure the wall clock time.

The computational times, speed up and efficiency of the parallel code (order 3) run in Atlas cluster are plotted in Figure 4. A similar computation set was run on ROCKS cluster for orders 2 and 3 and the results are plotted in Figures 5 and 6.

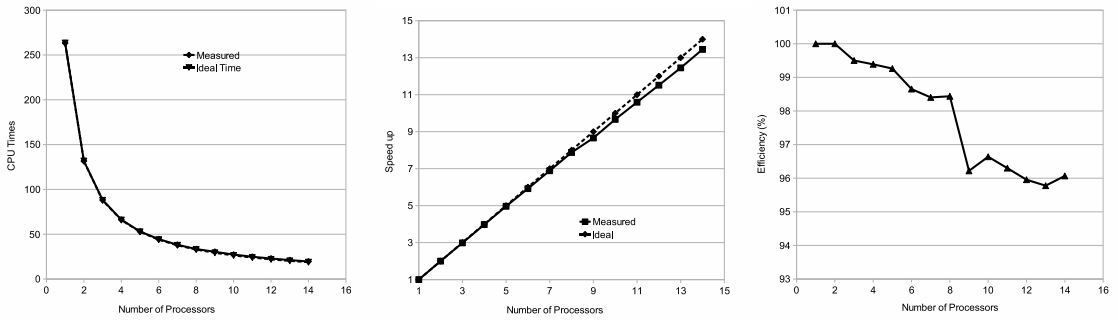
The parallel computation efficiency is around 93% and decreases gradually with increase in number of CPUs. A fluctuation of $\pm 5\%$ is observed in the efficiency. This trend could be because of the error generated due to difference in loads on the nodes and the available bandwidth for MPI communications.

The computations performed in ROCKS cluster show excellent performance characteristics with smoother trends. Two sets of solutions of orders 2 and 3 are computed to analyze the effect



(a) CPU Time (b) Speed up (c) Efficiency

Figure 5: Parallel computing of 2^{nd} order solution in ROCKS Cluster



(a) CPU Time (b) Speed up (c) Efficiency

Figure 6: Parallel computing of 3^{rd} order solution in ROCKS Cluster

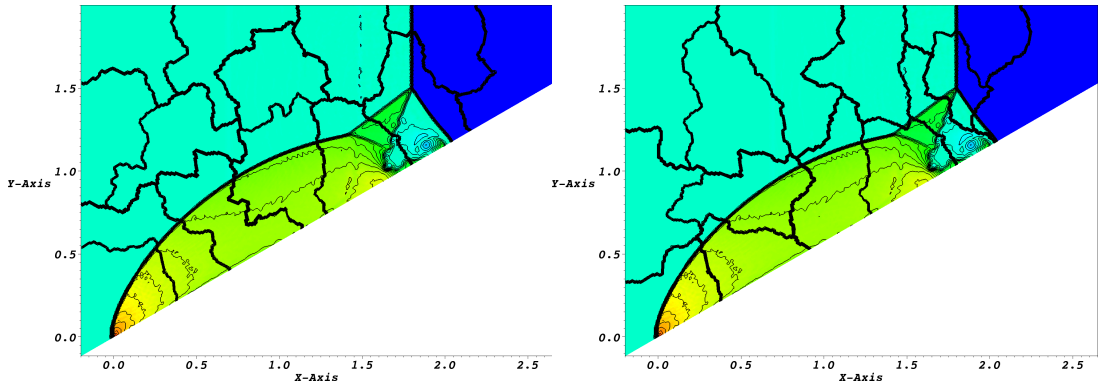
of total computational cost of the problem with respect to code performance. The 3^{rd} order DG scheme is twice as costly as 2^{nd} order DG scheme in the same mesh. For larger problems, the relative cost of block to block communication is very less compared to the total computational cost. This can be felt as an improvement in parallel efficiency with increase in problem size. A similar trend is observed for the computations presented in Figures 5 and 6. The computations corresponding to the 3^{rd} order solution have a slight improvement of around 1% in parallel efficiency compared to the 2^{nd} order solution set.

5.2 Dynamic Load Balancing

For the initial setup, we use the solution of double mach reflection problem at time $t = 0.2$ and a CFL of 0.2. The code is run in Atlas cluster for 1000 iterations. WENO reconstructions are employed to remove spurious oscillations in the solution. Since this contains shock interactions evolving with time, it would be ideal test case for the benchmarking the load balancing techniques. The number of processors are set to constant and the results are compared with different N_b values. Note that for efficient load balancing, N_b should be a multiple of N_P. Load balancing is disabled if there is only one block per processor ($g = 1$).

The flow solution and the change in block distribution for $g = 6$ is shown in Figure 7. It can be seen that the regions associated with the processors reduce in size near the shock regions. As expected, the processor region is smallest at the triple mach region where the solution is more oscillatory. Similarly, the processor regions are enlarged at regions that do not contain shock.

The computational time without load balancing is measured to be 1152 seconds, while the



(a) Before load balancing

(b) After load balancing

Figure 7: Computational region allocated for processors before and after load balancing

computational time with load balancing is measured as 995 seconds. A 14% decrease in computational time can be still considered significant as the WENO scheme is applied only in troubled region. Besides, the above problem is transient in which the solutions changes with time. The relative computational cost of load balancing is very low compared to the total computational cost as only the block graph is used for load balancing. This results in partitioning only $N_P \times N_b$ elements as opposed to the total number of elements in the mesh.

6 Summary

The newly proposed DG-WENO scheme is parallelized with the help of MPI libraries. The code is tested in two different clusters to obtain a machine independent evaluation of parallel performance. Significant improvement in computational time is achieved in both cases. An average parallel efficiency of of around 93% is observed in the Atlas cluster while an average efficiency of 97% is observed in ROCKS cluster. The difference in efficiency could be due to the existing computational loading of the clusters and the usage of bandwidth by other process. The results obtained from ROCKS cluster are computed with the cluster dedicated entirely to the parallel code. The trends reported in this work show that the code performs remarkably good in terms of parallel efficiency.

The WENO reconstructions are computationally expensive and hence are performed only at oscillatory regions. This cause a load imbalance among the processors. To avoid overloading of the processors due to these expensive WENO reconstructions, a load balancing technique is formulated with the help of METIS libraries. The processors are allocated optimal number of blocks depending on their respective computational cost. Recent test results show that the computational time is decreased by around 14% for a initial solution obtained from Double Mach Reflection problem.

The current work can be applied to a wide variety of problems ranging from acoustic scattering to the shock interaction problems. Viscous computations are also be performed, though not reported in this work. This makes the solver one of the most accurate available till date. For the future works, the code will be applied for problems pertaining to acoustic sound generation (like vocal chords), shock interactions with boundary layers, pulse detonation engines etc. One of the important factor to note is that the same code can be applied for all the above problems without any change in the source.

References

[1] Sigal Gottlieb, Chi-Wang Shu, and Eitan Tadmor. Strong stability-preserving high-order time discretization methods. *SIAM Review*, 43(1):89–112, 2001.

[2] Ethan J. Kubatko, Joannes J. Westerink, and Clint Dawson. Semi discrete discontinuous galerkin methods and stage-exceeding-order, strong-stability-preserving runge-kutta time discretizations. *Journal of Computational Physics*, 222(2):832 – 848, 2007.

[3] Jun Zhu, Jianxian Qiu, Chi-Wang Shu, and Michael Dumbser. Runge-kutta discontinuous galerkin method using weno limiters ii: Unstructured meshes. *Journal of Computational Physics*, 227(9):4330 – 4353, 2008.