Implementation and Performance Analysis of a Parallel Algebraic Multigrid Solver

Tony Youssef Saad

Mechanical Engineering, American University of Beirut.

Ain El Mreiseh, Beirut, Lebanon

tys01@aub.edu.lb

Abstract

In this paper, we present the performance analysis of a parallel Algebraic Multigrid Solver (AMG) for a finite volume unstructured CFD code. The parallel performance of the AMG is tested against the sequential implementation for a number of grid sizes and partitions in order to evaluate its scalability with increased mesh sizes. The performance is evaluated as a function of speed-up measures based on total cpu time. Results for an unstructured mesh indicate that the pAMG solver scalability depends on the mesh size and number of partitions. For a shadow to core elements ratio larger than 0.02 the scalability of the solver starts to deteriorate.

Keywords

Parallel Computing, Algebraic Multigrid, CFD, Finite Volume Method.

INTRODUCTION

Today's scientific computing communities are readily migrating their codes to parallel architectures using PC clusters as a cost effective alternative to classical parallel supercomputers [1]. The performance offered by the latest generation Pentium and AMD processors as well as the advancement in the manufacturing of efficient high speed interconnection networks such as Myrinet and Gigabit Ethernet, have allowed the set up of powerful PC clusters that can compete with more traditional parallel systems as illustrated by the TOP500 supercomputer sites list [2].

CFD simulations deal with a set of highly non-linear partial differential equations, which upon discretization are translated into a system of sparse algebraic equations. Iterative solvers are usually used for solving such systems because of their lower computational requirements both in memory and cpu time and since the presence of nonlinearities in the problem necessitates the assembly and solution of these equations repeatedly. However the performance of iterative solvers tend to decrease with increased mesh size; to this end convergence acceleration techniques such as multigrid methods [1] are usually employed with iterative solvers. In this work the Algebraic Multigrid Method was selected because of the simplicity of its implementation for Finite Volume methods. In the AMG, the different grid levels are constructed through an agglomeration of the equations using the original coefficient matrix.

In what follows, the Finite Volume methods and AMG are briefly presented, followed by a description of the parallelization approach that was used. Finally, results for the performance of the p AMG solver for different grid sizes and partition numbers are presented.

THE FINITE VOLUME METHOD

The FVM is a numerical method aimed at the solution of partial differential equations and especially tuned to those arising in fluid, heat and mass transfer problems. The general PDE governing the transport of a conserved passive scalar has the following form

$$\underbrace{\frac{\partial \rho\phi}{\partial t}}_{\text{Transient Term}} + \underbrace{\nabla\cdot(\rho \boldsymbol{u}\phi)}_{\text{Convection Term}} = \underbrace{\nabla\cdot(\Gamma\nabla\phi)}_{\text{Diffusion Term}} + \underbrace{S_\phi}_{\text{Source Term}} \quad (1)$$

where ρ denotes the density, t is the time, $\boldsymbol{u}$ is any present velocity field, Γ is the diffusivity and ϕ is the unknown scalar for which we seek a solution. Upon discretizing equation (1), we end up for an arbitrary control volume with an algebraic equation, , of the form

$$a_P\phi_P + \sum_{i=NB(P)} a_{Ni}\phi_{Ni} = b_P \quad (2)$$

where the coefficients depend on the specific schemes used in the discretization process [3]. The solution of these equations is the function of the linear iterative solver.

GENERAL ITERATIVE METHODS

The system of equations can be written in the form

$$\boldsymbol{A}\cdot\boldsymbol{x} = \boldsymbol{b} \quad (3)$$

where $\boldsymbol{A}$ is a $N\times N$ matrix, $\boldsymbol{x}$ is a $N\times 1$ vector of the unknowns and $\boldsymbol{b}$ is a $N\times 1$ vector usually called the source or load vector. The general principle in all iterative methods is that given an approximate solution $\boldsymbol{x}^k$, we wish to obtain a better approximation $\boldsymbol{x}^{k+1}$ and then repeat the whole process. One of the most familiar methods that takes advantage of the sparsity of the coefficient matrix is the Incomplete LU decomposition, *ILU*. In the standard ILU method, the coefficient matrix is decomposed into the product of a lower and an upper triangular matrix that have nonzero entries only where the original coefficient matrix entries are nonzero. Such a factorization is written in the form

$$A = (D + U_A)D^{-1}(D + L_A) \quad (4)$$

where U_A and L_A are the strictly upper and lower triangular parts of A while D is the diagonal of A.

The Algebraic Multigrid

Most of the standard iterative solvers (Jacobi, Gauss-Siedel, SOR) are efficient in removing the high frequency errors. However, once the high frequency errors have been removed, the remaining low frequency or smooth errors have a substantial negative effect on the convergence of the iterative solver. The multigrid method provides a means of defining a hierarchy of coarse grids on which the low frequency error is redistributed in the form of a high frequency error thus accelerating the performance of the iterative solver. There are two approaches for implementing a multigrid method: geometric and algebraic. In the geometric approach, the agglomeration is based on a relation between the elements of the grid while the algebraic approach is based on a certain relation between the mutual coefficients of elements. In the current implementation we follow the algebraic approach. In this respect, the coefficients of the coarser level are derived by summing up the appropriate coefficients of the fine level

$$A_{I,J}^{(l+1)} = \sum_{i \in G_I} \sum_{j \in G_J} A_{i,j}^{l} \tag{5}$$

while the source vector at the coarser level is defined as the sum of the residuals of the appropriate cells at the fine level

$$b_I^{(l+1)} = \sum_{i \in G_I} r_i^{(l)} \tag{6}$$

where the residual is defined as

$$\boldsymbol{r}^k = \boldsymbol{b} - \boldsymbol{A} \cdot \boldsymbol{x}^k \tag{7}$$

An example of such an agglomeration is shown in figure 1. The interpolation of the residual from the fine level to the coarse level is called *restriction* while that from the coarse level to the fine one is called *prolongation*.

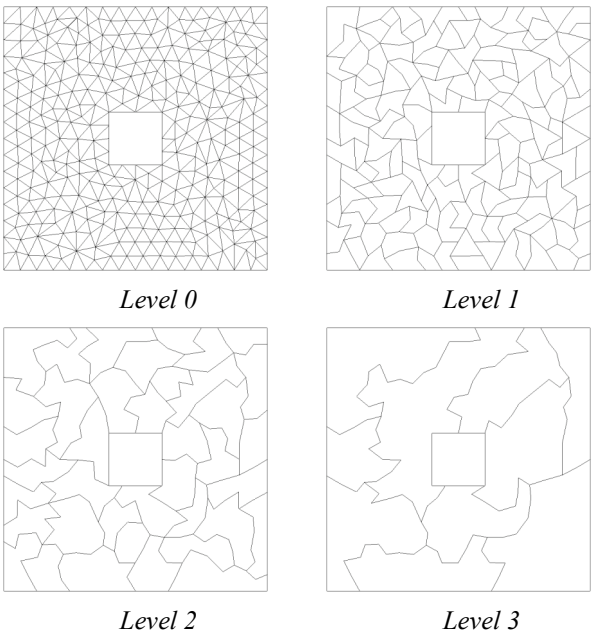

Figure 1: AMG agglomeration of an unstructured mesh

AMG Cycles

Once the coarse grids have been generated, one can define several ways to visit these coarse grids, this is called multigrid cycling. There are two categories of cycling methods in multigird: fixed and flexible cycles [4]. In this paper, we have implemented the fixed F cycle in which each of the coarse grids is visited successively until all coarser grids have been visited. This is illustrated in figure 2.

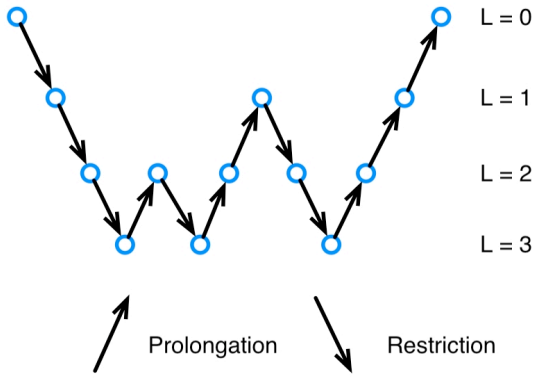

Figure 2: F Cycle

During restriction, the solver performs 2 sweeps on fine levels and 5 sweeps on the coarsest level while during prolongation; it performs only 1 sweep on all visited levels.

PARALLEL CFD

The parallelization of the CFD code is based on domain decomposition and the SPMD programming model (Single Program Multiple Data). Such an approach, usually called *coarse grain* parallelization, is more suitable when using a cluster of personal computers as a parallel computer. The steps followed in the parallelization will be discussed in the following sections.

Mesh Partitioning

A good mesh partitioner is one that divides the computational load equally amongst partitions and minimizes the amount of communication between partitions. We use the METIS [5] mesh partitioner because of its robustness and ease of use. Figure 3 shows an example of 4 partitions generated using METIS.

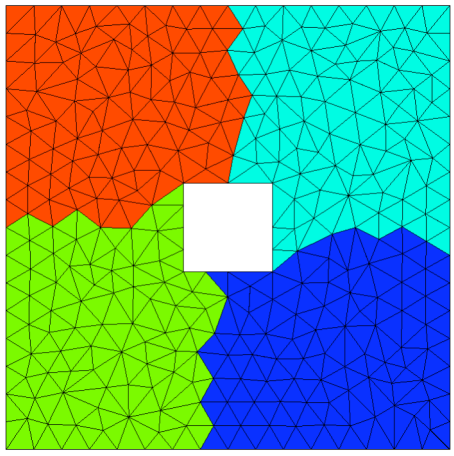

Figure 3: Mesh partitioning example

Partition Renumbering and Shadow Elements

Once the mesh has been partitioned and the partitions sent to the respective processors, they need to be re-numbered locally, so that each partition can be set as a nearly independent problem. An example of a renumbering scheme is shown in Figure 4.

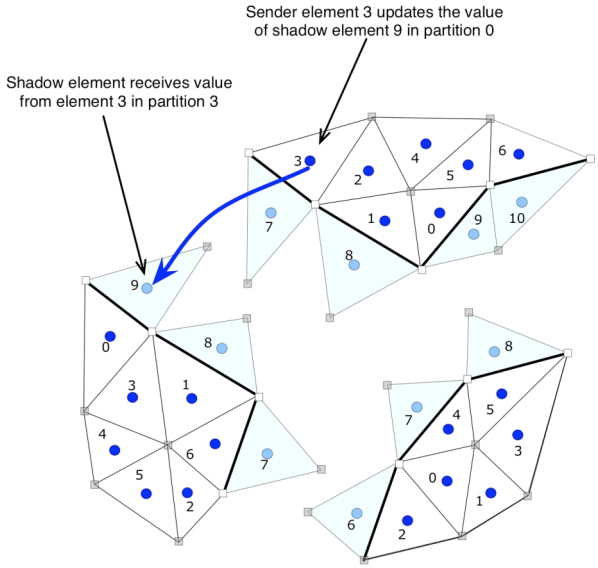

Figure 4: Partition renumbering

In the process of renumbering the partition entities, and in order to keep the coupling (and therefore enforce conservation) between the various partitions, shadow elements at the interface of each partition have to be correctly defined as depicted in figure 4. The shadow elements are numbered locally according the number of the concerned partition and they are usually placed at the end of the list of elements. Each partition has to have knowledge about its shadow elements and sender elements. Sender elements are defined as the elements that update the values of shadow elements in neighboring partitions. For example, as figure 4 shows, element 3 in partition 2 updates shadow element 9 in partition 0.

Linear Solver Synchronization

Proper parallelization of the linear solver is essential to the overall performance of the parallel code. In the method we follow, each processor assembles its own matrix and solves it locally while accounting for the changes of the variables at the interface through synchronization.

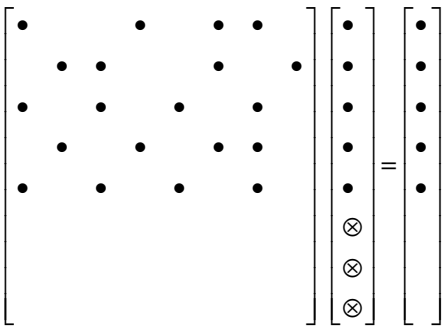

Figure 5: Linear system for a partition

A typical linear system associated with a given partition has the form shown in Figure 5. If synchronization is performed only on the fine grid, the partition coupling will resemble an SOR iteration and the efficiency and performance of the iterative solver will deteriorate with the number of processors [1]. This is due to the fact that shadow elements are treated as local Dirichlet boundary conditions.

Multigrid Synchronization

For the parallel AMG (pAMG) the agglomeration takes place in each partition independently but the same number of levels is enforced across all partitions. At each level, partitions exchange information at the interface regarding the agglomerated shadow and sender elements.

For the pAMG each level that is visited by the solver performs a synchronization of values at the interface of that level. This method ensures that we have the same overall residual at the end of each outer iteration. With this in mind, the parallel AMG method is most effective for improving the efficiency of the parallel code. There are various scenarios with which to perform the synchronization. For example, one can perform the synchronization while restricting. Another method is to synchronize at the finest and coarsest levels. The preferred method is to synchronize at all levels to make sure that we have the same residual, at least, in the outer iterations.

TEST PROBLEM

To test our parallelization, we choose to solve a nonlinear diffusion problem governed by

$$\nabla \cdot (\Gamma \nabla \phi) = 0 \qquad \Gamma = \phi^{0.1} \tag{8}$$

in the following domain

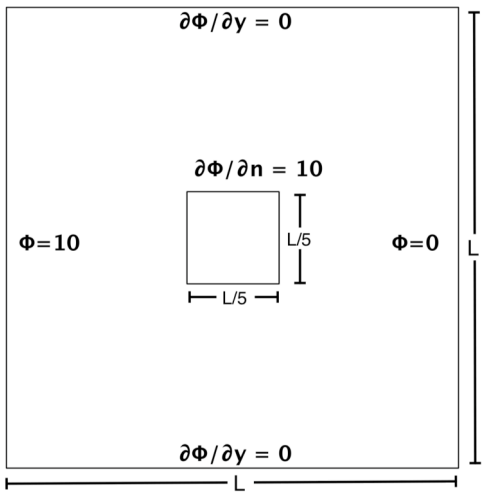

Figure 6: Test problem geometry

where the various boundary conditions are shown in Figure 6. Two unstructured meshes were used for the geometric discretization; 148,763 – 298,154 elements. The runs were performed on a cluster of ten G5 dual processor machines running at 2.0 GHz each with a memory of 512 MB per node. The interconnection network is a 100-MBits Ethernet switch. The runs were made until the residual was reduced to a value of 10^{-4} where the residual is defined as

$$r_P = \frac{b_P - \sum_{i=NB(P)} a_{Ni}\phi_{Ni}}{a_P} \quad (9)$$

Speedup

The speedup, for a given number of processors P, is defined as the ratio of the solution time of the sequential algorithm to the solution time on P processors

$$SP = \frac{T_s}{T_P} \quad (10)$$

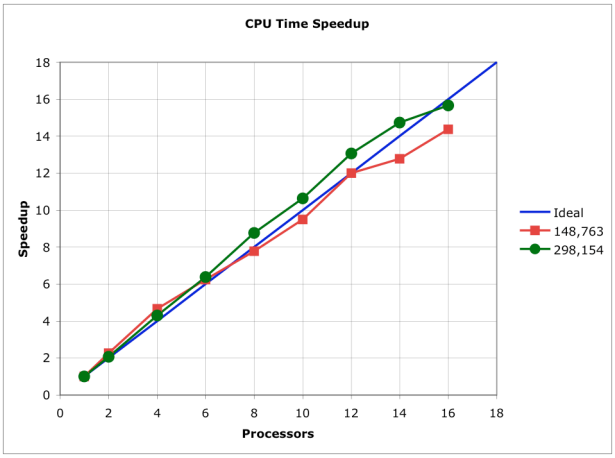

Figure 7: Speedup

Figure 7 shows the speedup obtained for the two meshes using up to 16 processors. For the 148,763 mesh, the behavior is superlinear up to 6 processors while it is almost linear up to 12 processors. The scalability starts to deteriorate as of 14 processors where the ratio of the number of shadow elements to the number of core elements exceeds 0.02. For the 298,154 mesh, it is noticed that the behavior is superlinear up to 16 processors at which the ratio of the number of shadow elements to the number of core elements remains almost the same but the parallel cost increases as shown in Figure 10.

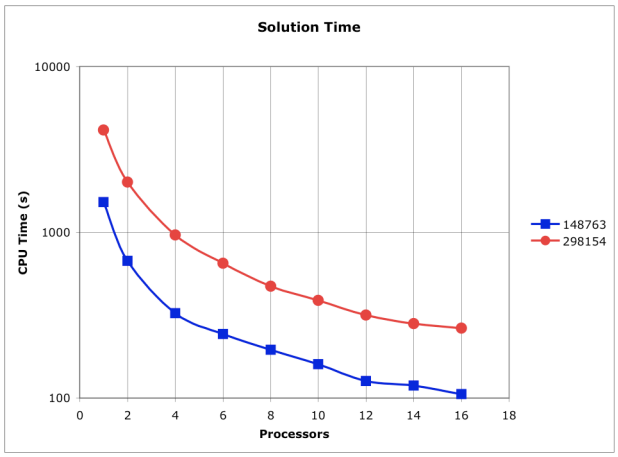

Figure 8: Solution time

The reason why large meshes have better scalability is that the ratio of the number of shadow elements to the number of core elements is less than that for smaller meshes; which accounts for the improved performance. The computational cost increases for the larger mesh while the parallel cost remains slightly the same. Figure 8 shows the solution time for as a function of the number of processors used.

Communication and Computational Costs

The communication and computational costs are directly related to the number of shadow elements and the number of core elements since the computation takes place over the core elements and the communication takes place over the shadow elements.

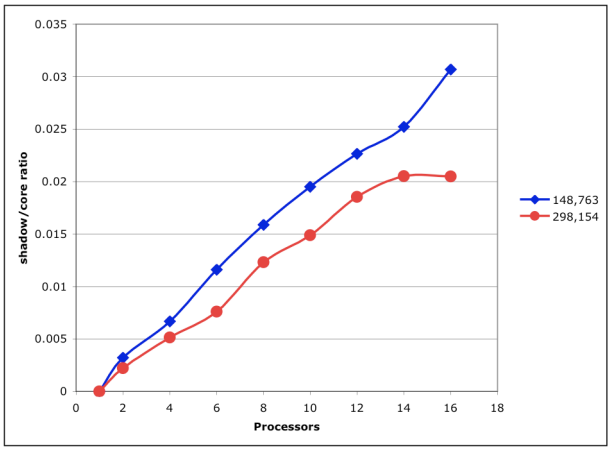

Figure 9: Shadow/Core elements ratio

Figure 9 shows the ratio of the number of shadow elements to the number of core elements versus the number of processors. It is noticed that for larger meshes, this ratio decreases which in turn favors better scalability.

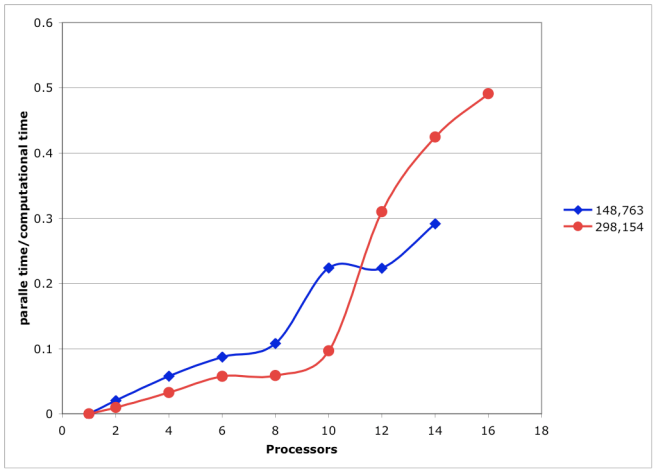

Figure 10: Parallel/Computational time ratio

Figure 10 shows the ratio of the parallel time to the computational time versus the number of processors. The parallel time is defined as the time spent on parallel communication while the computational time is defined as the time spent on computational operations only. As can be seen from the figure above, the parallel/computation ratio increases as the number of processors increases which imposes an upper limit for the scalability of a given problem. As long as this ratio is maintained below 0.25, scalability is well behaved.

Figure 11 shows the ratio of parallel time to computational time versus the ratio of the number of shadow to

the number of core elements. There is a direct correlation between both of these ratios and it is noticed that a sudden jump occurs for a shadow/core ratio of 0.02 approximately.

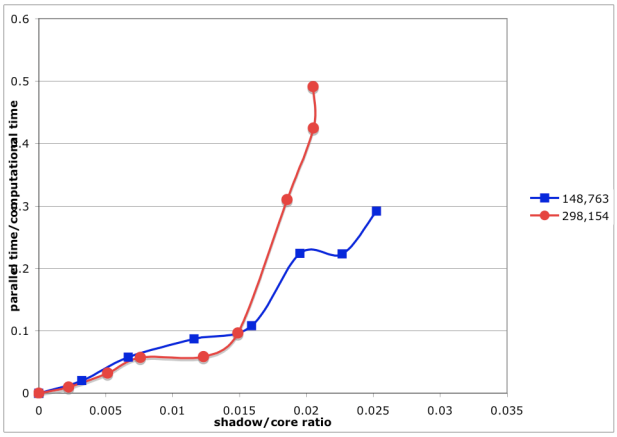

Figure 11: Parallel/Computational time versus shadow/core elements ratio

This can be attributed to the fact that as the number of partitions increases, the number of core elements significantly decreases which in turn raises the shadow/core elements ratio.

Conclusion

In this paper we have presented the performance of a pAMG solver used in a Finite Volume cell-centered unstructured CFD code. The performance of the parallel solver was compared to that of the sequential solver and it was shown that a substantial improvement in the solution time was gained as long as the computational load is well distributed amongst processors and the ratio of parallel time to the computational time does not exceed 0.25 which is equivalent to a value of 0.02 for the ratio of shadow to core elements. The solver scalability was shown to exceed the theoretical limit in some cases, which is encountered in practice, as the usage of the cache memory is more efficient for small problems. The solver behaves extremely well for a wide range of processors. Future work will include studies over a wider range of problem sizes, parallelization of the pressure correction algorithm for the full solution of fluid flow problems and implementation of other AMG fixed cycles such as the V and W cycles.

REFERENCES

[1] V. Dolean and S. Lanteri, "Parallel multigrid methods for the calculation of unsteady flows on unstructured grids: algorithmic aspects and parallel performances on clusters of PCs," *Parallel Computing,* vol. 30, pp. 503-525, 4. 2004.

[2] **http://www.top500.org**

[3] J.H. Ferziger and M. Peric, *Computational methods for fluid dynamics,* Berlin ; New York: Springer, 1999.

[4] J. Y. Murthy and S. R. Mathur, *Numerical Methods in Heat, Mass, and Momentum Transfer,* Purdue University: Draft Notes, 1998.

[5] G. Karypis and V. Kumar, "Multilevelk-way Partitioning Scheme for Irregular Graphs," *Journal of Parallel and Distributed Computing,* vol. 48, pp. 96-129, 1/10. 1998.