Performance of Algebraic Multigrid for Parallelized Finite Element DNS/LES Solvers

Gregory James Larson
Brigham Young University - Provo

PERFORMANCE OF ALGEBRAIC MULTIGRID FOR PARALLELIZED FINITE ELEMENT DNS/LES SOLVERS

by

Gregory J. Larson

A thesis submitted to the faculty of

Brigham Young University

in partial fulfillment of the requirements for the degree of

Master of Science

Department of Mechanical Engineering

Brigham Young University

December 2006

BRIGHAM YOUNG UNIVERSITY

GRADUATE COMMITTEE APPROVAL

of a thesis submitted by

Gregory J. Larson

This thesis has been read by each member of the following graduate committee and by majority vote has been found to be satisfactory.

Date	Deryl O. Snyder, Chair
Date	Scott L. Thomson
Date	Jeffrey P. Bons

ABSTRACT

PERFORMANCE OF ALGEBRAIC MULTIGRID FOR PARALLELIZED FINITE ELEMENT DNS/LES SOLVERS

Gregory J. Larson

Department of Mechanical Engineering

Master of Science

The implementation of a hybrid spectral/finite-element discretization on the unsteady, incompressible, Navier-Stokes equations with a semi-implicit time-stepping method, an explicit treatment of the advective terms, and an implicit treatment of the pressure and viscous terms leads to an algorithm capable of calculating 3D flows over complex 2D geometries. This also results in multiple Fourier mode linear systems which must be solved at every timestep, which naturally leads to two parallelization approaches: Fourier space partitioning, where each processor individually and simultaneously solves a linear system, and physical space partitioning, where all processors collectively solve each linear system, sequentially advancing through Fourier modes.

These two parallelization approaches are compared based upon computational cost using multiple solvers: direct sparse LU, smoothed aggregation AMG, and single-level ILUT preconditioned GMRES; and on two supercomputers of different memory architecture (distributed and shared memory). This study revealed Fourier space

partitioning outperforms physical space partitioning in all problems analyzed, and scales more efficiently as well. These differences were more dramatic on the distributed memory platform than the shared memory platform.

Another study compares the previously mentioned solvers along with one additional solver, pointwise AMG, in Fourier space partitioning without parallelization to better understand computational scaling for problems with large meshes. It was found that the direct sparse LU solver performed well in terms of computational time, scaled linearly, but had very high memory usage which scaled in a super-linear manner. The single-level ILUT preconditioned GMRES solver required the least amount of memory, which also scaled linearly, but only had acceptable performance in terms of computational time for coarse meshes. Both AMG methods scaled linearly in both memory usage and time, and were comparable to the direct sparse LU solver in terms of computational time.

The results of these studies are particularly useful for implementation of this algorithm on challenging and complex flows, especially direct numerical and large-eddy simulations. Reducing computational cost allows the analysis and understanding of more flows of practical interest.

ACKNOWLEDGMENTS

First I would like to thank my wife, Hilary, and family for their love and support throughout my education and the undertaking of this project. They continue to be a sustaining force in my life and a source of motivation.

I would also like to thank my advisor, Dr. Deryl Snyder, for his many hours of guidance, assistance, and teaching, with me individually.

I also acknowledge David Vanden Abeele, at the von Karman Institute for Fluid Dynamics, and Tanja Clees, at the Fraunhofer Institute, for their contributions in the first study, contained in Chapter 2, as well as Michael Heroux, at Sandia National Laboratories, for his assistance implementing the Trilinos software.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

In performing computational fluid dynamics (CFD), a significant amount of the computational cost is devoted to solving systems of linear equations. Many direct and iterative methods have been developed for more efficient calculation of linear systems in general as well as CFD systems in particular. Algebraic Multigrid (AMG) is a relatively new method which has been shown to dramatically improve the computational cost of solving linear systems, but has not yet been widely applied to CFD codes designed for Direct Numerical Simulation (DNS)/Large-Eddy Simulation (LES) applications. This thesis contains the method and results of implementing an AMG solver inside a parallelized DNS/LES code.

The DNS/LES CFD code implemented is "SFELES", which is based on a Spectral/Finite-Element (FE) algorithm solving the unsteady, incompressible, 3D Navier-Stokes equations on complex 2D geometries. The combined Spectral/FE method allows for two parallelization approaches; these approaches are implemented and their performance is analyzed.

1.1 Documentation Format

The main body of this thesis consists of two stand-alone papers. Chapter 2 compares various direct and iterative solvers, including "SuperLU", GMRES, and AMG, based upon computational cost. This has direct application to partitioning in Fourier space, which is one parallelization method. Chapter 3 focuses on comparing the two parallelization techniques. Because these chapters are intended to be stand-alone documents, some repetition exists. While much of the background is discussed in these two chapters (and the reader will be referred to them), the majority of this chapter is devoted to an introductory background discussion.

1.2 Linear Systems

Although the governing equations for fluid flow are a system of coupled non-linear partial differential equations, through numerical discretization an approximate solution can be obtained via the solution of one or many systems of linear equations. The basic form of these systems is

$$Ax = b, \tag{1.1}$$

where A is the stiffness matrix, x is the vector of unknowns, and b is the "force" or right-hand side vector. In order to solve for the unknown quantities, the inverse equation,

$$x = A^{-1}b \tag{1.2}$$

must be calculated. In general, this is the most costly phase of any CFD algorithm.

For typical discretizations used in CFD (finite element, finite volume, or finite difference), A is large and sparse. In addition, for certain discretizations A is symmetric or nearly symmetric. While there are numerous methods for solving linear systems, only methods related to those applied in this study are discussed here, which include both direct and iterative methods.

1.3 Direct Solvers

One method for directly solving linear systems is lower-upper (LU) decomposition. LU decomposition factorizes A into two triangular matrices, L and U:

$$A = LU. \tag{1.3}$$

This factorization may be performed using Gaussian elimination, which subtracts the multiple of one matrix row from another matrix row in order to eliminate values in the matrix. Gaussian elimination is equivalent to premultiplying A by a sequence of lower-triangular matrices,

$$L_{m-1}L_{m-2}\cdots L_2L_1A = U \tag{1.4}$$

where

$$L_{m-1}L_{m-2}\cdots L_2L_1 = L^{-1}. \tag{1.5}$$

By setting

$$L = L_1^{-1}L_2^{-1}\cdots L_{m-2}^{-1}L_{m-1}^{-1} \tag{1.6}$$

Eq. 1.3 is obtained.

Applying Eq. (1.3) to Eq. (1.1) yields:

$$LUx = b. \tag{1.7}$$

By creating a new vector y according to

$$y = Ux \tag{1.8}$$

and substituting into Eq. (1.7), gives

$$Ly = b. \tag{1.9}$$

Since L is lower triangular, the first value of the y-vector can easily be obtained. Whereafter, each successive value in the y-vector can be obtained by substituting in the previous values; this is called forward-substitution. Then, returning to Eq. (1.8), since U is upper triangular, x can easily be obtained through back-substitution.

Pivoting of the equations in the linear system is required if there is a zero on the diagonal. This is also helpful in reducing numerical errors, especially when there are values on the diagonal with small magnitudes relative to other matrix values.

Although the A matrix may be sparsely populated, the L and U triangular matrices are often dense, leading to high memory usage. The calculation of the LU decomposition is also computationally costly.

Demmel et al. [1], created an efficient LU decomposition algorithm for sparse matrices using a supernode-panel technique, called "SuperLU". Supernodes are created by grouping matrix columns together with the same nonzero structure below the diagonal, which often arise from discretized partial differential equations. This allows utilization of optimized vector-vector and matrix-vector routines from the BLAS library. The resulting code out-performed its predecessors on most test matrices, averaging over twice as fast [1]. SuperLU was originally written for a single processor, but another version called "SuperLU DIST" [2] was written for parallel processing on distributed memory machines.

1.4 Iterative Solvers

Linear systems can also be solved in an iterative manner. Iterative solvers avoid the computational cost of finding the inverse of a matrix and the memory cost of storing it. Iterative methods inherently require an initial guess for x. The better the initial guess, the quicker the solution converges.

1.4.1 Jacobi Method

The most basic iterative method is the Jacobi method. The Jacobi method solves for a single unknown at a time based upon the current values of the other unknowns. The Jacobi algorithm is written

$$x_i^{(k)} = \frac{b_i - \sum_{j \neq i} A_{i,j} x_j^{(k-1)}}{A_{i,i}}, \tag{1.10}$$

where i references the i^{th} equation and k the iteration number. The major advantage of the Jacobi method is its simplicity. It is easy to program and a computer can quickly perform many iterations.

One situation which causes the Jacobi method to fail is a zero on the diagonal of A, which results in an undefined value. Another problem arises if the initial guess has low frequency errors. Any initial value of x contains error. Since any discrete function can be represented as a truncated Fourier series, the error can be thought of as a range of error frequencies (according to the discretization points). Jacobi iterations quickly damp out the high frequency errors, but the low frequency errors take much longer to damp out. The error damping depending on error frequency is common to not just the Jacobi method but to many iterative methods [3].

1.4.2 Krylov Subspaces

Direct methods for solving Eq. (1.1) with a square matrix, A, of full rank m (where m is the number of rows and columns), requires solving the problem in all m dimensions. Many iterative methods simplify the problem by only solving the problem in n dimensions, where n is much less than m. Using Krylov subspaces, the solution successively incorporates the next largest Krylov dimension. The next Krylov dimension is determined by pre-multiplying the previous Krylov dimension by the matrix A, where the first Krylov dimension is the right-hand side vector, b. Therefore, the Krylov sequence of dimensions is b, Ab, A^2b, A^3b, etc. The n^{th} Krylov subspace is the space that the first n Krylov dimensions span [4, 5].

1.4.3 Generalized Minimal Residual Method (GMRES)

One iterative method which utilizes Krylov subspaces is the generalized minimal residual method (GMRES). GMRES is an iterative method that finds a vector in the n dimensional Krylov subspace that minimizes Eq. (1.11), thus approximating the actual solution.

$$||Ax_n - b|| \tag{1.11}$$

The method GMRES uses is based upon the Arnoldi iterative method, an algorithm for locating eigenvalues. When attempting to locate eigenvalues, only similarity transformations preserve the eigenvalues. A similarity transformation is written

$$B = X^{-1}AX. \tag{1.12}$$

The Arnoldi method reduces (not completely) A to Hessenberg form through a unitary similarity transformation. A Hessenberg matrix is one where all values below the diagonal are zero, except the one immediately below the diagonal. A full reduction of the A to Hessenberg would be

$$A = QHQ^* \qquad \text{or} \qquad AQ = QH \tag{1.13}$$

where Q is a unitary matrix, and Q^* is the conjugate transpose of Q. A unitary matrix satisfies

$$Q^*Q = I \tag{1.14}$$

where I is the identity matrix. The conjugate transpose of a matrix is obtained by transposing the matrix and taking the complex conjugate of every value. For a real matrix, the conjugate transpose is simply the transpose. The Arnoldi method only uses the first n columns of Q and therefore approximates the eigenvalues by iteratively creating a reduced Hessenberg matrix using

$$AQ_n = Q_{n+1}H_n. \tag{1.15}$$

As previously mentioned, GMRES attempts to minimize Eq. (1.11). One method would be to minimize

$$||AK_nc - b|| \tag{1.16}$$

where K_n is the n^{th} Krylov subspace, and then let

$$x_n = K_nc. \tag{1.17}$$

Unfortunately this method is numerically unstable.

The method actually implemented by GMRES uses Arnoldi iteration to create a sequence of Krylov matrices Q_n, where the columns span the successive Krylov subspaces, and minimizes

$$\|AQ_n y - b\|. \tag{1.18}$$

After y is found, x is calculated from

$$x_n = Q_n y. \tag{1.19}$$

The actual GMRES problem can be reduced further. Applying Eq. (1.15) of the Arnoldi method, Eq. (1.18) becomes

$$\|Q_{n+1} H_n y - b\|. \tag{1.20}$$

Since both vectors in the norm are in the column space of Q_{n+1}, they can be premultiplied by Q^*_{n+1} without changing the norm, leading to

$$\|H_n y - Q^*_{n+1} b\|. \tag{1.21}$$

A further examination of the Arnoldi method and the construction of Q_n reveals

$$Q^*_{n+1} b = \|b\| e_1 \tag{1.22}$$

where e_1 is the first column of the identity matrix. Applying Eq. (1.22) to Eq. (1.21) reveals the final reduced form of the GMRES problem:

$$\|H_n y - \|b\| e_1\|, \tag{1.23}$$

which is minimized to obtain y.

The residual does not need to be calculated directly, but is actually imbedded within the iteration method [5, 6]. The details are omitted here and the interested reader is referred to Ref. 5, 6.

1.4.4 BiConjugate Gradient (BiCG)

Another linear solving technique for non-symmetric matrices is the BiConjugate Gradient (BiCG) method. This technique, like GMRES, also stems from the Arnoldi method. Lanczos' method is essentially the Arnoldi method applied to a Hermitian matrix, and also finds eigenvalues. A Hermitian matrix is one for which

$$A = A^* \tag{1.24}$$

holds true.

When the Arnoldi method is applied to a Hermitian matrix, instead of H in Eq. (1.15) being upper Hessenberg, it is tridiagonal,

$$A = QTQ^* \tag{1.25}$$

where Q is once again a unitary matrix. Just as GMRES uses a variation of the Arnoldi method to solve linear systems, the conjugate gradient uses a variation of the Lanczos' method. If A is not Hermitian, then the matrix cannot be reduced to a tridiagonal matrix through unitary similarity transformations. Either unitary similarity transformations can reduce it to an upper Hessenberg matrix or non-unitary similarity transformations can reduce it to a tridiagonal matrix. While GMRES chooses to reduce the matrix to upper Hessenberg form, biorthogonalization methods (including BiCG) give up the unitary matrix in favor of the tridiagonal matrix, according to

$$A = VTV^{-1} \tag{1.26}$$

where V is not unitary. The "biorthogonalization" term comes from the fact that while the columns of V are not orthogonal to each other, they are orthogonal to $(V^{-1})^*$.

If W is defined by

$$W = (V^{-1})^*, \tag{1.27}$$

setting the first column of V as b, and choosing the first column of W to satisfy

$$w_1^T v_1 = 1 \tag{1.28}$$

where the subscript 1 indicates the first column of the matrix, the BiCG method is obtained.

In GMRES, the main idea is to minimize the residual while constraining the residual to be perpendicular to all previous Krylov subspaces. For BiCG, x_n is chosen in the same subspace, but the choice requires the residual to be perpendicular to the following subspace

$$w_1, A^* w_1, (A^*)^2 w_1, \cdots, (A^*)^{n-1} w \tag{1.29}$$

which does not minimize the residual nor the number of iterations required.

The advantage of BiCG over GMRES is the three term-recurrences [5,7], which occur because a tridiagonal matrix has only three terms in any given column, and means the work per step remains constant, whereas each iteration of GMRES introduces another term into the calculations. However, GMRES has two advantages over BiCG. First, each iteration's residual of GMRES is guaranteed to be reduced while BiCG often has erratic and slower convergence. The other disadvantage is that it requires multiplication by A^* and A, which can be computationally costly [5].

The conjugate gradient squared method (CGS) is a variation of the BiCG method, but eliminates the necessity of using the Hermitian of A. This is accomplished by redefining the residual. However, this squaring can lead to error growth due to rounding errors. The stabilized BiConjugate Gradient method (BiCG-stab) is a further variation of CGS and once again redefines the residual. The result is an algorithm that is stable and does not require using the transpose of A. See Ref. 7 for more details on CGS and BiCG-stab.

1.5 Multigrid

As previously mentioned, iterative methods converge faster with a better initial guess. Multigrid is a solving technique for discretized problems where coarse grids are created and solved in order to obtain better "guesses" for the fine grid solution.

1.5.1 Geometric Multigrid

As discussed in Section 1.4.1, since the low frequency errors are damped out slowly, if instead of using all the nodes in a given mesh for the FE problem, by just using half of the nodes the problem has been coarsened, i.e. the mesh is essentially less refined. Then, on the coarser mesh, the lower frequency error modes will become higher frequency error modes, relative to the mesh. By iterating on this coarse mesh, those previously low frequency errors, now high frequency errors, will be damped out faster than on the finer mesh. Also, since the problem is smaller, a single iteration is quicker. The solution obtained on the coarse grid can then be used for the initial solution for the fine mesh. As a result, multigrid effectively reduces the errors at all length (frequency) scales uniformly.

Figure 1.1 shows an example of how close nodes may be grouped together, or coarsened, based upon geometry. Once the coarser grid is created, more coarsening can be done until it is coarse enough for quick calculations of the unknowns. After obtaining the unknowns on the coarsest grid, those values can be used as the initial guess for all the fine nodes represented by a coarse node. This process is then repeated continuing from the coarser to finer grids until the solution of the original grid is determined.

The limit on coarsening does not need to be confined to one coarser level. Many levels of coarsening can be performed following the same principle. Since multiple meshes, or grids, are created in this process, it is called "multigrid". This process of coarsening based upon the mesh structure will later be referred to as geometric multigrid.

Because the coarsening process requires the mesh to be examined and nodes to be combined, the algorithm needs to be able to read the mesh and make decisions

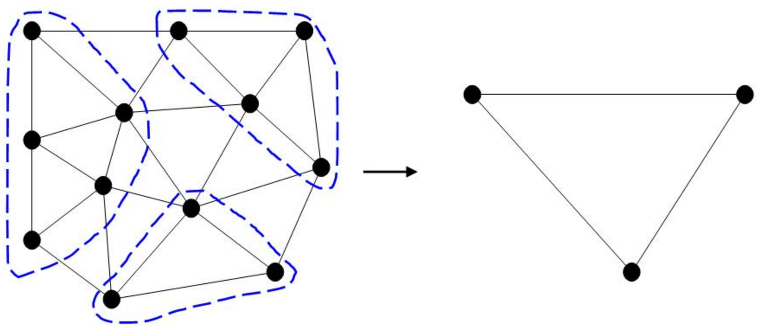


Figure 1.1: Coarsening Using Geometric Multigrid

on how to coarsen. This is therefore not robust and the coarsening process can fail, especially when applied to complex domains [8].

1.5.2 Algebraic Multigrid (AMG)

In algebraic multigrid (AMG), the matrix structure is examined instead of the mesh to avoid the failure that can occur in geometric multigrid, making it a more robust algorithm. In general, off-diagonal terms in the matrix determine the coupling between different unknowns. If the coefficient is large (relative to other coefficients on the row, especially the diagonal term) then there is a high correlation between the two unknowns.

For FEM problems, there can only be off-diagonal terms if the unknowns correspond to the same node for a different coupled unknown or two different nodes sharing an element.

The basic mathematical method is to convert the original equation,

$$A^f x^f = b^f \tag{1.30}$$

(where f refers to the fine grid) to a coarse linear set of equations,

$$A^c x^c = b^c \tag{1.31}$$

(where c refers to the coarse grid) using a coarsening n x m matrix, K_f^c, which follows

$$A^c = K_f^c A^f (K_f^c)^T. \tag{1.32}$$

Once x^c is found, it can be used to create a better guess for x^f, where typically only a few iterations are required to solve the problem down to a given tolerance.

One implementation method is to perform a few iterations on the fine grid problem, Eq. (1.30), in order to damp any high frequency errors on the fine grid. The fine grid residual is computed according to

$$r^f = b^f - A^f x^f \tag{1.33}$$

where r^f is the residual on the fine grid. The residual on the fine grid is then converted to the coarse grid using

$$r^c = K_f^c r^f \tag{1.34}$$

where r_c is the residual on the coarse grid. The error on the coarse grid, e^c, is computed according to

$$A^c e^c = r^c \tag{1.35}$$

where the error is defined by

$$e = x^* - x \tag{1.36}$$

where x^* is the exact solution to Eq. (1.1). The error on the coarse grid is then solved for and interpolated back to the fine grid using

$$e^f = K_c^f e^c \tag{1.37}$$

which is used to adjust the fine solution according to

$$x^f = x^f + e^f. \tag{1.38}$$

The fine solution is obtained by iterating on Eq. (1.30) until convergence is reached [3, 9].

The same idea is extended to a various number of grids, where iterations are performed on each level. To solve for the error on the coarsest grid, a direct solver is often used if the problem is reduced sufficiently. There are also different variations for cycling between the fine and coarser grids [3].

While an optimized geometric multigrid algorithm may be more efficient than algebraic multigrid, it lacks the robustness of algebraic multigrid [8]. Sections 2.3.3 and 3.4.2 describe the actual AMG algorithm implemented in this research.

1.6 Preconditioning

The most computationally expensive piece of solving Eq. (1.1) is finding A^{-1}. However, it is not important to actually find A^{-1}, since it is $A^{-1}b$ that is really desired. In the solving methods previously discussed, none focused on actually finding A^{-1}. $A^{-1}b$ can be indirectly approximated using a preconditioner according to

$$MAx = Mb \tag{1.39}$$

where M approximates A^{-1}. This yields the approximation

$$MA \approx I \tag{1.40}$$

where I is the identity matrix.

Since the A matrix in the SFELES algorithm is constant for all time steps, it only requires finding A^{-1} at the initial time step and successively applying it to a different b-vector at each future time step. Thus, the time required for finding this inverse is less important. The computational cost of memory usage is still important however. As mentioned earlier, while A is sparse, A^{-1} may not necessarily be sparse. Since A^{-1} does not actually need to be found, an approximate (sparse) matrix may be used instead to reduce memory usage.

Precision plays an important role in the accuracy of the solution. For any problem, the condition number of the A matrix and the norm of the b vector may cause inaccuracy of the solution due to machine precision of the computer [5].

In the linear system

$$Ax = b \tag{1.41}$$

if b is represented exactly, then the resulting relative error of x as a result of $A^{-1}b$ is given by

$$\frac{\|\delta x\|}{\|x\|} / \frac{\|\delta A\|}{\|A\|} \leq \kappa(A) \tag{1.42}$$

where $\kappa(A)$ is the condition number of the A matrix, defined as

$$\kappa(A) = \|A\|\|A^{-1}\|. \tag{1.43}$$

This means that x will have α less digits of accuracy than the machine precision, ϵ, where α is

$$\alpha = \log_{10} \kappa(A) \tag{1.44}$$

and the machine precision is defined as the smallest number such that

$$1 + \epsilon > 1. \tag{1.45}$$

For a 32-bit computer this is about 2 x 10^{-16} [5]. Therefore, the solution obtained even from a correct algorithm could be incorrect due to machine accuracy. Since the condition number of A naturally increases with the size of A, this becomes more of a problem with large linear systems, which is generally the case with DNS/LES CFD problems. While preconditioning may be used to find an approximate inverse of A, it can also be utilized to reduce the condition number of A in order to improve convergence of any solution method.

One method derived from the Jacobi method for choosing M is a diagonal matrix where the coefficients follow

$$M_{i,i} = \frac{1}{A_{i,i}}. \tag{1.46}$$

While this method is a poor choice for representing the approximate inverse of the matrix, it does scale all the equations, thus reducing the condition number.

1.7 Spectral/Finite-Element Large Eddy Simulation (SFELES) Algorithm

SFELES, a hybrid Spectral / FE algorithm, solves the 3D unsteady, incompressible, Navier-Stokes equations for 2D geometries [10, 11]. A representative mesh and flow past a circular cylinder are shown in Figure 1.2. This figure also demonstrates the implementation of a 2D unstructured mesh using linear triangular elements for discretization of planes of constant z. The transverse direction is discretized using a truncated Fourier series, which assumes periodicity in the z direction. While 3D flows can be calculated, only a single 2D mesh needs to be stored, thereby reducing memory requirements.

This algorithm uses a semi-implicit time-stepping method: an explicit treatment of the advective terms and an implicit treatment of the pressure and viscous terms. The semi-implicit time-stepping method requires small time steps due to limitations in the Courant number. However, this is already inherently required for DNS/LES solvers. Sections 2.2 and 3.2 contain more details on this algorithm.

1.8 Parallelization Techniques

The combination of spectral decomposition in the transverse direction and explicit treatment of the convective terms essentially converts the 3D problem into a system of 2D problems (one for each Fourier mode) that can be solved independently at each timestep. These methodologies naturally lead to two parallelization techniques when solved on multiple processors. The first is where multiple Fourier modes are solved simultaneously and independently on the multiple processors; and the second

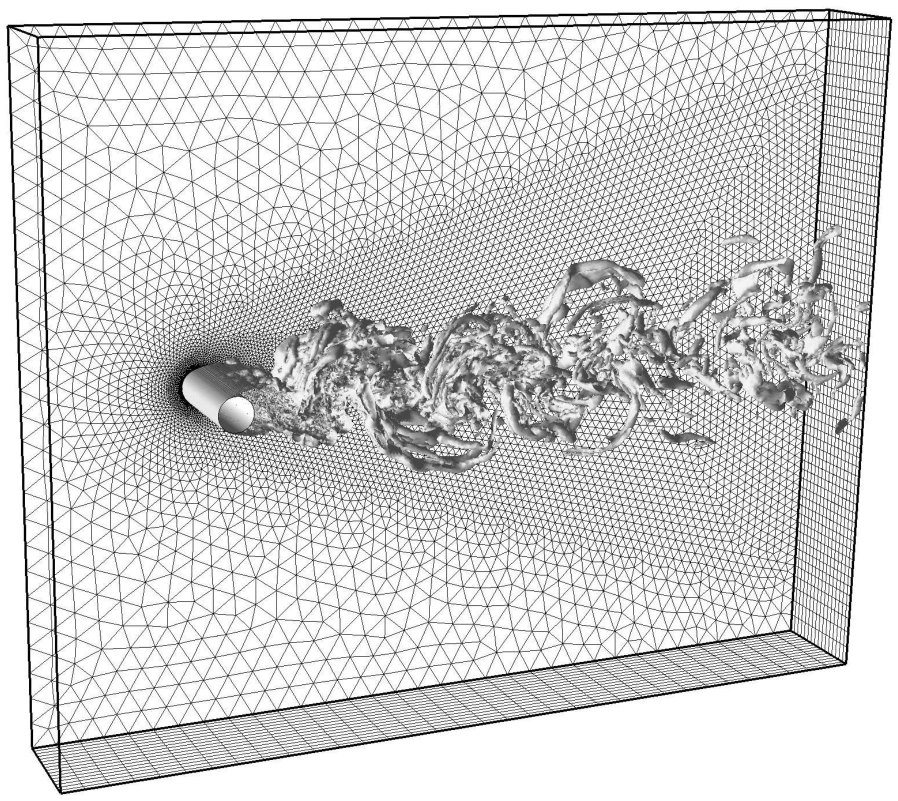


Figure 1.2: Representative hybrid spectral/finite-element mesh for circular cylinder flow, where linear finite-elements are employed in the two-dimensional cross planes, while a truncated Fourier series is employed in the transverse direction. Shown are isosurfaces of vorticity magnitude at Reynolds number 8 000.

where all processors work together to solve one system and sequentially work through all linear systems.

These two different parallelization techniques are analyzed and compared based upon computational cost (memory usage and solving time). By reducing memory usage, larger problems can be analyzed with the same processor capability. Reducing the solving time allows for quicker computation, which is also a limiting resource. However, improving one while worsening the other may or may not be an overall improvement and depends upon the problem at hand.

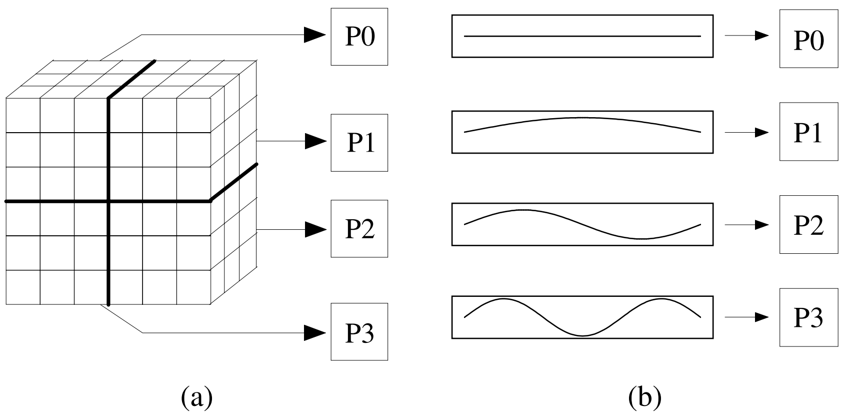


Figure 1.3: Illustration of the parallel partitioning schemes employed in SFELES (P_0 through P_3 represent different processors). (a) Physical space partitioning and (b) Fourier space partitioning.

1.8.1 Partitioning in Fourier Space

"Partitioning in Fourier space" parallelization is obtained by each Fourier mode's linear system being solved by a single independent processor. Multiple Fourier modes are solved simultaneously by multiple processors, and if sufficient processors are utilized, all Fourier modes can be solved simultaneously. This method implements a serial version of a modular solver in conjunction with SFELES, where multiple instances of the solver are called simultaneously (one for each Fourier mode). Figure 1.3 visualizes the differences between Fourier space and physical space partitioning.

Various serial solvers are compared, including direct and iterative solvers. The goal of analyzing various solvers is to improve computational cost by decreasing the memory usage and linear system solution time. In this method, the maximum number of processors that can be utilized is half the number of active Fourier modes.

Inefficiencies in this method are mainly due to the linear systems of different Fourier modes being solved in different amounts of time, causing all other processors to wait until the last Fourier mode has been completely solved. In addition, two large communication steps are required before and after the linear system solve.

1.8.2 Partitioning in Physical Space

The second type of parallelization that was investigated, "partitioning in physical space", is where all processors work together to solve one linear system and sequentially progress through the linear systems corresponding to each Fourier mode. In this method the linear system solver is parallelized and only one instance of the solver is active at any given time.

In this method, ownership of the nodes continues through the solving phase, while when partitioning in Fourier space, node ownership is ignored when solving the Fourier modes and values are communicated between the solving processor and owning processor.

This, "partitioning in physical space", method not only reduces the small overhead used in transferring the information from real to Fourier space partitions, but more importantly, the number of processors that can be used in solving the problem is not limited. Logically there exists a break point where more processors would become slower since communication between the processors in the solving phase increases. The ideal number of processors should be determined based upon overall computational speed and efficiency. The disadvantage with this method is that communication is required during the solving phase of the linear systems.

Multiple parallel solvers are analyzed and compared to the equivalent serial solver for both parallelization methods. Comparisons between computational efficiency and scalability are analyzed for both parallelization methods, emphasizing the AMG solver.

1.9 Contributions to State of the Art

Through this research, understanding of the performance of AMG, in terms of memory usage and solving time, on semi-implicit formulations of the Navier-Stokes equations is better understood. An understanding of performance of parallelization based upon "partitioning in Fourier space" versus "partitioning in physical space" has been realized, as well as knowledge regarding the scalability of both methods.

Through improvements shown by the AMG implementation for memory usage, analysis of more complicated flows using DNS/LES methods can be conducted.

There are also three main contributions made to the in-house SFELES code itself. First, the SFELES code has been linked to the Trilinos software package created by Sandia National Laboratories, enabling the implementation of an AMG solver along with preconditioning. Second, the original version of SFELES combined the 0^{th} and $N/2^{th}$ Fourier mode matrices into one linear system. This does not affect convergence of direct solvers, but does affect the convergence of iterative solvers. By splitting these apart it requires the calculation of two linear systems of $m/2$ unknowns instead of one system of m unknowns, which reduces the computational cost even for direct methods. The third contribution is allowing partitioning in physical space using a parallelized linear system solver. These contributions create more options for increased computational performance.

Chapter 2

Performance of AMG for DNS/LES Formulations

2.1 Introduction

In the past, applications of Computational Fluid Dynamics (CFD) to turbulent flows of practical interest have usually been restricted to seeking steady solutions to the Reynolds-Averaged Navier-Stokes (RANS) equations. Nowadays, such simulations can be performed even for realistic geometries in three-dimensions using commercial CFD solvers on relatively modest computers. This fact, combined with the sudden availability of large clusters of affordable personal computers, now motivates the scientific and engineering community to attempt much more challenging unsteady flow simulations on complex geometries.

In direct numerical simulations (DNS) of turbulent flows, one attempts to fully resolve all turbulence length scales and frequencies in the flow by solving the unsteady Navier-Stokes equations on a highly refined mesh over a large number of small time steps. For contemporary DNS, meshes may contain tens of millions of nodes, and the number of time steps to be taken may be on the order of one million. An often used and more affordable approach is Large Eddy Simulation (LES), in which only the larger eddies are resolved in space and time and small-scale turbulence is modeled in a statistical manner. However, both LES and DNS require state-of-the-art numerical methods and supercomputing facilities.

For the particular case of incompressible turbulent flows, DNS/LES solvers often use semi-implicit time stepping methods in which the advective terms are treated explicitly, while the pressure (elliptic) and viscous (parabolic) terms are treated implicitly. Such approaches are particularly appropriate for DNS/LES because the smallest spatial scales in the turbulent flow are usually comparable to the size of individual mesh elements. The physics therefore dictate that Courant numbers should be of order one to resolve these smallest features, so little can be gained by treating the convective terms implicitly. The pressure always requires an implicit treatment in time-accurate incompressible CFD methods. The viscous terms could be treated explicitly, but for certain types of flows, this can lead to time step restrictions corresponding to Courant numbers well below one.

These semi-implicit schemes require that at least one linear system be solved at each time step. The systems are characteristically large, sparse, positive definite or indefinite, symmetric (or nearly so), and constant in time. In addition, because highly anisotropic, stretched meshes are required near solid walls to resolve viscous boundary layers, these systems can be very stiff. Not surprisingly, the majority of the computational expense in DNS/LES is related to solving these systems.

This paper assesses the performance of algebraic multigrid (AMG) techniques—which are known to be well suited for this class of linear system—when employed in an incompressible DNS solver. The term AMG will be used to indicate both 'classical' algebraic multigrid [12] and related techniques involving automatic generation of coarse grid levels and operators, such as agglomeration [13] and smoothed aggregation multigrid [14]. Two well-known AMG packages are tested within a stabilized finite-element (FE) flow solver discussed in Section 2.2, and their performance is compared with a sparse direct method and with a one-level iterative method. The performance characteristics seen in this particular application can be expected to hold reasonably well for any linear system with the traits stated above.

AMG solvers possess several properties that make them desirable for use in solving DNS problems using unstructured meshes. First, multigrid methods in general reduce the errors at all length scales efficiently. This is especially important considering the fine meshes employed for DNS, and the fact that turbulent flow solutions typically contain significant features on a wide range of length scales. Second, CPU requirements should scale linearly with the number of unknowns, rather than super-linearly, as is the case for approaches that do not rely upon multigrid. Thirdly, the coarse grid matrices are derived automatically through algebraic methods, and much better coarsening can often be obtained on complex unstructured meshes. In fact, because this coarsening depends upon the level of coupling between the unknowns at the nodes, the flow field itself is accounted for in the process.

Several researchers have applied AMG techniques to solve linear problems arising from discretizations of the Navier-Stokes equations, with the early works relying heavily on the pioneering efforts of Ruge and Stüben [12].

Lonsdale [9] (1993) appears to have made some of the first efforts in this area, solving the three-dimensional, steady segregated Navier-Stokes equations on meshes containing as many as 11 000 nodes. Although success was achieved in terms of convergence behavior and scaling, a lack of robustness was encountered. This behavior appears to have been associated with the discretization technique (specifically, Rhie-Chow interpolation was noted to cause difficulties).

Shortly thereafter, Webster [15] (1994) employed a different discretization method and a simplified AMG algorithm for coupled variables. When applied to two-dimensional Navier-Stokes problems with up to approximately 1 700 nodes, greatly improved robustness was achieved.

Raw [16] (1996) used AMG to solve linear systems originating from a fully implicit finite-volume discretization of the steady Navier-Stokes equations. Good performance and robustness was demonstrated for a wide variety of flows (compressible, incompressible, reacting and non-reacting). Moreover, a convincing grid refinement study was presented for meshes with 25 000, 100 000 and 400 000 nodes.

Griebel, Neunhoeffer, and Regler [8] (1998) employed AMG to solve the unsteady Navier-Stokes equations in complex geometries. The AMG techniques were used to solve scalar linear problems arising from segregated pressure correction type strategies. Despite the fact that the convergence rates of standard multigrid methods usually deteriorate in complicated geometries and convection-dominated problems, it was found that AMG methods did not. However, it was noted that in order to optimize the performance of AMG solvers, coarsening parameters must be tuned. These are often dependent upon the mesh, and poorly understood.

Using AMG as a multi-level preconditioner for a Krylov subspace method, as is done in this study, has also shown improved robustness for the Navier-Stokes equations. The reason is that the Krylov method tends to reduce those eigenmodes that are not successfully smoothed by AMG [17]. Weiss, Maruszewki, and Smith [18] (1999) applied such an approach to two-dimensional steady problems as large as 80 000 cells, with stable results for both compressible and incompressible flows, while maintaining nearly linear scalability.

Mavriplis [19] (2003) employed an agglomeration multigrid method to solve the steady subsonic RANS equations on an unstructured mesh of 61 000 nodes, where the linear systems originated from a Newton-Raphson discretization of the discrete system of equations on the fine grid level. A comparison with a nonlinear agglomeration multigrid method indicates that, although linear methods are more expensive because a Jacobian matrix must be stored, they may be preferred because no time-consuming non-linear residual evaluations are needed.

The simulations presented in this paper differ from those discussed above mainly in the application to algorithms suited for DNS/LES applications. For instance, with the exception of a sub-study in Ref. [8], the above references all apply AMG to full convection-diffusion problems. As mentioned previously, in this study—as is common for DNS/LES—the convective terms are treated explicitly. Hence, only elliptic/parabolic terms enter in the system matrices; AMG methods have been originally developed for precisely this type of linear problems. From more specialized literature, it appears there is still no well-settled, optimal AMG or, more generally, algebraic multi-level method for all convection-dominated problems. Depending on the discretization, the use of stronger smoothers (of ILU-type), adjusted coarsening parameters, and special coarsening and/or interpolation schemes have been reported to be helpful (see, for instance Refs. [20, 21, 22, 23]).

The remainder of this paper is as follows. The stabilized FE formulation of the incompressible Navier-Stokes equations used to generate the linear systems for this study is first discussed. Next, the linear solvers under consideration are described, followed by a description of the test case employed. The performance of the linear

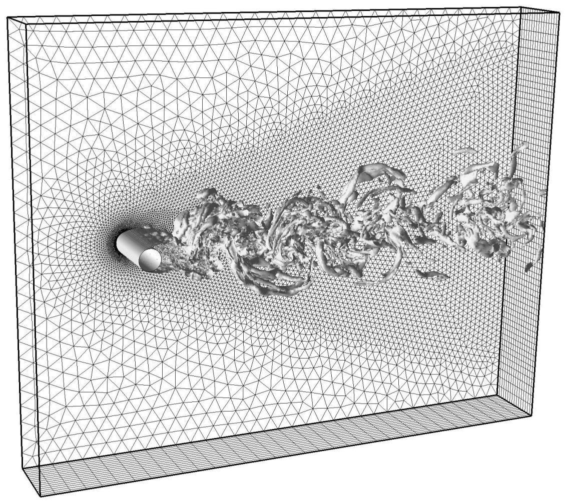


Figure 2.1: Representative hybrid spectral/finite-element mesh for circular cylinder flow, where linear finite-elements are employed in the two-dimensional cross planes, while a truncated Fourier series is employed in the transverse direction. Shown are isosurfaces of vorticity magnitude at Reynolds number 8 000.

solvers is then presented, along with a discussion of the results. Finally, a realistic three-dimensional calculation on a 50 million node mesh is considered in order to show the potential of AMG methods in a somewhat less academic context.

2.2 Stabilized Finite-Element Formulation

The DNS solver used for these tests is the hybrid Spectral/FE code SFELES [10, 11], which allows simulation of three-dimensional unsteady incompressible flows with planar or cylindrical geometries of arbitrary complexity. A representative hybrid mesh is illustrated in Figure 2.1, along with a representative flow field past a circular cylinder. The governing equations are discretized in cross planes (z = constant) using linear triangle finite-elements. The transverse direction is represented by means of a truncated Fourier series, which assumes periodicity in the z direction. Complex two-dimensional geometries can be accommodated through the unstructured in-plane mesh, while storage requirements are reduced because only a single two-dimensional mesh and associated overhead need be stored.

The dimensionless form of the incompressible Navier-Stokes equations is first discretized in time in a second-order accurate manner. An implicit Crank-Nicholson scheme is used for the pressure and viscous terms, while an explicit Adams-Bashforth method is used for the convective terms. The resulting expressions for the continuity and momentum residuals are set to zero at each time step:

$$\begin{aligned}
R^{n+1}_{Cont} &= \nabla \cdot (\vec{u}^{n+1}) = 0 \qquad (2.1)\\
\vec{R}^{n+1}_{Mom} &= \frac{1}{\Delta t}(\vec{u}^{n+1} - \vec{u}^{n}) + \nabla p^{n+\frac{1}{2}} - \frac{1}{2Re}\nabla^2(\vec{u}^{n+1} + \vec{u}^{n})\\
&\quad + \frac{3}{2}\nabla \cdot (\vec{u}^{n} \otimes \vec{u}^{n}) - \frac{1}{2}\nabla \cdot (\vec{u}^{n-1} \otimes \vec{u}^{n-1}) = 0 \qquad (2.2)
\end{aligned}$$

The solution at time t is represented in the FE plane using linear triangle elements for both the pressure and the velocity (P1/P1):

$$q(x, y, z, t) = \sum_{j=1}^{N_n} q_j(z, t)\, \phi_j(x, y) \qquad (2.3)$$

where q_j represents the value of the unknown at node j, N_n is the number of two-dimensional nodes, and ϕ_j represents the piecewise linear basis function associated with node j. The in-plane discretization at any node follows directly from the variational statement:

$$\int \phi_i R^{n+1}_{Cont} d\Omega - \int \phi_i \tilde{\nabla} \cdot (\tau_{pspg} \vec{R}^{n+1}_{Mom})\, \mathrm{d}\Omega = 0 \qquad (2.4)$$

$$\int \phi_i \vec{R}^{n+1}_{Mom} d\Omega + \int \phi_i \tau_{supg} (\vec{u} \cdot \tilde{\nabla}) \vec{R}^{n+1}_{Mom}\, \mathrm{d}\Omega = 0 \qquad (2.5)$$

where the operator $\tilde{\nabla}$ is the cross plane gradient, defined as $(\partial/\partial x,\ \partial/\partial y,\ 0)$. The second terms in the above equations are PSPG and SUPG stabilizations, respectively, that are added to remove compatibility and convective instabilities (see, for instance Ref. [24]).

The attentive reader may object that no such stabilization terms should be used in DNS, as they act as artificial subgrid scale stresses. This is certainly true for marginally resolved simulations, in which the PSPG and SUPG terms can be of order zero. However, for well-resolved simulations, such as presented here, the solution varies smoothly between elements. The numerical stabilization terms then become small second-order discretization errors. This is an area of active research (see, for instance, Ref. [25,26]); numerical experiments carried out using this solver for marginally-resolved DNS of channel flow show virtually no artificial dissipation due to the PSPG stabilization, and minimal dissipation due to the SUPG stabilization. Regardless of accuracy issues, the SUPG/PSPG formulation generates linear systems that are quite appropriate for testing AMG within the context of semi-implicit time-stepping techniques for the Navier-Stokes equations.

The discretized continuity and momentum equations may be cast in a matrix form, where all nonlinear contributions have been gathered on the right hand side as follows:

$$\left[M_1 + M_2\frac{\partial}{\partial z} + M_3\frac{\partial^2}{\partial z^2}\right]\begin{bmatrix}\Delta p(z)\\ \Delta\vec{u}(z)\end{bmatrix} = \begin{bmatrix}R_p^n(z)\\ \vec{R}_u^n(z)\end{bmatrix} \tag{2.6}$$

The precise form of the matrices M_1, M_2, and M_3 can be easily derived, and for purposes of brevity are not written here (see Ref. [11] for details).

In the transverse, z, direction a truncated Fourier series is employed, for which periodicity is assumed. A nodal unknown $q_{j,n} = q(x_j, y_j, z_n, t)$, is therefore represented as

$$q_{j,n}(t) = \frac{1}{N_f}\sum_{k=-N_f/2+1}^{N_f/2-1} \hat{q}_{j,k}(t)\exp\left(\frac{2\pi Ik}{N_f}n\right) \tag{2.7}$$

where $I = \sqrt{-1}$ and N_f is the number of modes used in the expansion. Doing so, the global three-dimensional linear problem in physical space, Eq. (2.6), is transformed into a series of two-dimensional linear problems for each Fourier mode k:

$$A_k\begin{bmatrix}\Delta\hat{p}_k\\ \Delta\hat{\vec{u}}_k\end{bmatrix} = \begin{bmatrix}\hat{R}_{p,k}^n\\ \hat{\vec{R}}_{u,k}^n\end{bmatrix} \tag{2.8}$$

where $A_k = [M_1 + M_2(2\pi Ik/L) - M_3(2\pi k/L)^2]$. For computational efficiency, the nonlinear terms, $R_p^n(z)$ and $\vec{R}_u^n(z)$, are computed in physical space and then transformed to Fourier space, resulting in a pseudo-spectral procedure.

Several key items directly affecting the linear system solution procedure should be noted:

- In practice, only $N_f/2$ real-valued linear systems need be solved; the remaining modes are calculated from the inherent symmetry present in the transform ($\hat{q}_{j,k} = \hat{q}^*_{j,-k}$).

- The system matrices for each mode, A_k, become better conditioned as k increases, primarily because the contribution of second derivatives in the $z-$direction add positive terms $\sim k^2$ to the diagonal. For iterative solvers, then, the higher modes tend to converge sooner, which can result in load imbalancing in parallel solutions.

- At each node there are a total of eight degrees of freedom, which correspond to the eight Fourier components, i.e. real and imaginary components of u, v, w, and p.

- The linear systems corresponding to individual Fourier modes can be solved independently. This is because the coupling is contained only in the nonlinear (convective and associated) terms. Thus, the explicit temporal treatment of these terms decouples the modes within each time step. In essence, the three-dimensional problem has been transformed into a series of two-dimensional problems that are independent of one another, at least as far as the solution of linear systems is concerned.

- The system matrices, A_k are constant. As a consequence, these matrices and any decomposition or preconditioning of these matrices need only be computed once at the beginning of the simulation and can then be reused for all later time steps.

Because the discretization procedure produces an independent matrix equation for each Fourier mode rather than a single large coupled system, a hybrid parallelization approach is employed, where partitioning is performed both in FE space (using METIS [27]) and Fourier space (based on the Fourier mode, k). First, the nonlinear terms in the matrix equation are computed in physical space on the different FE partitions, as illustrated in Figure 2.2a. FFTs are then performed on each subdomain to

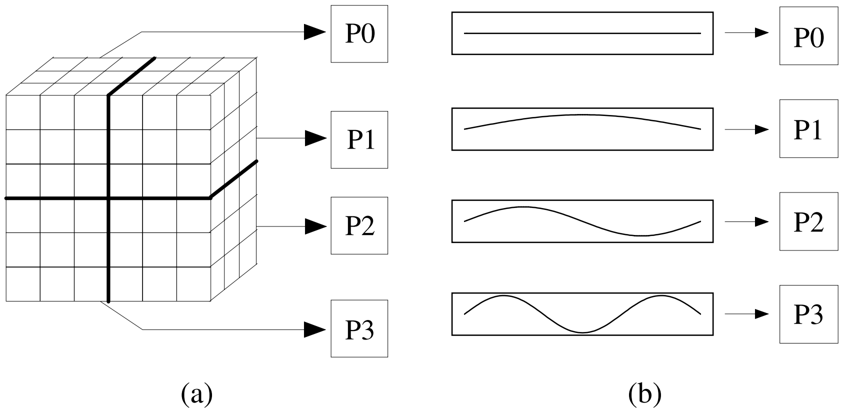


Figure 2.2: Illustration of the parallel partitioning schemes employed in SFELES (P_0 through P_3 represent different processors). (a) Physical space partitioning used for calculation of the terms in the matrix equation. (b) Fourier space partitioning used for the solution of the linear systems, which allows the use of multiple instances of a serial linear solver.

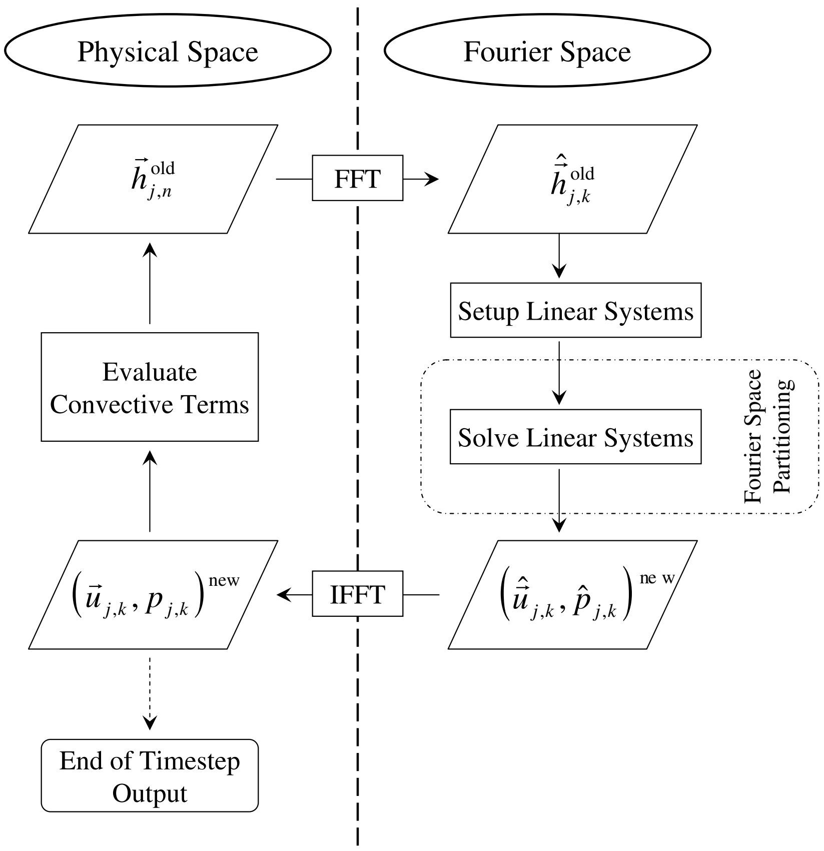

Physical Space
Fourier Space
$\vec{h}^{\text{old}}_{j,n}$
FFT
$\hat{\vec{h}}^{\text{old}}_{j,k}$
Setup Linear Systems
Solve Linear Systems
Fourier Space Partitioning
Evaluate Convective Terms
$(\vec{u}_{j,k}, p_{j,k})^{\text{new}}$
IFFT
$(\hat{\vec{u}}_{j,k}, \hat{p}_{j,k})^{\text{new}}$
End of Timestep Output

Figure 2.3: Illustration of the solution procedure for a typical timestep showing the pseudo-spectral treatment of the convective terms. All parallel calculations are partitioned using the usual physical space approach except for the solution of the linear systems, as indicated.

obtain the matrix terms in Fourier space, where the linear system will be solved. A transformation of data partitioning then occurs from FE to Fourier space. That is, for each Fourier mode k (k between 0 and $N_f/2$), the residuals are gathered from all partitions onto process k, and the solution update is obtained in Fourier space by solving the independent linear systems in a serial manner, as illustrated in Figure 2.2b. Once the linear system solution is complete, the solution update is scattered back to the respective partitions in FE space. This hybrid parallelization approach is found to work well on contemporary Linux clusters [28]. While such clusters nowadays have acceptable bandwidths, they can still suffer from large latency times. The strategy adopted here aims at minimizing these latency times by gathering all parallel communications in two large global communication steps (physical to Fourier space and vice versa), which occur only once per timestep. A broad overview of this procedure is shown in Figure 2.3.

2.3 Linear Solvers

As discussed in the previous section, at each time step a set of $N_f/2$ large sparse linear systems must be solved. These systems are indefinite and nearly symmetric. Each of the $N_f/2$ systems is different, but all are constant in time. Four different linear solvers are employed in this effort, and are now briefly described. Of the four, all are freely available for academic use, with the exception of SAMG, which is only available through a commercial license. Note that, due to the inherent partitioning in Fourier space described previously, the serial implementations of the linear system solvers are used. That is, parallelism is achieved by running multiple simultaneous instances (one on each processor) of the serial solver, as shown previously in Figure 2.2.

2.3.1 Sparse LU Solver

The baseline to which all iterative solvers are compared is the direct sparse LU solver SuperLU [1]. This solver is particularly appropriate for non-symmetric matrices, and uses a variation of Gaussian elimination that is optimized to take advantage of matrix sparsity. In addition, memory cache behavior is optimized in order to significantly improve efficiency. It has been noted that SuperLU is at least three times faster than other direct sparse solvers that do not attempt to optimize based on memory architecture [29]. LU decomposition is an especially attractive approach for the DNS algorithm described in the previous section: because the system matrices are constant in time, the expensive task of performing the matrix decomposition is only performed once at the beginning of the simulation. At each time step, only the inexpensive LU back-substitution phase is performed. Disadvantages include significant time spent in the decomposition phase, and very high memory consumption.

2.3.2 Pointwise AMG Solver

Pointwise AMG (or point-based AMG, see [30], for instance) is a generalization of classical AMG, suitable for many linear problems that arise from discretizations of systems of partial differential equations (PDEs) since it exploits the "physical structure" of the system.

If we discretize a problem governed by n PDEs, where each PDE represents one specific physical unknown, and all unknowns are discretized on the same grid, then the system matrix will be block-sparse: most entries are zero, except for a sparse pattern of dense n by n sub-blocks. The "physical structure" of a PDE system, as given by unknowns and grid points, should be exploited by an AMG solver. However, classical AMG is neither able to distinguish between physical unknowns nor between grid points. Therefore, it usually does not work for PDE systems.

From the perspective of geometric multigrid, it is as if one would mistakenly consider different solution variables (pressure, velocity, etc.), stored at a single node of the mesh, as the same variable, but stored in several nodes that are connected in a dense manner. From a purely physical point of view, this means that variables with different units (Pa, m/s, etc.) are combined.

Pointwise AMG circumvents this problem by using a coarsening based on the (grid) points instead of the variables of the discrete PDE system (as classical AMG does). For this purpose an auxiliary matrix is set up, the so-called primary matrix, in which each scalar entry represents one sub-block of the original matrix. Since neither the definition of the primary matrix nor the concrete inter-level interpolation procedure is unique and strongly depends on the application at hand, pointwise AMG forms a whole framework of AMG methods (for concrete algorithms and applications, see [30]). A classical AMG method is recursively applied to this primary matrix in order to obtain a splitting of the points into coarse and fine level points as well as to construct the interpolation formulas. These are then transferred to the original problem in a straightforward manner.

The pointwise AMG solver investigated here is implemented in the SAMG package [30] and is a "point-based AMG method with a user-defined primary matrix and single unknown interpolation". A classical AMG V-cycle is used as a preconditioner for BiCG-stab [31]. The number of levels depends upon the problem size, as shown in Table 2.1. The coarse grids were generated using a variant of the well-known Ruge-Stüben coarsening algorithm (see [12,32]); to save memory, an aggressive coarsening strategy [32] was used, in which coarser grid levels contain fewer nodes than in the regular Ruge-Stüben algorithm. Mesh coupling strengths were determined using a finite-element discretization of a simple scalar Laplacian on the given mesh. All variables located at the same grid node were interpolated in the same manner by using a "single-unknown interpolation". For its construction, the interpolation formulas computed for the primary matrix were copied pointwise. ILUT, with a drop-tolerance of 0.05 and a maximal number of retained elements per row in the L and U factors equal to 9, was used as the fine grid smoother, with 1 pre- and 1 post-smoothing step.

2.3.3 Smoothed Aggregation Multigrid Solver

The essential difference between classical/pointwise AMG and smoothed aggregation lies in the generation of the coarsening and the interpolation operators. For pointwise AMG, the classical AMG algorithm (or a variant thereof) is applied to an auxiliary scalar problem, and coarsening takes place by eliminating nodes in the finer grid levels. In aggregation solvers, coarsening is then accomplished by first aggregating (block) rows and columns in the finer problems; interpolation is then piecewise constant on the aggregates. As for classical and pointwise AMG, restriction is defined to be the transpose of the interpolation.

While the interpolation operators can be obtained easily in this manner, they are of relatively poor quality unless an additional smoothing procedure is applied. The high frequencies introduced by the piecewise-constant interpolators can for instance be reduced by applying a few Jacobi steps based upon the system matrix. The aggregation itself can be done in an isotropic manner, or can again be based upon strengths taken from an auxiliary matrix.

The smoothed aggregation AMG solver examined in this work is implemented in the Trilinos project [33], which aims at developing robust parallel sparse linear solvers using an object-oriented software design. Of specific interest to this effort is the ML package [34]. For this study, an unrestarted GMRES solver [6] was used with a multilevel AMG preconditioner based on smoothed aggregation [35].

Much effort was spent tuning the algorithm to provide robust and efficient solution of the matrices generated by SFELES. A V-cycle was used, with the number of levels dependent upon the problem size, as shown in Table 2.1. Aggregation was accomplished using a hybrid approach [36]: on the fine levels, the standard method of Vanek [35, 14] was used, while on the coarser levels a maximal independent set (MIS) algorithm [37] was used. The isotropic aggregation procedure is formulated in a block fashion, ensuring that all the unknowns at a particular node belong to the same aggregate. ILUT with a fill value of 2.5 and a drop tolerance of 0.0001 was employed as the fine grid smoother, using a single post-smoothing step and no pre-smoothing. SuperLU was used as the coarse-grid solver. Because the system matrices are constant in time, all preconditioning information was computed only once, and retained for use during all subsequent time steps.

2.3.4 Single-Level Solver

For comparison, a single-level unrestarted GMRES solver was also examined, again using ILUT preconditioning with a fill value of 2.5 and a drop-tolerance of 0.0001.

2.4 Problem Description

Because the linear system solutions for each Fourier mode take place independent of each other, the AMG solver performance can be evaluated using a case with a single active mode on a single processor. In this way, the behavior of the linear solver is separated from the parallel performance of the routines which implement the discretization of the governing equations. Certainly, the overall performance of the three-dimensional DNS solver on large problems is of interest (and will be briefly discussed later), but it is the performance of the "modular" linear solver that is of primary importance here.

As a test case, two-dimensional flow past a circular cylinder at Reynolds number 300 based on cylinder diameter (D) is considered. Circular cylinder flow is well-studied both experimentally and numerically, and the interested reader is referred to review articles by Morkovin [38], Berger and Wille [39], Beaudan and Moin [40], and Williamson [41]. This is an appropriate academic test case because it is well defined, yet contains many interesting flow features including boundary layers, flow separation, shear layers, and a wake region. Moreover, it is physically relevant because at the low Reynolds number of 300 considered here, the approximation of two-dimensional flow is not too far-fetched; this is no longer the case at higher Reynolds numbers.

The computational domain is similar to that shown in Figure 2.1, and is sketched in Fig. 2.4. Note that although the domain is three-dimensional, only the zeroth Fourier mode is activated so the calculated flow field is constrained to be two-dimensional. A Dirichlet u-velocity inlet condition is applied $5D$ upstream of the cylinder center, while a Neumann outlet condition exists $15D$ downstream. Slip-wall conditions are applied at the lateral extents of the domain (top and bottom), located $10D$ from the cylinder center. Three meshes were used, hereafter referred to as Coarse, Medium, and Fine, and are described in Table 2.1. Although the Medium and Fine meshes could, in principle, have been generated in a systematic manner by subdividing triangles in the Coarse mesh, instead they were generated separately in an entirely unrelated manner, which corresponds more closely to what is done in practice. Each case was run for 20 000 time steps ($\Delta t U_\infty / D = 0.002$) using the direct

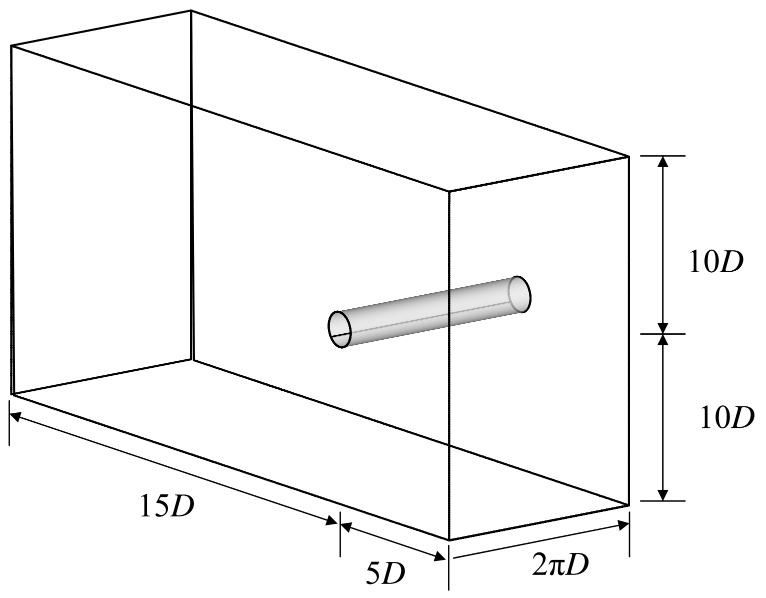


Figure 2.4: Sketch of the computational domain for circular cylinder flow at Reynolds number 300.

solver, until the unsteady wake behind the cylinder was well-developed and no longer contained transient features linked to the initial solution. All runs to determine linear solver performance were restarted from this solution.

The DNS algorithm is formulated such that the change in degrees of freedom during the time step is calculated by the linear system. Therefore, the linear system solution at the previous time step was used as the initial guess for the solution at the following timestep. This assumes, then, that the changes in flow field values that occurred at the previous timestep are similar to those that will occur at the current time step. This was found to provide a slightly improved performance over a zero initial guess. Convergence at each iteration was monitored using a relative residual based upon the residual at the zeroth iteration:

$$\epsilon = \frac{||r||_{L2}}{||r^{(0)}||_{L2}} \tag{2.9}$$

Table 2.1: Summary of parameters for the three different mesh sizes used in this study.

Name	2D Nodes	Levels (AGGR)	Levels (PAMG)
Coarse	22 747	5	6
Medium	105 313	6	7
Fine	172 046	6	8

Table 2.2: Nomenclature of linear solvers and software packages.

Solver	Abbreviation	Package
Direct sparse LU	LU	SuperLU
Preconditioned GMRES	ILUT	ML
Smoothed Aggregation AMG	AGGR	ML
Pointwise AMG	PAMG	SAMG

2.5 Results

The performance of the direct and iterative solvers are presented in this section. Throughout this discussion, the nomenclature shown in Table 2.2 will be used when referring to the solvers introduced in Section 2.3.

2.5.1 Convergence Behavior

Determination of the appropriate relative convergence tolerance for the iterative solvers was based on validation of the time-accurate flow field solution compared to that obtained using the direct sparse LU solver. In fully implicit time-dependent solvers that take large time steps, this question is known to be important because insufficient convergence at the linear level results in incorrect computed frequency spectra [42]. For the semi-implicit scheme implemented in SFELES, the situation turns out to be less ambiguous: it is found that insufficient iterative convergence simply results in a diverging solution as time-stepping progresses.

The unsteady flow properties at a fixed point in the domain, located $5D$ downstream of the center of the cylinder, were monitored to verify the time-accurate solution. This data was obtained by running the calculations for 100 000 time steps (approximately 50 shedding cycles) after the periodic flow field had developed. A

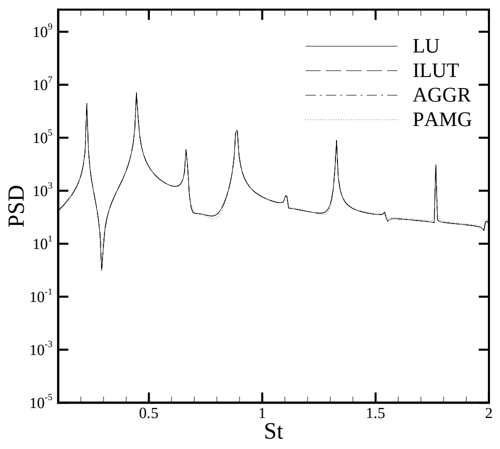
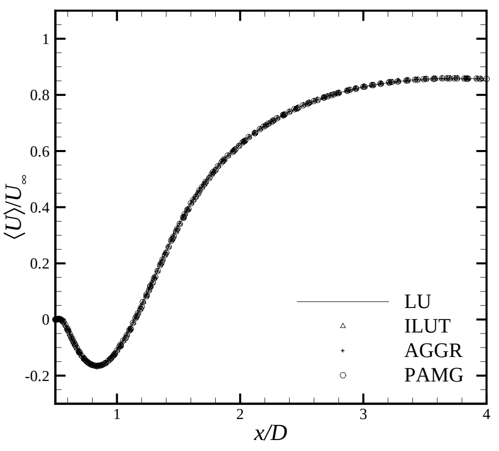


Figure 2.5: Comparison of the temporal behavior of the cylinder wake using the direct linear solver and the iterative solvers with a convergence threshold of $\epsilon = 0.001$. Plotted is the nondimensional power spectral density of the unsteady u-velocity as a function of non-dimensional frequency, $St = fD/U_\infty$. The data is taken at a control point located $5D$ downstream of the center of the cylinder on the Coarse mesh.

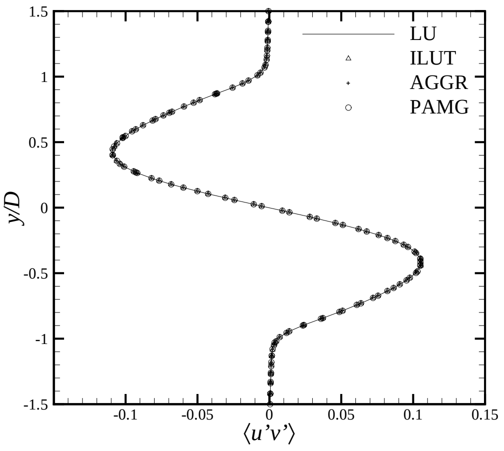
representative power spectral density plot of the unsteady u-velocity signal at the monitor point is shown in Figure 2.5 for the relative convergence threshold $\epsilon = 0.001$. It is found that all iterative solvers produce essentially the same spectrum as the direct solver for this threshold.

In addition to the time-accurate monitoring of the single control point, the complete time-averaged flow field was also compared to the direct sparse LU solution. The results, both in terms of zeroth-order (mean velocities) and first-order (Reynolds stresses) statistics, were found to be virtually identical for $\epsilon = 0.001$, as shown in Figure 2.6. Therefore, considering the direct solver to produce the “exact” data, using a relative convergence threshold of $\epsilon = 0.001$ is sufficient for all of the iterative solvers investigated here.

General convergence behavior for each of the solvers on the three levels of grid refinement can be seen in Figure 2.7. As expected, the single-level solver converges quickly only during the initial iterations, when high-frequency errors are damped rapidly. Thereafter, it nearly stagnates at four orders of magnitude convergence. The



(a)



(b)

Figure 2.6: Comparison of the mean flow field statistics computed using the direct linear solver and the iterative solvers with a convergence threshold of $\epsilon = 0.001$. (a) Time-averaged u velocity along a horizontal line through the center of the cylinder and extending into the wake region. (b) $u'v'$ Reynolds stress along a vertical line located $5D$ downstream of the center of the cylinder. All data is computed using the Coarse mesh.

situation is much better when using pointwise or smoothed aggregation AMG. The spikes observed in the SAMG convergence histories are normal, since BiCG-stab, not GMRES, is used as an accelerator here. At first sight, SAMG appears to converge about twice as fast as ML. However, it should be noted that two AMG-V cycles per iteration are performed for SAMG, whereas only a single V-cycle is performed for ML. Finally, note that for both of the AMG solvers, the convergence tolerance ($\epsilon = 0.001$) is reached in only 1 to 2 Krylov iterations.

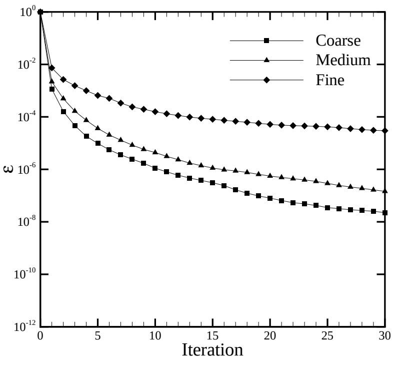
2.5.2 Performance

Memory consumption and CPU time data were collected by running an additional 200 time steps after the 20 000 initial time steps had been completed. All CPU timings reflect the linear solution phase only, and do not include the additional overhead associated with the linear solvers (set-up, computation of exact or approximate factorizations, etc.) and the DNS solver itself. Similarly, the memory consumption values presented only reflect the memory usage of the linear solver.

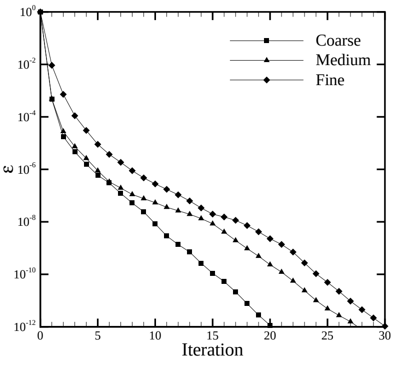
Figure 2.8a shows the normalized memory usage for each solver and mesh, according to

$$M^* = \frac{M}{M_0} \tag{2.10}$$

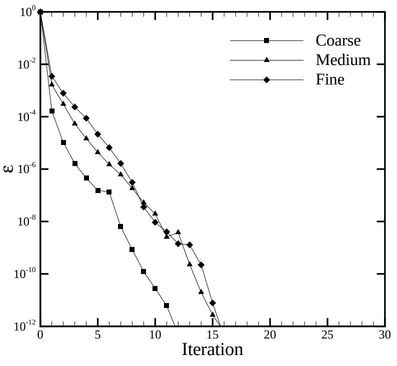
where M is the memory usage of the given solver on a particular mesh, and M_0 is the memory usage of SuperLU on the Coarse mesh. The sizable memory requirement of the direct sparse LU solver is clearly evident, even on the Coarse mesh. As expected, the rate at which memory usage increases with mesh size is also dramatically larger than for the iterative solvers. In order to better see the performance of the AMG solvers, the data is re-plotted in Figure 2.8b for the iterative solvers only, and tabulated in Tables 2.3/2.4. Memory requirements scale linearly to within a reasonable approximation. The pointwise AMG solver requires less memory than both the single-level and smoothed aggregation AMG solvers. All iterative solvers perform remarkably well in terms of memory: the Fine mesh problem contains nearly 1.4 million unknowns, yet fits within the 2GB memory constraint of 32-bit processors.



(a)



(b)



(c)

Figure 2.7: Convergence behaviors of the iterative solvers: (a) GMRES with single-level ILUT preconditioning (ILUT), (b) smoothed aggregation AMG (AGGR), and (c) pointwise AMG (PAMG).

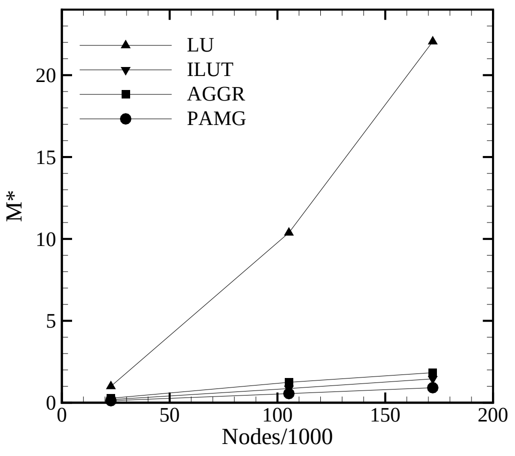
Figure 2.9a shows the normalized linear solving time for each solver and mesh according to

$$T^* = \frac{T}{T_0} \tag{2.11}$$

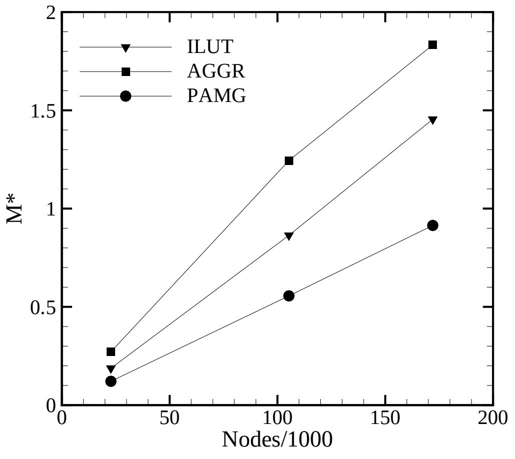
where, again, T is the linear solving time of the given solver on a particular mesh, and T_0 is the linear solving time of SuperLU on the Coarse mesh. Here we see the obvious drawback to the single-level iterative solver, where the performance progressively degrades as the mesh is refined. The AMG and direct sparse LU solvers exhibit much better scaling; the fact that the scaling is not entirely linear is likely due to the unrelated manner in which the Coarse, Medium and Fine meshes were generated. In order to better see the performance of the AMG solvers, the data is re-plotted in Figure 2.9b for the AMG and direct sparse LU solvers, and tabulated in Tables 2.3/2.5. Again, the pointwise AMG solver outperforms the smoothed aggregation AMG solver, with performance improving with mesh size. The competitive scaling of the direct sparse LU solver is a byproduct of the semi-implicit time-stepping algorithm, which requires only the inexpensive substitution phase at all but the first time step.

An explanation for the linear scaling of the SuperLU CPU time observed in Figure 2.9 is appropriate. The scaling of the substitution phase is linear in the number of nonzeros in the L and U factors. In the dense case, this results in n^2 scaling. For sparse systems, however, better performance can often be reached. This is apparent in the memory usage (Figure 2.8a), but even more so in the nearly linear CPU time scaling, likely because SuperLU exploits the CPU cache more efficiently for larger problems.

The actual values for the memory usage and averaged linear solving time using the direct sparse LU solver for the three meshes are given in Table 2.3. From these values, it is readily apparent that memory is the limiting factor when using direct solvers with semi-implicit formulations. It should be noted that typical CPU times on state-of-the art Linux workstations with Intel CPUs are approximately 1/5th of those shown in Table 2.3.



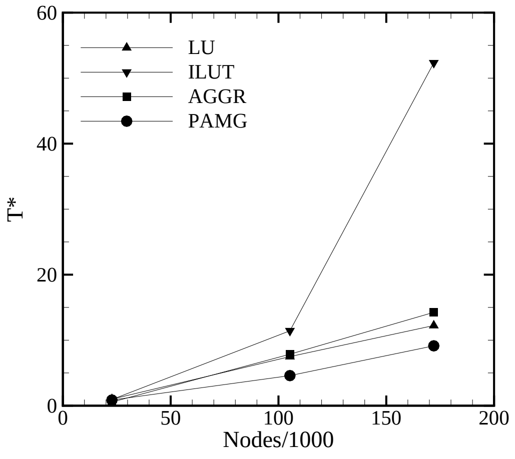
(a)



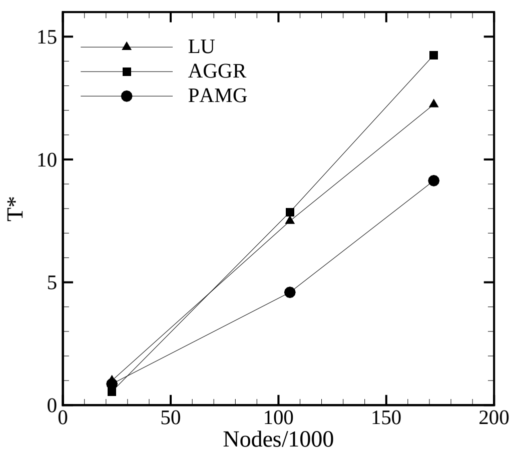
(b)

Figure 2.8: Memory usage normalized to the memory usage of the sparse direct LU solver on the Coarse mesh; (a) all solvers and (b) iterative solvers only.

SAMG clearly appears to outperform ML in terms of memory and CPU for the considered case. We should nevertheless remark that exact comparisons between the pointwise and smoothed aggregation AMG solvers are difficult due to the significant amount of tuning required to obtain good results. For both solvers, CPU and memory were improved approximately an order of magnitude by tuning from the default values. It is important to note that the GMRES algorithm used in conjunction with ML



(a)



(b)

Figure 2.9: Linear system solution time normalized to the solution time of the direct sparse LU solver on the Coarse mesh; (a) all solvers and (b) excluding single-level ILUT.

consumes significantly more memory than the BiCG-stab solver adopted for SAMG. Most tuning decisions require a trade-off between memory and CPU. For instance, in the smoothed aggregation solver, significant memory reduction can be obtained by reducing the ILUT fill value from 2.5 to 1.5. In doing so, memory is reduced by 14%, but CPU time is increased by 25%. For both packages, guidance from the developers was required to achieve the performance presented here. Nonetheless, implementa-

Table 2.3: Memory usage (double precision) and linear system solving time for the direct sparse LU solver on an SGI Origin 3900 running MIPS R16000 processors at 700MHz.

Mesh	Memory Usage (MB)	Linear Solving Time (sec)
Coarse	933	3.47
Medium	9,682	26.03
Fine	20,574	42.52

Table 2.4: Memory usage normalized to memory usage of the direct sparse LU solver on the Coarse mesh.

Solver	Coarse	Medium	Fine
LU	1.00	10.38	22.05
ILUT	0.19	0.86	1.45
AGGR	0.27	1.24	1.83
PAMG	0.12	0.56	0.91

tion of both AMG packages as "modular" linear solvers has proven very successful considering the performance improvements over single-level and direct solvers.

2.6 Application to Fully Turbulent DNS

Based on the encouraging results of SAMG on the single Fourier mode case, this linear solver has been tested on a large three-dimensional DNS case of unsteady turbulent flow past a circular cylinder at Reynolds number 3 900. The computational geometry sketched in Figure 2.4 was again used, but with the spanwise dimension reduced to πD. In the plane of the cylinder, approximately 200 000 linear triangle elements are used, with typical element side lengths of $0.007D$ near the cylinder. Streamwise vortical structures in the fully developed Kárman vortex street are expected to scale with Reynolds number as [43]

$$\lambda_z \approx 20 Re^{-1/2} D \approx 0.3D. \qquad (2.12)$$

In order to resolve these structures, approximately 64 transverse modes are required. This number was increased by a factor of four to capture the fine structures in the

Table 2.5: Linear system solution time normalized to solution time of the direct sparse LU solver on the Coarse mesh.

Solver	Coarse	Medium	Fine
LU	1.00	7.49	12.24
ILUT	0.94	11.43	52.34
AGGR	0.55	7.86	14.25
PAMG	0.86	4.59	9.14

fully turbulent wake behind the cylinder. In order to de-alias the spectral formulation, only half of the total modes are activated, resulting in 512 total Fourier modes (256 active). The complete mesh contains approximately 51 million nodes (204 million unknowns).

Performance of the discretization formulation in regards to correctly capturing the flow physics will not be discussed in this paper (see [11] for LES results). Rather, of interest is the computational performance in order to see whether the use of AMG allows feasible solution of such large problems. The case was run on the "Marylou4" Linux cluster, part of the Ira and Marylou Fulton Supercomputing center at Brigham Young University. A total of 128 Pentium EM64T Xeon processors running at 3.6GHz were used, resulting in one linear system solve per processer per time step. Parallel performance is summarized in Table 2.6, where CPU time for the major components of the parallel algorithm are compared for a single processor (4 total Fourier modes, 2 active) and for 128 processors (512 total Fourier modes, 256 active). In both cases, turbulent disturbances were active in the flow field, and the number of iterations required for linear system convergence was fixed at 2.

When determining the parallel efficiency for 3D code runs, the cost of the FFTs is not included. When many Fourier modes are used, the FFTs certainly incur a non-negligible extra cost. However, this cost is totally unrelated to performance losses due to parallel communication. At present, the overall parallel efficiency of the solver lies near 77% for this type of very large computation, which is well above the typical performance of other contemporary implicit Navier-Stokes solvers. Consider-

Table 2.6: Parallel performance of the SFELES algorithm using pointwise AMG on a large three-dimensional DNS case containing 51 million nodes.

	1 Proc.	128 Proc.	Eff.
Lin. Terms/B.C. (s)	0.79	0.90	
Convective Terms (s)	4.32	4.33	
Parallel Gather/Scatter	0.0	1.48	
Lin. System Solve (s)	6.68	8.58	
Fwd/Inv FFT (s)	0.47	1.54	
Total with FFT (s)	12.3	16.9	
Total w.o. FFT (s)	11.83	15.36	77%

ing the overall cost of 16.9 s per time step, a 100 000 time step run would require approximately three weeks to complete.

While for state-of-the-art DNS calculations this is still feasible, it is obvious that any gain in parallel performance would be of great interest. We note from Table 2.6 that the cost of certain sequential tasks (linear terms and boundary conditions, linear system solve) increases when running in parallel. This is due to the excessive use of parallel barrier statements used to synchronize the different processors. Through further optimization of the code, this shortcoming can be removed. The remaining parallel overhead lies in the FEM to Fourier space partition conversion using parallel gather/scatter statements, performed to allow solving the systems in sequential mode. These steps are limited by the bandwidth of the cluster and can unfortunately not be improved. For this reason, a follow-on study is underway in which the large communication steps are bypassed by solving the linear systems in Eq. (2.8) in a distributed manner. By building and solving on the physical space partitions, the global communication is replaced with smaller partition boundary data communications. While the volume of data to be communicated strongly decreases, more communication steps will be needed and the associated latency costs may be significant.

One might also suggest that the global communication step could be eliminated by building the system in a plane-wise manner using the same partitioning as currently employed for the system solution phase. However, such a data structure requires that all (I)FFTs be performed in parallel, which was found to be very costly.

2.7 Conclusion

In this contribution, the use of AMG solvers as an alternative to the single-level and direct solvers currently used in many DNS/LES codes has been investigated. This study is different from previous AMG studies in the sense that for DNS/LES applications, convection is not treated implicitly. Consequently, only elliptic and parabolic terms contribute to the linear systems. Both classical point-wise AMG and smoothed aggregation AMG have been tested, and their performance has been compared to single-level ILUT preconditioned GMRES and direct sparse LU.

By considering the time-dependent solution signal as well as the zeroth and first order time-averaged solutions, it was found that a relative convergence threshold of $\epsilon = 0.001$ was sufficient for the iterative solvers. In general, only 1-2 Krylov iterations were required to reach this level of convergence. The single-level ILUT solver, as well as the pointwise and smoothed aggregation AMG solvers, were found to yield linear scaling in terms of memory consumption. As expected, this is not at all the case for the sparse LU solver. In terms of CPU time, the single level solver was found to have acceptable convergence behavior only on coarse meshes. Surprisingly, the CPU cost of the direct solver scales nearly linearly, although the fill-in increases superlinearly with the problem size. Both multigrid solvers work very well; the fact that the scaling is not entirely linear is likely due to the manner in which the different grids were generated. For the time being, the pointwise AMG solver outperforms the smoothed aggregation solver both in terms of memory usage and CPU time. However, given the amount of fine-tuning that was required to obtain these results, it is difficult to draw any definite conclusions from this.

Finally, the feasibility of applying AMG strategies to full DNS cases was tested by considering the simulation of the flow around a circular cylinder at a Reynolds number of 3 900 using SAMG as the linear solver. It has been shown that such problem sizes now indeed lie within reach. A comparison of the scalar and sequential timings of SFELES shows that a quite satisfactory parallel efficiency of 77% is obtained on 128 processors of a modern Linux cluster. In the near future it will be investigated whether the parallel efficiency of this solver can be further improved by employing the distributed-memory versions of the solvers in the SAMG and Trilinos packages.

Chapter 3

Performance of the Parallelization Approaches

3.1 Introduction

As large clusters of computers become prevalent and processors increase in computational power, more complicated computational fluid dynamics (CFD) problems can be analyzed. At first, only solutions to the Reynolds-Averaged Navier-Stokes (RANS) equations could feasibly be solved, but these technological advancements increasingly allow for more complicated analyses, such as direct numerical simulation (DNS) and large-eddy simulation (LES). The complexity of DNS/LES problems solvable on modern computers is still highly limited by available computational speed and memory.

This paper presents a numerical formulation for solving the unsteady, incompressible Navier-Stokes equations utilizing a hybrid Spectral/Finite-Element (FE) discretization. This discretization allows full three-dimensional flowfield simulations for complex two-dimensional or axi-symmetric geometries. As a result of this formulation, the full three-dimensional linear system is converted into a series of independent linear systems (one for each Fourier mode) within each time step. As a consequence, two parallelization techniques are readily implemented. The first (termed Fourier space partitioning) is to simultaneously solve each Fourier mode's linear system independently on each processor. The second (termed physical space partitioning) is to employ all processors in solving each linear system, sequentially working through the Fourier modes.

Snyder and Degrez [11] were the first to implement the hybrid Spectral/FE discretization for DNS/LES problems. They employed a (somewhat inefficient) form of the Fourier space partitioning specifically aimed at shared-memory computing architectures with relatively few processors. Load imbalancing was noted as a potential problem due to linear systems taking varying amounts of time to solve. Another discovery was that as more Fourier modes were employed (using the same number of processors), the parallelization became more efficient because as systems finished, they were able to "ask" for the next system to solve instead of idly waiting.

Larson et al [44] found AMG implemented in the Fourier space partitioning approach was computationally cheaper (in terms of memory usage and computational time) than direct LU-Decomposition methods, which was only exaggerated for problems of larger size. Vanden-Abeele et al [28], however, found that on a Linux cluster with distributed memory, even for a moderate number of 32 Fourier modes using 16 processors, communication between processors before and after the linear solution solve was so computationally expensive that it ran slower than with only 8 processors. It appears that additional (possibly cluster-related) factors contributed to this conclusion, as no such behavior was observed in this study.

The purpose of this chapter is to compare the two parallelization methods to show that the novel Fourier space partitioning approach is superior to the more common physical space partitioning.

3.2 Stabilized Finite-Element Formulation

The DNS solver used for these tests is the hybrid Spectral/FE code SFELES [10, 11], which allows simulation of three-dimensional unsteady incompressible flows with planar or cylindrical geometries of arbitrary complexity. A representative hybrid mesh is illustrated in Figure 3.1, along with a representative flow field past a circular cylinder. The governing equations are discretized in cross planes (z = constant) using linear triangle finite-elements. The transverse direction is represented by means of a truncated Fourier series, which assumes periodicity in the z direction. Complex two-dimensional geometries can be accommodated through the unstructured in-plane mesh, while storage requirements are reduced because only a single two-dimensional mesh and associated overhead need be stored.

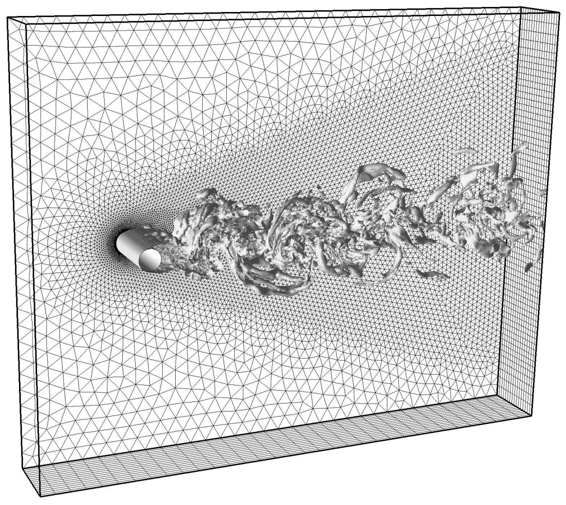


Figure 3.1: Representative hybrid spectral/finite-element mesh for circular cylinder flow, where linear finite-elements are employed in the two-dimensional cross planes, while a truncated Fourier series is employed in the transverse direction. Shown are isosurfaces of vorticity magnitude at Reynolds number 8 000.

The dimensionless form of the incompressible Navier-Stokes equations is first discretized in time in a second-order accurate manner. An implicit Crank-Nicholson scheme is used for the pressure and viscous terms, while an explicit Adams-Bashforth method is used for the convective terms. The resulting expressions for the continuity and momentum residuals are set to zero at each time step:

$$\begin{aligned} R_{Cont}^{n+1} &= \nabla \cdot (\vec{u}^{n+1}) = 0 && (3.1) \\ \vec{R}_{Mom}^{n+1} &= \frac{1}{\Delta t}(\vec{u}^{n+1} - \vec{u}^{n}) + \nabla p^{n+\frac{1}{2}} - \frac{1}{2Re}\nabla^2(\vec{u}^{n+1} + \vec{u}^{n}) \\ &\quad + \frac{3}{2}\nabla \cdot (\vec{u}^{n} \otimes \vec{u}^{n}) - \frac{1}{2}\nabla \cdot (\vec{u}^{n-1} \otimes \vec{u}^{n-1}) = 0 && (3.2) \end{aligned}$$

The solution at time t is represented in the FE plane using linear triangle elements for both the pressure and the velocity (P1/P1):

$$q(x,y,z,t) = \sum_{j=1}^{N_n} q_j(z,t)\,\phi_j(x,y) \qquad (3.3)$$

where q_j represents the value of the unknown at node j, N_n is the number of two-dimensional nodes, and ϕ_j represents the piecewise linear basis function associated with node j. The in-plane discretization at any node follows directly from the variational statement:

$$\int \phi_i R^{n+1}_{Cont} d\Omega - \int \phi_i \tilde{\nabla} \cdot (\tau_{pspg} \vec{R}^{n+1}_{Mom})\, \mathrm{d}\Omega = 0 \tag{3.4}$$

$$\int \phi_i \vec{R}^{n+1}_{Mom} d\Omega + \int \phi_i \tau_{supg} (\vec{u} \cdot \tilde{\nabla}) \vec{R}^{n+1}_{Mom}\, \mathrm{d}\Omega = 0 \tag{3.5}$$

where the operator $\tilde{\nabla}$ is the cross plane gradient, defined as $(\partial/\partial x,\ \partial/\partial y,\ 0)$. The second terms in the above equations are PSPG and SUPG stabilizations, respectively, that are added to remove compatibility and convective instabilities (see, for instance Ref. [24]).

The attentive reader may object that no such stabilization terms should be used in DNS, as they act as artificial subgrid scale stresses. This is certainly true for marginally resolved simulations, in which the PSPG and SUPG terms can be of order zero. However, for well-resolved simulations, such as presented here, the solution varies smoothly between elements. The numerical stabilization terms then become small second-order discretization errors. This is an area of active research (see, for instance, Ref. [25,26]); numerical experiments carried out using this solver for marginally-resolved DNS of channel flow show virtually no artificial dissipation due to the PSPG stabilization, and minimal dissipation due to the SUPG stabilization.

The discretized continuity and momentum equations may be cast in a matrix form, where all nonlinear contributions have been gathered on the right hand side as follows:

$$\left[M_1 + M_2 \frac{\partial}{\partial z} + M_3 \frac{\partial^2}{\partial z^2}\right] \begin{bmatrix} \Delta p(z) \\ \Delta \vec{u}(z) \end{bmatrix} = \begin{bmatrix} R^n_p(z) \\ \vec{R}^n_u(z) \end{bmatrix} \tag{3.6}$$

The precise form of the matrices M_1, M_2, and M_3 can be easily derived, and for purposes of brevity are not written here (see Ref. 11 for details).

In the transverse, z, direction a truncated Fourier series is employed, for which periodicity is assumed. A nodal unknown $q_{j,n} = q(x_j, y_j, z_n, t)$, is therefore represented as

$$q_{j,n}(t) = \frac{1}{N_f} \sum_{k=-N_f/2+1}^{N_f/2-1} \hat{q}_{j,k}(t) \exp\left(\frac{2\pi I k}{N_f} n\right) \tag{3.7}$$

where $I = \sqrt{-1}$ and N_f is the number of modes used in the expansion. Doing so, the global three-dimensional linear problem in physical space, Eq. (3.6), is transformed into a series of two-dimensional linear problems for each Fourier mode k:

$$A_k \begin{bmatrix} \Delta\hat{p}_k \\ \Delta\hat{\vec{u}}_k \end{bmatrix} = \begin{bmatrix} \hat{R}^n_{p,k} \\ \hat{\vec{R}}^n_{u,k} \end{bmatrix} \tag{3.8}$$

where $A_k = [M_1 + M_2(2\pi I k/L) - M_3(2\pi k/L)^2]$. For computational efficiency, the nonlinear terms, $R^n_p(z)$ and $\vec{R}^n_u(z)$, are computed in physical space and then transformed to Fourier space, resulting in a pseudo-spectral procedure.

Several key items directly affecting the linear system solution procedure should be noted:

- In practice, only $N_f/2$ real-valued linear systems need be solved; the remaining modes are calculated from the inherent symmetry present in the transform ($\hat{q}_{j,k} = \hat{q}^*_{j,-k}$). In addition, because the Fourier coefficients of the 0^{th} mode are real valued, the linear system contains only half the number of unknowns.

- The system matrices for each mode, A_k, become better conditioned as k increases, primarily because the contribution of second derivatives in the $z-$direction add positive terms $\sim k^2$ to the diagonal. For iterative solvers, then, the higher modes tend to converge sooner, an important consideration in making parallelization decisions.

- At each node there are a total of eight degrees of freedom, which correspond to the eight Fourier components, i.e. real and imaginary components of u, v, w, and p.

- The linear systems corresponding to individual Fourier modes can be solved independently. This is because the coupling is contained only in the nonlinear (convective and associated) terms. Thus, the explicit temporal treatment of these terms decouples the modes within each time step. In essence, the three-dimensional problem has been transformed into a series of two-dimensional problems that are independent of one another, at least as far as the solution of linear systems is concerned.

- The system matrices, A_k are constant. As a consequence, these matrices and any decomposition or preconditioning of these matrices need only be computed once at the beginning of the simulation and can then be reused for all later time steps.

3.3 Parallelization Techniques

Because the discretization procedure produces an independent matrix equation for each Fourier mode at each time step rather than a single large coupled system, two parallelization techniques naturally arise. The first, where each Fourier mode linear system is calculated on a separate processor, will be referred to as "Partitioning in Fourier Space". The second is to parallelize the linear system solves, sequentially solving each Fourier mode, hereafter called "Partitioning in Physical Space".

3.3.1 Partitioning in Fourier Space

While the first parallelization method is called "partitioning in Fourier space", in reality it is a hybrid parallelization approach which uses partitioning in both Fourier space (according to Fourier mode) and FE space (using METIS [27]). This is accomplished by first, calculating the right hand side terms for the linear systems in physical space on all FE partitions, shown in Figure 3.2a. Next, FFTs are performed on each subdomain to get the right hand side terms in Fourier space. The residuals from all FE space partitions for a given Fourier mode are then gathered onto a single processor for solving the linear system. Each processor then simultaneously calls a serial version of the linear solver and individually solves its linear system, illustrated in Figure 3.2b. While reducing the solving time of the linear solver accelerates the algorithm at the same rate, reducing the memory usage of the linear solver reduces overall memory

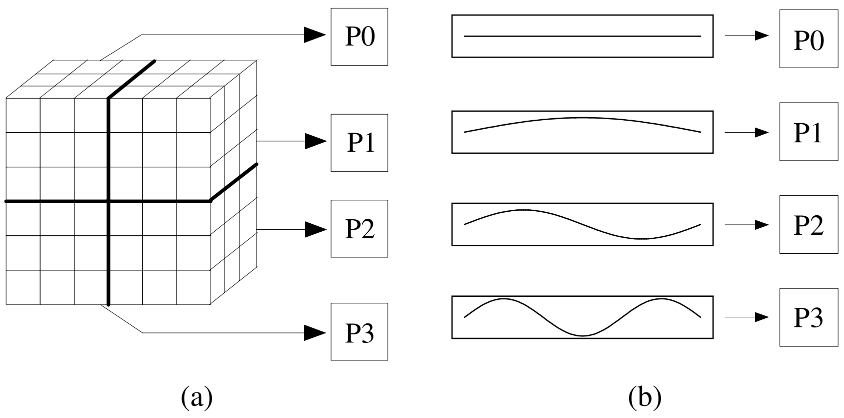


Figure 3.2: Illustration of the parallel partitioning schemes employed in SFELES (P_0 through P_3 represent different processors). (a) Physical space partitioning used for calculation of the terms in the matrix equation. (b) Fourier space partitioning used for the solution of the linear systems.

usage by that amount times the number of linear systems. After solving, the solution is updated and then the information is sent back to the processor containing each FE partition. This parallelization method has been found to work well on Linux clusters [28]. Since latency is still an issue when communicating between processors, this method attempts to minimize latency by requiring only two global communication steps for each algorithm time step. These occur before and after solving the linear system of each Fourier mode.

3.3.2 Partitioning in Physical Space

Partitioning in physical space is performed strictly in FE space. Once again, first the right hand side terms for the linear systems are calculated in physical space on all FE partitions. Next, FFTs are performed on each subdomain to get the right hand side terms in Fourier space. However, at this point, no gathering is required because each processor already has the information it requires. Each Fourier mode linear system is then solved by all processors using a parallel linear solver. In this case only one instance of the linear solver is active at any given time, but multiple calls are made to the same solver each time step. Therefore, reduction of the memory usage and solving time for one linear system reduces the overall memory usage and computation time by that rate times the number of linear systems. As a result of this partitioning method, since the solution results are already partitioned in FE space, gathering is again unnecessary. By eliminating the global communication before and after linear system solves, communication is introduced in the linear system solution phase itself.

While communication between processors can be costly, there are other considerations when comparing the performance of these two parallelization methods. When partitioning in Fourier space, the number of processors is limited to half the number of Fourier modes. In contrast, there is no theoretical limit to the number of processors which can be implemented in physical space partitioning.

As mentioned previously, system matrices for each mode, A_k, become better conditioned as k increases, allowing the higher modes to converge sooner when using iterative solvers. When partitioning in Fourier space, the processors solving the higher modes must therefore inactively wait until the lower modes have all converged. For physical space partitioning, if a mode converges quickly, all processors advance to the next system.

3.4 Linear Solvers

At each time step $N_f/2$ large sparse linear systems must be solved. While all of these systems are positive indefinite, nearly symmetric, and constant in time, each is different. All three solvers employed for comparison are freely distributed for academic use.

3.4.1 Sparse LU Solver

One solver evaluated is the direct sparse LU solver, SuperLU, developed at University of California at Berkeley. Demmel et al. [1] created this efficient direct solver especially for sparse matrices. By grouping columns of similar structure together into supernodes, highly optimized BLAS routines can be utilized for more efficient factorization. The code outperformed other direct sparse solvers (not optimized according to memory architecture) by a factor of three [29]. LU decomposition tends to be expensive, but because the system matrices are constant at all time steps, the factorization only needs to be calculated once and the inexpensive back-substitution is applied at each subsequent time step. Some disadvantages include large CPU time requirements in the decomposition phase and high memory consumption. SuperLU DIST [2] is a parallel version of SuperLU designed for clusters having distributed memory. While SuperLU MT [45] is a parallel version of SuperLU designed for shared memory systems, the Trilinos project (see 3.4.2) is currently not compatible with this package.

3.4.2 Smoothed Aggregation Multigrid Solver

The second solver employed is the smoothed aggregation AMG solver developed by Sandia National Laboratories in the ML package [34] of the Trilinos project [33]. The Trilinos project's purpose is to develop robust parallel linear solvers using object-oriented software. ML allows for both serial and parallel implementation of many linear AMG algorithms. An unrestarted GMRES solver [6] is used with a multilevel AMG preconditioner based on smoothed aggregation [35].

The AMG algorithm employed a single V-cycle containing 5 levels, created through a hybrid aggregation approach [36], where the standard method of Vanek [35, 14] is used on the fine grids, and the uncoupled maximal independent set [37] method is used on the coarse grids. Block isotropic aggregation, according to the number of unknowns at each node, is used to ensure all unknowns for a given node belong to the same aggregate. A single post-smoothing step using ILUT (with a fill value of 2.5 and a drop tolerance of 0.0001) is used on the fine grids, while SuperLU is used as the coarse-grid solver. Once again, since the matrices are constant at every timestep, the preconditioning information is only calculated once and reused at each subsequent time step.

Table 3.1: Summary of parameters for the two in-plane mesh sizes used in this study.

Name	2D Nodes	2D DoFs
Coarse	12 275	98 200
Fine	32 383	259 064

3.4.3 Single-level Generalized Minimal Residual (GMRES) Solver

A single-level unrestarted GMRES solver [6], also from the Trilinos project [33], is used as the third solver. This also uses ILUT preconditioning (with a fill value of 4.0 and a drop-tolerance of 0.0001).

3.5 Problem Description

The test case for comparison of the solvers employed in this study is three-dimensional flow past a circular cylinder at Reynolds number 300, based upon cylinder diameter D. This problem was chosen based upon the periodic temporal behavior and various spatial flow features, including boundary layers, flow separation, shear layers, and a wake region. Circular cylinder flow is also a well-studied phenomena [38, 39, 40, 41].

The computational domain, sketched in Figure 3.3, consists of a velocity inlet applied $5D$ upstream of the cylinder and a pressure outlet condition $15D$ downstream. The upper and lower walls are treated with a slip wall condition, located $10D$ above and below the cylinder. Two different meshes are used in this study, referred to as Coarse and Fine, as described in Table 3.1. The DNS algorithm using the direct sparse LU solver and partitioning in Fourier space was run for 20 000 time steps ($\Delta t U_\infty / D = 0.005$), until the flowfield was fully developed and void of any initial condition influences. All performance comparisons (solvers and partitioning methods) continued calculations from these flow quantities.

The DNS algorithm is set up in such a manner that the change in the unknown quantities is calculated at each time step instead of the actual unknown quantities. For the iterative solvers the initial guess is the solution at the previous time step, making

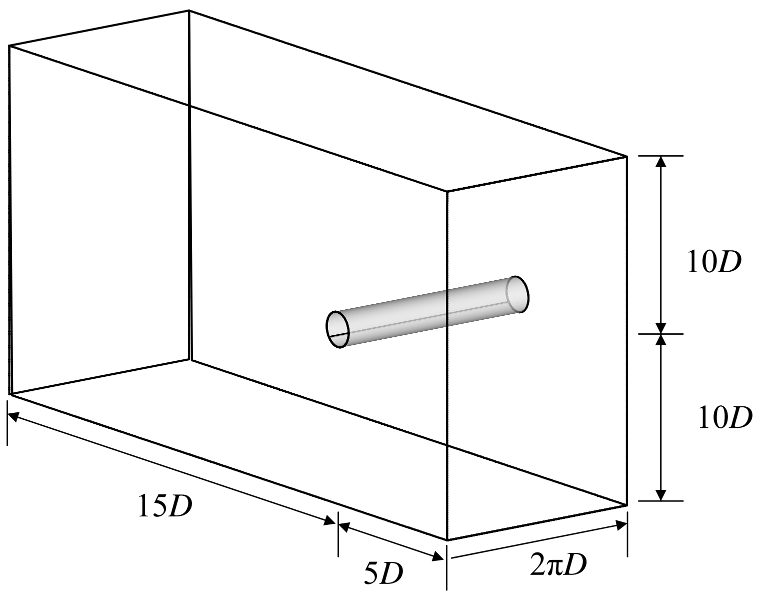


Figure 3.3: Sketch of the computational domain for circular cylinder flow at Reynolds number 300.

the assumption that the change in the next time step will be comparable to the change at the previous time step. This approach resulted in slightly improved performance compared to assuming a zero initial guess (assuming the unknown quantities had no change from the previous time step). The iterative solvers monitored convergence based upon a relative convergence threshold, calculated according to the residual at the zeroth iteration, according to

$$\epsilon = \frac{\|r\|_{L2}}{\|r^{(0)}\|_{L2}}. \tag{3.9}$$

3.6 Computer Hardware

For these comparisons, two different computing systems are compared, "Marylou" and "Marylou4", which are both part of the Ira and Mary Lou Fulton Supercomputing Laboratory at Brigham Young University. Marylou is an SGI Origin 3900 with 128 MIPS R16000 700 MHz processors and 64 GB of total shared memory. Marylou4 is a distributed memory Linux cluster with 610 compute nodes having two Intel Xeon

Table 3.2: Nomenclature of linear solvers and software packages.

Partitioning	Solver	Abbreviation	Package
Fourier Space	Direct sparse LU	LU-F	SuperLU
Fourier Space	Smoothed Aggregation AMG	AGGR-F	ML
Physical Space	Direct Sparse LU	LU-P	SuperLU DIST
Physical Space	Smoothed Aggregation AMG	AGGR-P	ML
Physical Space	Preconditioned GMRES	ILUT-P	ML

2.66 GHz Dual-Core processors and 8 GB of memory per node. These clusters enable comparisons between shared and distributed memory architectures.

3.7 Results

The performance of the two parallelization techniques are presented in this section. In this discussion, the nomenclature shown in Table 3.2 is used for condensed referencing to the solvers introduced in Section 3.4. In an attempt to help the reader remember and distinguish the methods, "-F" is used for partitioning in Fourier space and "-P" for partitioning in physical space.

3.7.1 Solver Verification

Previous research indicated the relative convergence threshold for the iterative solvers should be $\epsilon = 0.001$ [44]. Verification for this tolerance when solving in physical space was conducted using AGGR-P by monitoring the unsteady lift coefficient (c_l) on the cylinder for 50 000 time steps (approximately 50 shedding cycles) after the periodic flow field had developed. The power spectral density of the unsteady c_l of AGGR-P and AGGR-F compared to LU-F, shown Figure 3.4, for the relative convergence threshold $\epsilon = 0.001$, demonstrates temporal consistency between all solution methods.

Spatial verification for AGGR-F and AGGR-P were also compared to the LU-F solution. The zeroth-order (mean velocities) and first-order (Reynolds stresses) statistics are shown in Figure 3.5. The Reynolds stress ($u'v'$) located $2D$ downstream are essentially identical, while $5D$ downstream there are differences. A smaller conver-

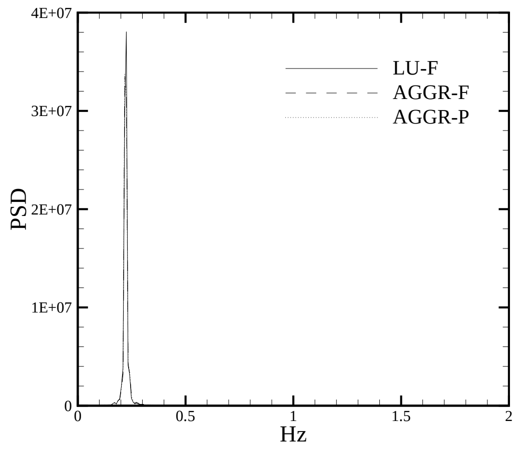
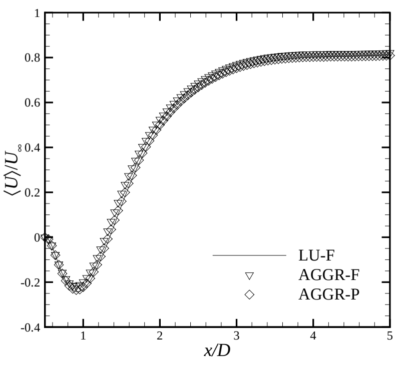


Figure 3.4: Comparison of the temporal behavior of the cylinder wake using the direct linear solver in Fourier space (LU-F) and the iterative solvers AGGR-F and AGGR-P with a convergence threshold of $\epsilon = 0.001$. Plotted is the nondimensional power spectral density of the unsteady c_l as a function of non-dimensional frequency, $St = fD/U_\infty$. The data is obtained on the Coarse mesh.

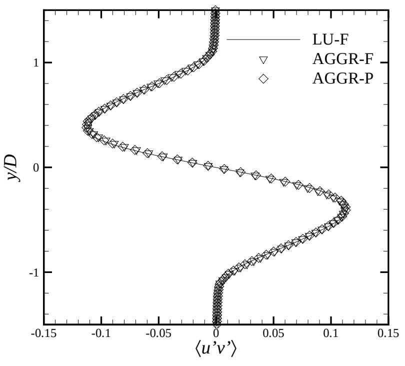
gence tolerance was tested and slightly shifted the iterative results towards the direct sparse solver. However, the reason for these differences is yet to be determined.

From these spatial and temporal comparisons, considering the direct solver partitioned in Fourier space to be the benchmark, the convergence tolerance of $\epsilon = 0.001$ is sufficient for all of the iterative solvers included in this study.

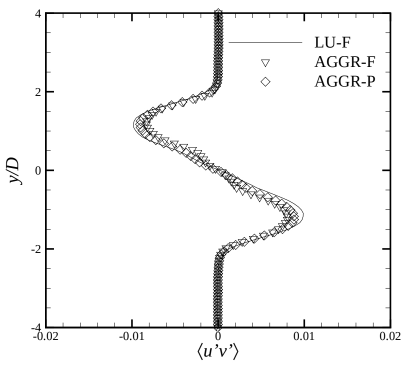
Some comments regarding the general behavior of the different approaches can be made from these verification calculations. First, the number of iterations required for convergence of the AMG solver typically doubled when using physical space partitioning. This is not unexpected, however, considering that inter-partition communication during the solve is necessary. Second, as expected the 0^{th} Fourier mode, which contains the majority of the energy and is most poorly conditioned, requires significantly more iterations to converge compared to the higher Fourier modes. However, because this Fourier mode's linear system contains only half as many unknowns, the computation time was still generally shorter. The number of iterations



(a)



(b)



(c)

Figure 3.5: Comparison of the mean flow field statistics computed using the direct linear solver and the iterative solvers with a convergence threshold of $\epsilon = 0.001$. (a) Time-averaged u velocity along a horizontal line through the center of the cylinder and extending into the wake region. (b) $u'v'$ Reynolds stress along a vertical line located $2D$ downstream of the center of the cylinder. (c) $u'v'$ Reynolds stress along a vertical line located $5D$ downstream of the center of the cylinder. All data is computed using the Coarse mesh.

Table 3.3: Iterations required to reach 0.001 relative convergence tolerance for each solver.

Solver	Iterations Required
AGGR-F	2
AGGR-P	3-4
ILUT-P	4-10

required to reach the convergence tolerance for the higher modes (i.e. Fourier modes other than the 0^{th}) are shown in Table 3.3.

3.7.2 Performance

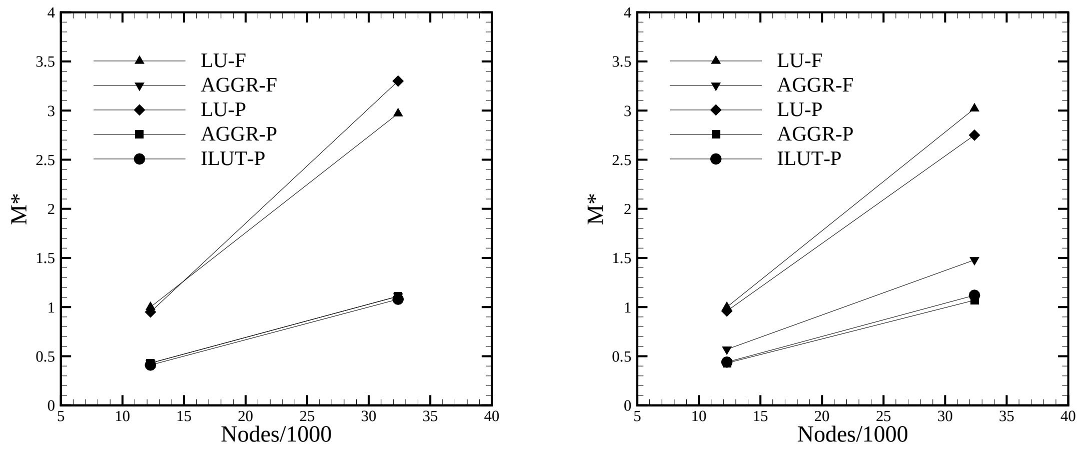
Two types of performance measures are analyzed. The first entails comparing the Coarse and Fine in-plane meshes, both containing 16 active Fourier modes. This provides comparisons when only the 2D mesh size, i.e. the number of unknowns in each linear system, is modified; hereafter referred to as mesh scaling. The second compares the Fine in-plane mesh with 16 active Fourier modes to the same mesh with 32 active Fourier modes, where the number of processors is also doubled from 8 to 16. In this case, the size of the linear systems remain constant, but the number of systems doubles, as does the number of processors; hereafter referred to as Fourier scaling. In each case memory and CPU time data are collected while running the DNS algorithm an additional 200 time steps beyond the 20 000 initial time steps.

3.7.3 Memory Usage

Memory usage is considered to be the total summed usage by all processors. Figure 3.6 shows the normalized memory usage for mesh scaling, according to

$$M^* = \frac{M}{M_0} \tag{3.10}$$

where M is the memory usage of the given method on a particular mesh, and M_0 is the memory usage of LU-F on the Coarse mesh for the particular computer hardware.



(a) (b)

Figure 3.6: Mesh Scaling — Memory usage normalized to the memory usage of the sparse direct LU solver (LU-F) on the Coarse mesh; (a) Linux cluster and (b) SGI.

Figure 3.7 shows the normalized memory usage for Fourier scaling, also normalized according to Eq. (3.10).

The sizable memory requirement of the direct sparse LU solver is clearly evident, even on the Coarse mesh. However, the memory usage of ILUT-P is no less than AGGR-P, independent of cluster, and only slightly less than AGGR-F on the shared memory machine. As expected, the rate at which memory usage increases with mesh size is also dramatically larger for the direct solvers. Also expectedly, when doubling the number of Fourier modes (Fourier scaling), essentially doubling the number of linear systems, the overall memory usage is also roughly doubled. Lastly, the memory usage is essentially the same for both clusters, but there are differences mainly for AGGR-F, using 33% more on the shared memory machine. This data is tabulated in Tables 3.4-3.6.

3.7.4 CPU Time

Two sets of CPU timings are given: those that reflect the linear solution phase only (excluding set-up, computing of explicit terms, etc.) and those that include total

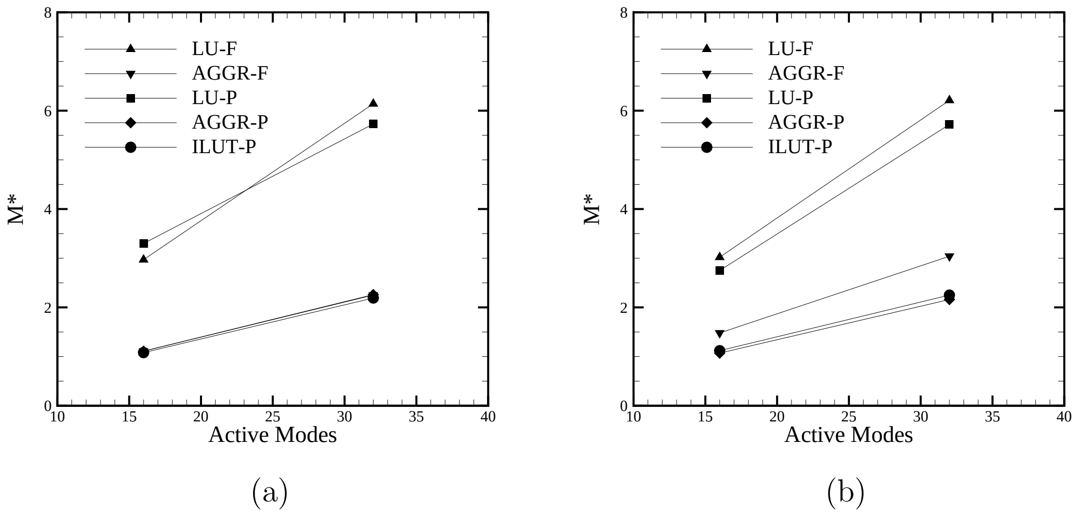


Figure 3.7: Fourier Scaling — Memory usage normalized to the memory usage of the sparse direct LU solver (LU-F) on the Coarse mesh; (a) Linux cluster and (b) SGI.

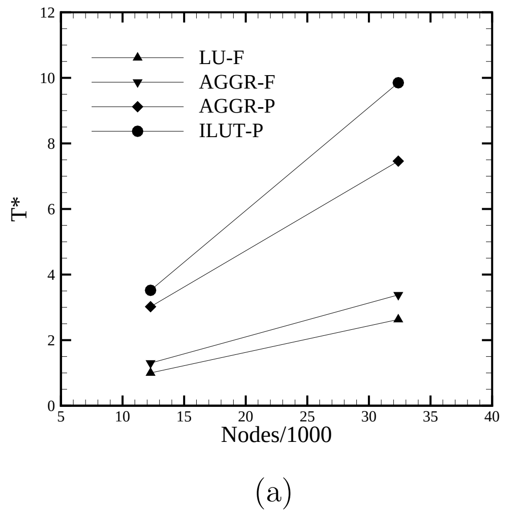
Table 3.4: Memory usage (double precision) and timings for the direct sparse LU solver (LU-F) on Linux and SGI machines for the Coarse mesh and 16 active Fourier modes.

Cluster	Memory Usage (MB)	Linear Solving Time (sec)	Time per Time Step (sec)
Linux	3,849	0.42	0.99
SGI	4,336	2.68	5.57

CPU time for the algorithm time step. Figure 3.8 shows the normalized linear solving time for mesh scaling, according to

$$T^* = \frac{T}{T_0} \tag{3.11}$$

where, again, T is the linear solving time of the given method on a particular mesh, and T_0 is the linear solving time of LU-F on the Coarse mesh for the particular cluster. Figure 3.9 similarly shows the CPU time per algorithm time step (including the solving time) for Fourier scaling, where again Eq. (3.11) is used for normalizing.



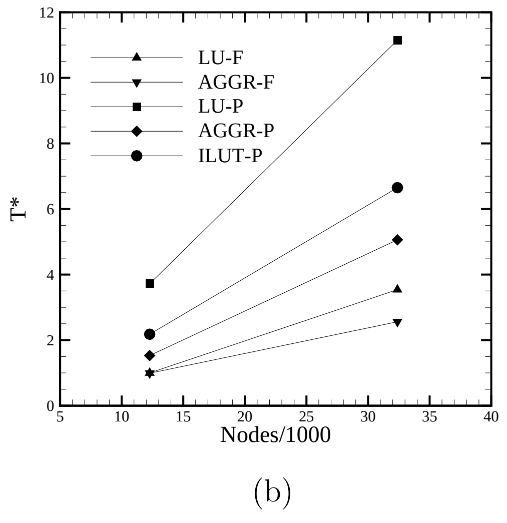
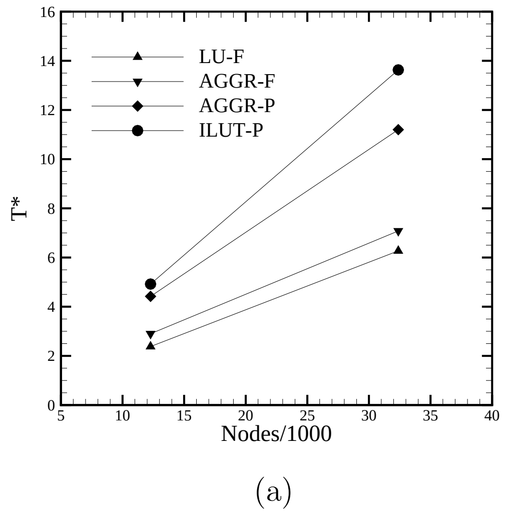


Figure 3.8: Mesh Scaling — Linear system solution time normalized to the solution time of the direct sparse LU solver (LU-F) on the Coarse mesh; (a) Linux and (b) SGI clusters.



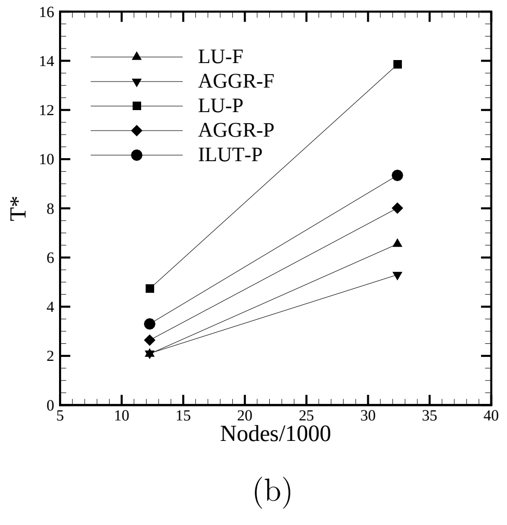


Figure 3.9: Mesh Scaling — Time per algorithm Time Step (including Solving Time) normalized to the solution time of the LU solver (LU-F) on the Coarse mesh; (a) Linux and (b) SGI clusters.

Table 3.5: Linux Memory usage normalized to memory usage of the direct sparse LU solver (LU-F) on the Coarse mesh.

Solver	Coarse	Fine (16)	Fine (32)
LU-F	1.00	2.97	6.14
AGGR-F	0.43	1.11	2.25
LU-P	0.95	3.30	5.73
AGGR-P	0.43	1.11	2.26
ILUT	0.41	1.08	2.19

Table 3.6: SGI Memory usage normalized to memory usage of the direct sparse LU solver (LU-F) on the Coarse mesh.

Solver	Coarse	Fine (16)	Fine (32)
LU-F	1.00	3.02	6.21
AGGR-F	0.57	1.48	3.04
LU-P	0.96	2.75	5.72
AGGR-P	0.43	1.07	2.16
ILUT	0.44	1.12	2.25

Comparing CPU time results for mesh scaling reveals large differences which may occur depending on the parallelization technique. When comparing just the CPU time spent solving the linear systems, in all instances LU-P is the slowest and increases at the highest rate. It should be noted LU-P is not included in the Linux time plots due to the enormous comparative computation time: nearly three orders of magnitude larger than any other solver examined! All the solvers improved in relative time from the distributed memory cluster to the shared memory machine, except LU-F on the Fine mesh. This is due to the shared memory architecture, since communication is less costly. In all instances, the partitioning in Fourier space method surpasses the partitioning in physical space method. The fastest overall solver was dependent upon the machine architecture, with LU-F performing best on the Linux cluster, and AGGR-F performing best on the shared memory SGI.

The total CPU time per algorithm time step expectedly behaves just like the linear system time regardless of the solver used since the remaining portion of the CFD algorithm remained the same. The majority of the algorithm CPU time not

Table 3.7: Linux Linear Solving CPU Time normalized to Linear Solving CPU Time of the direct sparse LU solver (LU-F) on the Coarse mesh.

Solver	Coarse	Fine (8)	Fine (16)
LU-F	1.00	2.63	2.74
AGGR-F	1.30	3.38	3.53
LU-P	1910	5250	9160
AGGR-P	3.02	7.46	11.8
ILUT-P	3.52	9.85	11.6

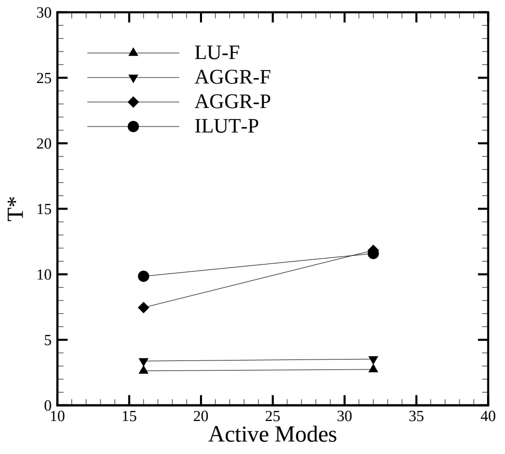
Table 3.8: SGI Linear Solving CPU Time normalized to Linear Solving CPU Time of the direct sparse LU solver (LU-F) on the Coarse mesh.

Solver	Coarse	Fine (8)	Fine (16)
LU-F	1.00	3.54	3.69
AGGR-F	0.99	2.56	2.64
LU-P	3.72	11.2	17.4
AGGR-P	1.53	5.06	6.75
ILUT-P	2.18	6.65	10.7

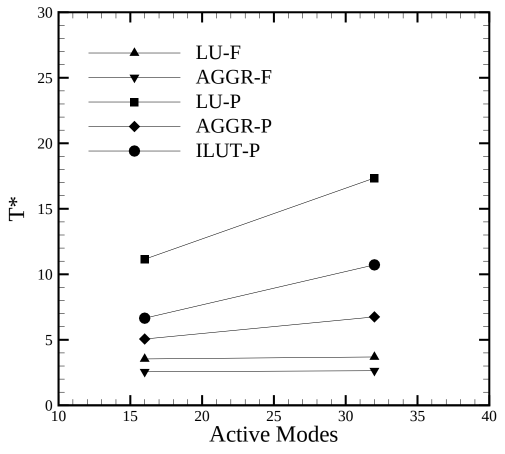
spent in solving the Fourier mode linear systems is spent calculating the convective terms in physical space. It should be noted that for the quickest solving method, the CPU time spent outside the linear solve accounts for over 50% of the total time. This data is tabulated in Tables 3.7/3.8.

Figure 3.10 shows the normalized linear system solution CPU time for Fourier scaling. The times are again scaled according to Eq. (3.11). Figure 3.11 similarly shows the normalized total CPU time per algorithm time step (including the solving time) for Fourier scaling. This data is tabulated in Tables 3.9/3.10.

Analyzing the differences for Fourier scaling reveals the advantage of partitioning in Fourier space. By doubling the number of linear systems and also doubling the number of processors to do the work, neglecting communication, essentially the same amount of work per processor should be required. This is actually not quite true, since the 0^{th} Fourier mode's linear system is half the size of the others. However, this would not affect partitioning in Fourier space, since the processor assigned to solve the smaller matrix would finish first and would wait for the others to finish. Also,

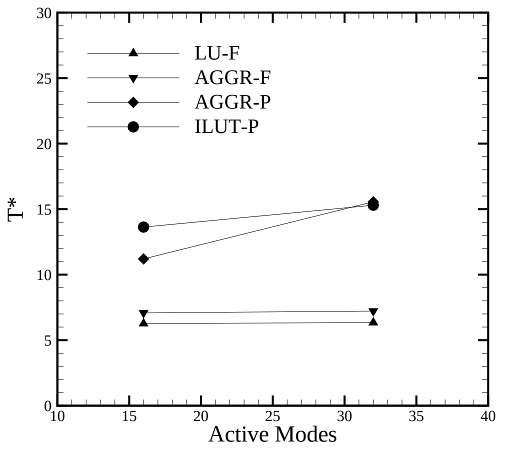


(a)

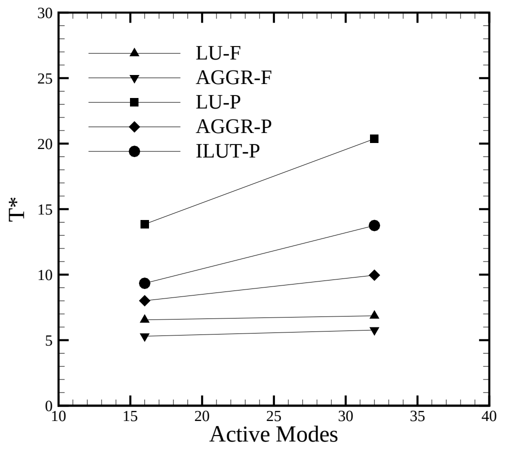


(b)

Figure 3.10: Fourier Scaling — Linear system solution time normalized to the solution time of the direct sparse LU solver (LU-F) on the Coarse mesh; (a) Linux and (b) SGI clusters.



(a)



(b)

Figure 3.11: Fourier Scaling — Time per algorithm Time Step (including Solving Time) normalized to the solution time of the direct sparse LU solver (LU-F) on the Coarse mesh; (a) Linux and (b) SGI clusters.

Table 3.9: Linux Time per algorithm Time Step normalized to the solving time of the direct sparse LU solver (LU-F) on the Coarse mesh

Solver	Coarse	Fine (8)	Fine (16)
LU-F	2.38	6.27	6.34
AGGR-F	2.90	7.08	7.22
LU-P	1910	5260	9170
AGGR-P	4.42	11.2	15.6
ILUT-P	4.92	13.6	15.3

Table 3.10: SGI Time per algorithm Time Step normalized to the solving time of the direct sparse LU solver (LU-F) on the Coarse mesh

Solver	Coarse	Fine (8)	Fine (16)
LU-F	2.08	6.55	6.86
AGGR-F	2.10	5.30	5.77
LU-P	4.73	13.9	20.4
AGGR-P	2.64	8.01	9.96
ILUT-P	3.30	9.34	13.8

since the higher Fourier mode linear systems are better conditioned, the additional linear systems should converge in a shorter amount of time than the lower Fourier mode linear systems. Therefore, as long as there are at least 4 active Fourier modes (2 linear systems), doubling the number of active Fourier modes and doubling the number of processors, neglecting communication, would ideally result in the same CPU time spent solving the linear systems.

Alternatively, for partitioning in physical space, the evaluation becomes more complex. Doubling the number of linear systems and doubling the number of processors not only causes more communication within each linear solve (since there are more processors to communicate with each other) but there are also more systems to solve. While the 0^{th} Fourier mode is half the size, the higher Fourier modes are better conditioned, so it is difficult to generally predict this effect.

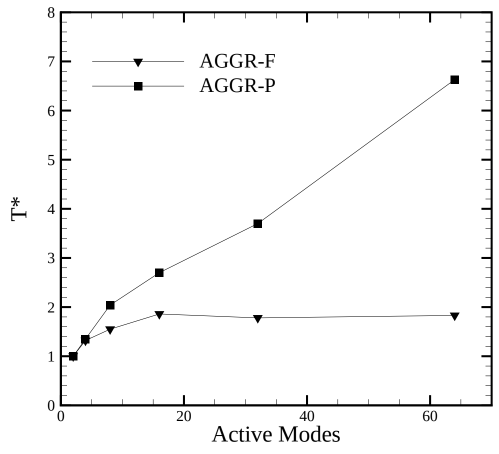
The data confirms the expected Fourier space partitioning behavior, since the CPU time does not increase significantly. The small increase that is seen could reasonably be due to increased amount of information required to be communicated before and after the linear system solves. The extra communication time in physical space partitioning obviously outweighs the better conditioned linear systems at the higher Fourier modes. This extra communication is further evidenced by the extra iterations per linear system solve required when doubling the number of processors.

Since memory usage is essentially the same independent of parallelization method, CPU time is the controlling variable in determining the better partitioning approach. In all the results thus far, partitioning in Fourier space is superior.

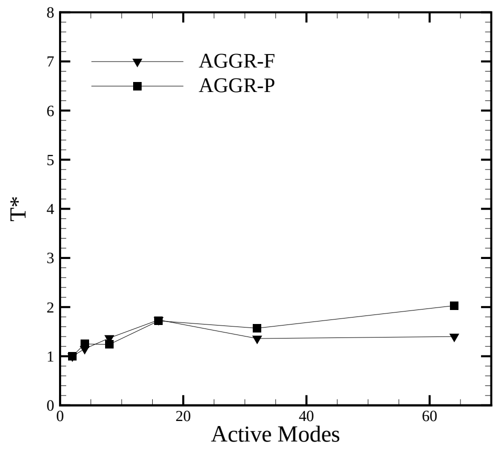
3.7.5 Scalability

The previous results indicate partitioning in Fourier space is superior to partitioning in physical space. However, due to the small number of data points, more evaluation is necessary. For these analyses, two scaling techniques were utilized. The first is an extension of Fourier scaling, in that both the Coarse and Fine meshes start with 2 active Fourier modes (yielding 1 linear system per algorithm time step) and utilize 1 processor. The problem is scaled by doubling both the number of Fourier modes (thus doubling the number of linear systems) and the number of processors for the problem. The second scaling technique uses the typical approach of solving a fixed problem using varying number of processors. This technique will be referred to as processor scaling. Both scaling methods were examined for both partitioning methods. Unfortunately, the processor scaling exceeded the memory capabilities of the (distributed memory) Linux cluster, so results are only available for the shared memory SGI machine.

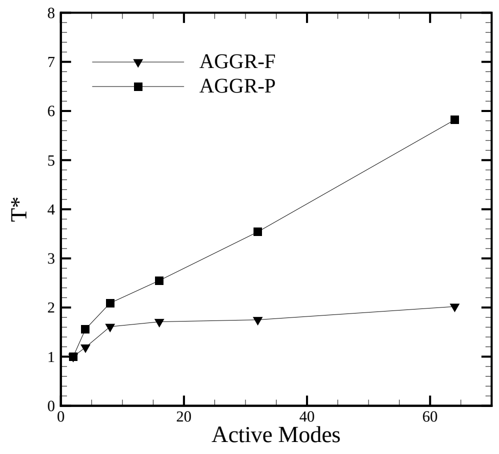
Figure 3.12 shows the normalized linear solving time for each solver when performing Fourier scaling. The results are normalized once again according to Eq. (3.11). However, this time T_0 is the CPU time per algorithm time step (including solving time) of the single processor applied to the 2D (1 linear system) problem. Tables 3.11/3.12 give the total CPU time per algorithm time step for the 2D problem for both in-plane meshes utilizing 1 processor. The CPU times for 1 processor and 2 active Fourier modes (1 linear system) should be equivalent since only one linear system is actually being solved, forcing it to be solved serially. Another important



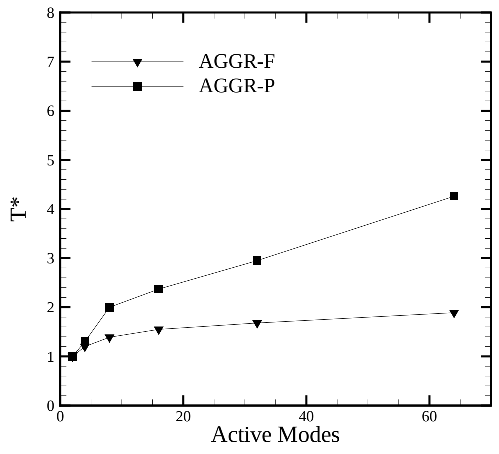
(a)



(b)



(c)



(d)

Figure 3.12: Fourier Scaling — Scalability of CPU Time per algorithm Time Step (including Solving Time) normalized to the CPU Time per algorithm Time Step of the direct sparse LU solver (LU-F) on the Coarse mesh; (a) Linux cluster Coarse mesh, (b) SGI machine Coarse mesh, (c) Linux cluster Fine mesh, and (d) SGI machine Fine mesh.

note to remember is that when there are only 2 active Fourier modes, only one linear system must be solved, the 0^{th} Fourier mode, which has only half the number of unknowns, making the overall system half the size.

Table 3.11: Linux actual CPU Time per Time Step (sec) for 2D problem utilizing 1 processor for AGGR-F and AGGR-P on both clusters.

Partitioning Method	Coarse	Fine
Fourier Space	0.65	1.72
Physical Space	0.68	1.83

Table 3.12: SGI actual CPU Time per Time Step (sec) for 2D problem utilizing 1 processor for AGGR-F and AGGR-P on both clusters.

Partitioning Method	Coarse	Fine
Fourier Space	3.23	9.19
Physical Space	4.11	9.06

Here, a clear advantage is seen in Fourier space partitioning, where the efficiency in solving additional Fourier modes (i.e. increased transverse resolution) is excellent. Partitioning in physical space does not scale as well, and unlike partitioning in Fourier space, CPU time continually increases when increasing the problem size (Fourier modes) despite also increasing the number of processors. While processor numbers increase with problem size and better conditioned linear systems are added, communication overrides these benefits. Another indication of this extra communication is again the increase in number of iterations required for each Fourier mode. Tables 3.13/3.14 show how the typical number of iterations required increases as the number of processors increases. It should be noted that for 1 processor the only Fourier mode active is the 0^{th} Fourier mode. Since this system, which normally takes more iterations than the other Fourier modes, requires only two, this further elucidates more processors cause more iterations for the same linear system.

Comparing the two parallelization methods reveals for Fourier scaling, partitioning in Fourier space is less computationally expensive than partitioning in physical space and scales better for all problems examined. This advantage is also only exaggerated as the problem size increases.

When comparing the Fourier scaling across platforms, the Fourier space partitioning performs essentially the same, while physical space partitioning scales better

Table 3.13: Fourier Scaling — Typical Number of Iterations required for Physical Space partitioning for Coarse Mesh.

Processors	Iterations
1	2
2	3
4	3
8	3-4
16	3-4
32	4

Table 3.14: Fourier Scaling — Typical Number of Iterations required for Physical Space partitioning for Fine Mesh.

Processors	Iterations
1	2
2	2
4	3
8	3-4
16	4
32	5

on the shared memory machine due to faster communication during the linear system solves.

Figure 3.13 shows the application speedup for processor scaling, defined by

$$\mathit{Speedup} = \frac{T_0}{T} \tag{3.12}$$

where, again, T_0 is the time per algorithm time step of the given problem using a single processor and T is the time per algorithm time step of the given problem for a particular number of processors. Ideal speedup would be a straight line where the number of processors is equal to the speedup. Speedup is shown for the three main mesh/Fourier mode problems by only increasing the number of processors utilized for a given problem. As previously mentioned, due to memory capability limitations of the Linux cluster, which arose when attempting to solve the full problem using a single processor, this analysis is limited to the shared memory cluster. Also, when

Table 3.15: SGI Actual Time per algorithm Time Step (sec) for processor scaling utilizing 1 processor for the AGGR solver using both partitioning methods.

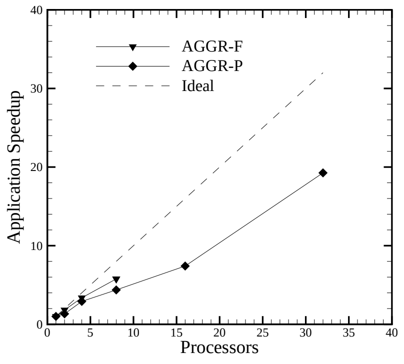
Solver	Coarse	Fine (16)	Fine (32)
AGGR-F	32.4	105	200
AGGR-P	30.9	99.1	216

partitioning in Fourier space, the number of processors is limited to half the number of active Fourier modes. Partitioning in physical space does not have this restriction, thus the plots do not contain an equal number of data points.

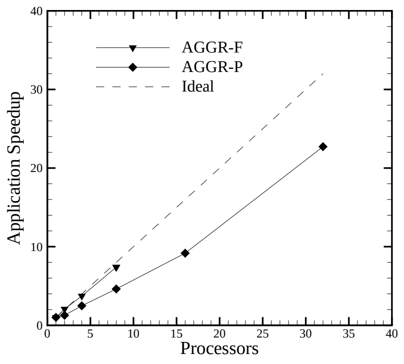
As in all previous data, partitioning in Fourier space scales better than partitioning in physical space for processor scaling. An interesting note is that for the amount of data shown, the results remain fairly linear. Also, in practice, continuing to increase the number of processors when partitioning in physical space would eventually start to reduce the speedup due to large amounts of communication during the solving time. This result would also be expected when partitioning in Fourier space, due to communication before and after the solve, but previous data implies this is less important. From these results it appears these issues are not quickly realized for either partitioning method. Despite Fourier partitioning's obvious advantage in efficiency, because of the limitation on the maximum number of processors that can be used (one-half the number of active modes) it is not always the fastest method available. In all cases, doubling the number of processors past this limit resulted in a slightly faster overall solution time.

3.8 Conclusion

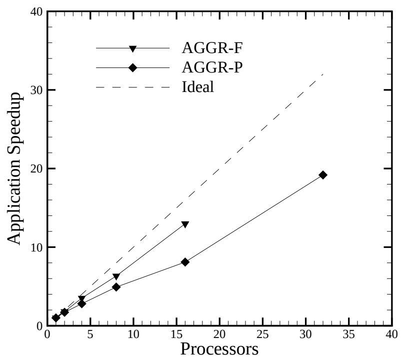
This study analyzed the two parallelization techniques which naturally arise from the hybrid Spectral/Finite-Element discretization technique first implemented for DNS/LES CFD solvers by Snyder and Degrez [11]. In one approach, multiple simultaneous instances of a serial linear solver are used while the other consists of multiple sequential calls to a parallel solver. These methods were compared based upon CPU time and memory usage.



(a)



(b)



(c)

Figure 3.13: Processor Scaling — Application speedup according to single processor CPU time obtained on SGI machine; (a) Coarse Mesh with 16 active Fourier Modes, (b) Fine Mesh with 16 active Fourier Modes, and (c) Fine mesh with 32 active Fourier Modes.

While memory usage was fairly consistent for both partitioning methods, for all instances, partitioning in Fourier space was more efficient and computationally less expensive than partitioning in physical space. In terms of CPU time, the sparse LU algorithm worked well when applied to partitioning in Fourier space but when used in conjunction with partitioning in physical space it was almost three orders of magnitude slower than any other algorithm on the distributed memory Linux cluster. AMG for either partitioning method proved efficient.

Chapter 4

Conclusion

The research presented in this thesis has analyzed the computational performance of two parallelization techniques naturally arising from the described Spectral/Finite Element discretization method. One parallelization approach partitions in Fourier space for the solving phase, where simultaneous multiple instances of a serial solver are employed. The second approach partitions in physical space for all phases of the algorithm and has multiple sequential calls to a parallel solver.

Through serial implementation of the code, classical point-wise AMG, smoothed aggregation AMG, single-level ILUT preconditioned GMRES, and direct sparse LU were compared for a 2D problem. Fully 3D parallelized calculations of the code were implemented for comparisons between partitioning methods using smoothed aggregation AMG, single-level ILUT preconditioned GMRES, and direct sparse LU. By considering the time-dependent solution signal as well the zeroth and first order time-averaged solutions, it was found that a relative convergence threshold of $\epsilon = 0.001$ was sufficient for all iterative methods when applied to either partitioning method. The single-level ILUT solver demonstrated linear scaling in terms of memory consumption, but had significant increases in computational time. The sparse LU solver's memory consumption expectedly scaled super-linearly. However, interestingly, the computational time scaled linearly. Both pointwise and smoothed aggregation AMG solvers had nearly linear scaling for both memory usage and computation time. While the pointwise AMG solver outperformed the smoothed aggregation AMG solver, due to the considerable amount of fine-tuning of parameters required to obtain these results, no definite conclusions can be drawn.

Comparing parallelization techniques of Fourier and physical space partitioning revealed distinct advantages of partitioning in Fourier space. As expected, memory usage of the sparse LU solver was higher than any other method, while the aggregated AMG and single-level ILUT solvers were comparable to each other. In terms of CPU time, the smoothed aggregation AMG and single-level ILUT solvers performed fairly well for both parallelization techniques. The sparse LU solver performed well only when partitioning in Fourier space; physical space partitioning performance was much worse, which was further exaggerated for the distributed memory machine.

Partitioning in Fourier space also scaled better in terms of CPU cost than physical space partitioning, especially for higher numbers of Fourier modes. These differences were more apparent on the distributed memory machine compared to the shared memory machine. However, utilizing the limiting maximum number of processors in Fourier space partitioning (half the number of active Fourier modes) was not the fastest method. Physical space partitioning was able to decrease total CPU time by doubling the number of processors for the same problem. However, this is obviously less efficient.

Bibliography

[1] J. W. Demmel, S. C. Eisenstat, J. R. Gilbert, X. S. Li, and J. W. H. Liu, "A supernodal approach to sparse partial pivoting," *SIAM J. Matrix Analysis and Applications*, vol. 20, no. 3, pp. 720–755, 1999. 4, 30, 56

[2] X. S. Li and J. W. Demmel, "SuperLU DIST: A scalable distributed-memory sparse direct solver for unsymmetric linear systems," *ACM Trans. Mathematical Software*, vol. 29, no. 2, pp. 110–140, June 2003. 4, 57

[3] W. L. Briggs, V. E. Henson, and S. F. McCormick, *A Multigrid Tutorial*, 2nd ed. SIAM, 2000. 5, 13

[4] Y. Saad, "Krylov subspace methods for solving large unsymmetric linear systems," *Mathematics of Computation*, vol. 37, no. 155, pp. 105–126, July 1986. 5

[5] L. N. Trefethen and D. B. III", *Numerical Linear Algebra.* Society for Industrial and Applied Mathematics, 1997. 5, 7, 9, 14

[6] Y. Saad and M. Schultz, "A generalized minimal residual algorithm for solving nonsymmetric linear systems," *Journal on Scientific and Statistical Computing*, vol. 7, pp. 856–869, 1986. 7, 33, 57, 58

[7] Y. Saad, *Iterative Methods for Sparse Linear Systems.* PWS Publishing Company, 1996. 9

[8] M. Griebel, T. Neunhoeffer, and H. Regler, "Algebraic multigrid methods for the solution of the Navier-Stokes equations in complicated geometries," *International Journal for Numerical Methods in Fluids*, vol. 26, pp. 281–301, 1998. 11, 13, 23, 24

[9] R. D. Lonsdale, "An algebraic multigrid solver for the Navier-Stokes equations on unstructured meshes," *International Journal of Numerical Methods for Heat and Fluid Flow*, vol. 3, pp. 3–14, 1993. 13, 23

[10] D. O. Snyder, "A parallel finite-element/spectral LES algorithm for complex two-dimensional geometries," Ph.D. dissertation, Utah State University/von Karman Institute for Fluid Dynamics, Logan, Utah, USA, January 2002. 15, 25, 50

[11] D. O. Snyder and G. Degrez, "Large-eddy simulation with complex 2-D geometries using a parallel finite-element/spectral algorithm," *International Journal for Numerical Methods in Fluids*, vol. 41, no. 10, pp. 1119–1135, 2003. 15, 25, 27, 45, 49, 50, 75

[12] J. W. Ruge and K. Stüben, “Algebraic multigrid,” in *Multigrid Methods*, S. F. McCormick, Ed. Philadelphia: SIAM, 1987. 22, 23, 32

[13] V. Venkatakrishnan and D. Mavriplis, “Agglomeration multigrid for the three-dimensional Euler equations,” Institute for Computer Applications in Science and Engineering (ICASE), Tech. Rep. TR-94-5, 1994. 22

[14] P. Vanek, J. Mandel, and M. Brezina, “Algebraic multigrid based on smoothed aggregation for second and fourth order problems,” *Computing*, vol. 56, pp. 179–196, 1996. 22, 33, 57

[15] R. Webster, “An algebraic multigrid solver for Navier-Stokes problems,” *Internation Journal for Numerical Methods in Fluids*, vol. 18, no. 8, pp. 761–780, 1994. 23

[16] M. Raw, “Robustness of coupled algebraic multigrid for the Navier-Stokes equations,” AIAA, Reno, Nevada, Technical Paper 1996-0297, January 1996. 23

[17] K. Stüben, “A review of algebraic multigrid,” *Journal of Computational and Applied Mathematics*, vol. 128, pp. 281–309, 2001. 24

[18] J. M. Weiss, J. P. Maruszewski, and W. A. Smith, “Implicit solution of preconditioned Navier-Stokes equations using algebraic multigrid,” *AIAA Journal*, vol. 37, no. 1, pp. 29–36, 1999. 24

[19] D. Mavriplis, “An assessment of linear versus nonlinear multigrid methods for unstructured mesh solvers,” *Journal of Computational Physics*, vol. 175, no. 1, pp. 302–325, 2003. 24

[20] P. M. de Zeeuw, “Matrix dependent prolongations and restrictions in a black-box multigrid solver,” *Journal of Computational and Applied Mathematics*, vol. 33, pp. 1–27, 1990. 24

[21] H. Guillard and P. Vanek, “An aggregation multigrid solver for convection-diffusion problems on unstructured meshes,” University of Denver, Denver, Colorado, Report 130, 1998. 24

[22] Y. Notay, “A robust allgebraic preconditioner for finite difference approximations of convection-diffusion equations,” Service de Métrologie Nucléaire, Université Libre de Bruxelles, Brussels, Belgium, Report GANMN 99–01, 1999. 24

[23] A. Reusken, “On the approximate cyclic reduction preconditioner,” *SIAM Journal on Scientific Computing*, vol. 21, pp. 565–590, 2000. 24

[24] T. E. Tezduyar, S. Mittal, S. E. Ray, and R. Shih, “Incompressible flow computations with stabilized bilinear and linear equal-order interpolation velocity-pressure elements,” *Computer Methods in Applied Mechanics and Engineering*, vol. 95, no. 2, pp. 221–242, 1992. 26, 52

[25] K. Jansen, “A stabilized finite element method for computing turbulence,” *Computer Methods in Applied Mechanics and Engineering*, vol. 174, no. 3–4, pp. 299–317, 1999. 27, 52

[26] A. E. Tejada-Martinez and K. E. Jansen, “On the interaction between dynamic model dissipation and numerical dissipation due to streamline upwind/Petrov-Galerkin stabilization,” *Computer Methods in Applied Mechanics and Engineering*, vol. 194, no. 9-11, pp. 1225–1248, 2005. 27, 52

[27] G. Karypis and V. Kumar, “METIS: A software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices, version 4.0,” University of Minnesota, Twin Cities, Minnesota, Manual, 1998. 28, 54

[28] D. Vanden-Abeele, G. Degrez, and D. O. Snyder, “Parallel turbulent flow computations using a hybrid spectral/finite-element method on Beowulf clusters,” in *Proceedings of ICCFD3*, Toronto, Canada, 2004. 30, 50, 55

[29] J. Irwin, J.-M. Loingtier, J. R. Gilbert, G. Kiczales, J. Lamping, A. Mendhekar, and T. Shpeisman, “Aspect-oriented programming of sparse matrix code,” in *Proceedings International Scientific Computing in Object-Oriented Parallel Environments (ISCOPE)*, Marina del Rey, CA, USA, 1997. 31, 56

[30] T. Clees, “AMG strategies for PDE systems with applications in industrial semiconductor simulation,” Ph.D. dissertation, University of Cologne, Germany, November 2004. 31, 32

[31] G. L. G. Sleijpen and H. A. van der Vorst, “An overview of approaches for the stable computation of hybrid BiCG methods,” *Applied Numerical Mathematics: Transactions of IMACS*, vol. 19, no. 3, pp. 235–254, 1996. 32

[32] K. Stüben, “An introduction to algebraic multigrid,” in *Multigrid*, U. Trottenberg, C. Oosterlee, and A. Schueller, Eds. Oxford, UK: Academic Press, 1999, pp. 413–532. 32

[33] M. A. Heroux and J. M. Willenbring, “Trilinos user’s guide,” Sandia National Laboratories, Albuquerque, NM, Tech. Rep. SAND2003-2952, 2003, trilinos. 33, 57, 58

[34] M. Sala, J. J. Hu, and R. S. Tuminaro, “ML 3.1 smoothed aggregation user’s guide,” Sandia National Laboratories, Tech. Rep. SAND2004-4819, 2004. 33, 57

[35] P. Vanek, “Acceleration of convergence of a two level algorithm by smooth transfer operators,” *Applied Mathematics*, vol. 37, pp. 265–274, 1992. 33, 57

[36] R. S. Tuminaro and C. Tong, “Parallel smoothed aggregation multigrid: aggregation strategies on massively parallel machines,” in *Supercomputing 2000 Proceedings*, Dallas, TX, USA, 2000, trilinos. 33, 57

[37] M. T. Jones and P. E. Plassman, “A parallel graph coloring heuristic,” *SIAM Journal on Scientific Computing*, vol. 14, pp. 654–669, 1993. 33, 57

[38] M. V. Morkovin, “Flow around a circular cylinder—a kaleidoscope of challenging fluid phenomenon,” in *Proceedings of the ASME Symposium on Fully Separated Flows*, Philadelphia, PA, 1964, pp. 102–119. 34, 58

[39] E. Berger and R. Wille, “Periodic flow phenomena,” *Annual Review of Fluid Mechanics*, vol. 4, pp. 313–340, 1972. 34, 58

[40] P. Beaudan and P. Moin, “Numerical experiments on the flow past a circular cylinder of sub-critical Reynolds number,” Department of Mechanical Engineering, Stanford University, Tech. Rep. TF-62, 1994. 34, 58

[41] C. H. K. Williamson, “Vortex dynamics in the cylinder wake,” *Annual Review of Fluid Mechanics*, vol. 28, pp. 477–539, 1996. 34, 58

[42] D. C. Sternel and M. Schäfer, “Efficient large eddy simulations in complex geometries by using an improved multigrid algorithm,” in *EMG 2005 Proceedings*, Scheveningen, the Netherlands, 2005. 36

[43] C. H. K. Williamson, J. Wu, and J. Sheridan, “Scaling of streamwise vortices in wakes,” *Physics of Fluids*, vol. 7, pp. 2307–2309, 1995. 44

[44] G. Larson, D. Snyder, D. V. Abeele, and T. Clees, “Application of single-level, pointwise algebraic, and smoothed aggregation multigrid methods to direct numerical simulations of incompressible turbulent flows,” *Computing and Visualization in Science*, 2006 (In Press). 50, 60

[45] J. W. Demmel, J. R. Gilbert, and X. S. Li, “An asynchronous parallel supernodal algorithm for sparse gaussian elimination,” *SIAM J. Matrix Analysis and Applications*, vol. 20, no. 4, pp. 915–952, 1999. 57