ADVANCING ANALYSIS CAPABILITIES IN ANSYS THROUGH SOLVER TECHNOLOGY

GENE POOLE , YONG-CHENG LIU[*], AND JAN MANDEL[†]

May 2001, Revised October 2001

Abstract. This paper describes substantial improvements in analysis capabilities in a large scale commercial finite element program made possible by the implementation of solver technology. The ANSYS program is a commercial finite element analysis program which has been in use for thirty years. The original code, developed around a direct frontal solver has been expanded over the years to include full featured pre- and post- processing capabilities which support a comprehensive list of analysis capabilities including linear static analysis, multiple nonlinear analyses, modal analysis and many other analysis types. The finite element models on which these analyses are used have continued to grow in size and complexity. This growth in size and complexity has been both enabled by and dependent on new solver technology along with increased computer memory and CPU resources. Beginning in 1994 ANSYS added a Jacobi preconditioned conjugate gradient solver (JCG) and subsequently an Incomplete Cholesky Preconditioned Conjugate Gradient solver (ICCG) to improve thermal analysis capabilities. In recent years the addition of the Boeing sparse matrix library for modal and static analysis, and a proprietary preconditioned conjugate gradient solver as well as additional iterative solvers to support new CFD capabilities have greatly increased the number of solver options available in ANSYS. Most recently, in version 5.7, ANSYS has added a new domain decomposition solver for solving very large structural analysis solutions on distributed MPI-based computer systems and the newest iterative solver option, an algebraic multi-grid iterative solver (AMG).

This paper will describe implementation considerations for the addition of new solver technology to a large legacy code, compare resource requirements for the various solver choices and present some comparative results from several customer generated problems. The AMG solver benefits, both in improved robustness and parallel processing efficiency will be described. The paper will also discuss some of the implementation challenges that have been overcome to add new solver technology to a large existing code. The role of solver technology in meeting current and future demands of large scale commercial analysis codes will be discussed.

Key words. Finite Elements, elasticity, iterative solvers, Algebraic multigrid

1. Introduction.

Over a 30 year history of development the ANSYS program has evolved from an original finite element code designed for computers with limited memory to solve modest problems with several thousand equations to the current software system which is routinely used to solve problems with hundreds of thousands to several million equations. ANSYS software is a family of engineering analysis tools including DesignSpace, ANSYS/Professional and ANSYS/MultiPhysics. These analysis tools are used to model, analyze and simulate systems in structural, thermal, mechanical, computational fluid dynamics, electromagnetics, and other applications. The solution of linear systems of equations is the dominant computational step for most, if not all, analyses of very large finite element models in ANSYS. This paper describes solver technology improvements in ANSYS which have enabled the growth in analysis capabilities.

The outline of the paper is as follows. First, some motivating factors for solver improvements are discussed. Then a chronological description of solver improvements in ANSYS is given followed by a more detailed description of the newest iterative solver added to ANSYS, the AMG solver, an algebraic multigrid solver. Next, implementation considerations are discussed in the context of integrating new solver technology into an existing program. Results demonstrating ANSYS solver usage and

[*] ANSYS, Inc., 275 Technology Drive, Canonsburg, PA 15317
[†]Department of Mathematics, University of Colorado at Denver, Denver CO 80217-3364

relative performance are presented. Finally, a summary and discussion of future solver directions is given.

2. Motivation for solver improvements. The changes to ANSYS equation solvers have been motivated by several factors, which often compete for attention. Larger memory in computer systems coupled with greatly improved automatic meshing capabilities has increased the detail and size of finite element models routinely analyzed in recent years. These models lead to linear systems of equations that are now typically over 100,000 degrees of freedom (DOFs). The software capability exists to generate even larger models with millions of degrees of freedom but solver limitations often limit the practicality of such large systems.

Time to solution is a second competing demand on solver resources. No matter how large the linear systems become, engineers desire turnaround time of minutes rather than days for solution of routine problems. Most users expect interactive response while analyses that run overnight or longer are confined to the largest challenge problems.

Computer hardware changes also produce competing demands on solver resources. Though memory sizes and disk capacity have both increased dramatically in recent years the growth has not always been sufficient to meet the demands of the largest problems, particularly for direct methods. This problem is still most acute for desktop NT workstation users who are usually limited to 2 Gbytes of user address space and often are given systems with well under 2 Gbytes of real physical memory. In addition, large capacity disks on NT workstations are available but are often so much slower than the processors that the largest jobs are completely I/O dominated. While it is routine today to find computer processors capable of computing hundreds of Mflops to even Gflop performance on computational kernels, delivering sustained peak performance to solve the problems of large scale applications codes remains a challenge for hardware and software developers.

In addition to problem size growth, response time and hardware changes, robust and comprehensive analysis capabilities also motivate solver improvements. For example, large modal analyses were in the past limited by computer resource requirements. Solution techniques such as the Guyan Reduction method solved the larger problem by forming a reduced system, solving the smaller linear system and expanding the results back to approximate the solution of the full model. Solver improvements and increased computer resources have enabled modal solution of the full model improving accuracy and robustness. Nonlinear analyses, with both material and geometric nonlinearities, with and without contact surfaces often provide challenges for iterative solvers due to potentially ill-conditioned systems. Continual improvements in pre-conditioners have extended the usefulness of iterative solvers.

Many of these motivating factors produce competing demands for the solvers. Increasing problem size while reducing or maintaining analysis time to solution is an obvious example. Solver changes that reduce memory usage may also increase disk usage and time to solution. Iterative solvers may provide a faster method to solve linear systems but are not as robust as direct solvers nor as accurate in all cases for modal analyses. The competing demands are satisfied in a large-scale code by adding an array of high performance solvers. The collection of solvers becomes the enabling engine for robust solution methods to meet the analysis demands.

3. Description of linear solvers in ANSYS. This section describes chronologically the addition of solver technology to the ANSYS program. A brief description of each solver is given along with a discussion of the factors motivating each solver addition and the analysis capabilities improved by each solver change.

3.1. ANSYS frontal solver. The original solver used in ANSYS is the frontal solver, a direct Cholesky factorization algorithm [5, 6]. This direct factorization method was designed for computers with very limited memory but remained an efficient direct method for many years. The frontal solution is combined with element assembly in a manner that requires only the current active front to be in memory at any time during factorization. However the limitations of frontal solvers for very large problems are very well known. Memory requirements for the frontal solver are $O(max\_wavefront^2/2)$ and the total floating point operations increase as $n \times rms\_wavefront^3$. Here $rms\_wavefront$ refers to average, or root mean square, bandwidth. The disk storage requirements for the frontal solver also become prohibitive as the model size grows. The factored matrix disk file will be $n \times rms\_wavefront$ in double precision words. For example, a 100,000 DOF matrix with an rms_wavefront of 5,000 would require 500 million double precision (DP) words or approximately 4 Gbytes. A 1 million DOF matrix with an rms wavefront of 15,000 would require over 114 Gbytes! These memory and disk requirements for the frontal solver clearly demonstrate the need to reduce both in order to solve routine ANSYS analyses. The computational kernels for the frontal solver are very efficient on most computer architectures and run at near peak performance with modest optimizations. However, the number of floating point operations required for this algorithm becomes prohibitive even at sustained flop rates of several hundred Mflops (million flops per second) to a few Gflops (billion flops per second).

3.2. Iterative solvers added to ANSYS. Iterative solvers offer a potential advantage over direct factorization based solvers for two main reasons. First, since linear systems in finite element applications typically have well under 100 non-zeros per row and can take advantage of symmetric storage schemes, the disk storage requirements are far less than for direct solvers. Indeed, in most cases the entire sparse linear system and preconditioner is in memory. More importantly, the computational cost for iterative solvers grows linearly with problem size, provided the iterative method converges independently of the size of the problem. Iterative solvers were introduced to the ANSYS program with a Jacobi preconditioned conjugate gradient solver (JCG) used primarily for thermal analyses. The thermal elements have a single thermal degree of freedom per node and the matrices are very well conditioned. The initial JCG implementation used a sparse matrix file formed from the frontal assembly process. This process allowed existing element assembly and solution logic to remain relatively unchanged and at the same time benefitted from the algorithmic improvements from the iterative JCG solver. The JCG iterative method is the simplest in a class of preconditioned conjugate gradient methods. It is not, however, robust enough to replace the frontal solver. In many structural analysis problems the JCG method will fail to converge.

Subsequent to the JCG solver several standard preconditioned conjugate gradient methods were added including ICCG, ILU, BICG and GMRES. Real and complex versions of each iterative solver are implemented in ANSYS and are used in a variety of analysis types including thermal/fluid coupled field analyses, acoustic/thermal analyses and electro-magnetic analyses. All of these early iterative solvers used the frontal solver assembly process with a sparse matrix file interface. While each of the iterative solvers performs well on selected problems, none of these initial solvers were robust enough to replace the frontal solver for general structural analysis applications.

NOTE: ILU, BICG and GMRES real or complex solvers are accessed through a single equation solver label, ICCG, for simplicity of input.

3.3. ANSYS PowerSolver. The use of iterative solvers in ANSYS took a dramatic step forward in 1995 in version 5.1. A new element based preconditioned conjugate gradient method (PCG) was added to ANSYS that is robust and runs as a true "black-box" iterative solver [11]. The solver is licensed from Computational Applications and Systems Integration (CA&SI) and includes a sparse matrix assembly process featuring efficient use of memory and I/O resources. Details of the preconditioner are proprietary. The general form of the preconditioner is based on the underlying physics of the problem and is provably good for all problems where the matrix arises from the discretization of a continuous problem. While the size of the preconditioner varies, it is generally smaller than the input matrix and also saves memory by using single precision for the preconditioner. Further memory savings are possible for certain commonly used elements by using an element by element matrix-vector multiplication implementation, thereby eliminating the requirement to store the assembled stiffness matrix. This solver is referred to as the PowerSolver, or PCG method, within ANSYS. It can be used on many structural analysis applications and is particularly effective for models using shell elements. It is, however, sensitive to some forms of ill-conditioning caused by poor aspect ratios in solid elements and is also sensitive to ill-conditioning caused by the use of shell elements attached to solid elements. The PowerSolver combines the benefits of linear scaling of solution time as problem size increases, robust preconditioning, and greatly reduced I/O and has replaced the frontal solver as the most often used solver in many cases, particularly on NT computers with limited I/O performance.

3.4. Sparse direct methods in ANSYS. New and improved modal analysis capabilities were added in ANSYS 5.3 by incorporating the Boeing Computer Services (BCS) block Lanczos solver [1]. This state-of- the-art Lanczos solver includes a sparse matrix direct solver and features an efficient implementation, which is mathematically robust and uses I/O and memory adaptively to fit the problem size to existing memory resources. An interface was developed within ANSYS, which allowed the sparse direct solver to be used as an alternative to the frontal solver. The BCS sparse solver has all the advantages of robust solution of linear systems by direct methods and is now used in ANSYS as the default direct solver. The computational kernels in the BCS solver have been optimized for most hardware platforms using calls to the BLAS3 matrix-matrix routine DGEMM and BLAS2 matrix-vector routine DGEMV. Complex and non-symmetric versions of this solver are also available in ANSYS. The current implementation of this solver uses the frontal assembly interface in most cases but sparse assembly options are also available and are under development. Sparse assembly will reduce potential memory limitations for very large problems which occur using the frontal assembly process, particularly for problems which contain hundreds or thousands of constraint equations while also reducing matrix assembly time.

3.5. Latest iterative solvers in ANSYS 5.7. Two new iterative solvers were added to ANSYS 5.7 that offer new parallel capabilities and further improvement of robustness in preconditioning. The domain decomposition solver (DDS), a distributed memory parallel solver based on the FETI method [2, 3], solves very large problems on large single-image shared memory machines running MPI as well as loosely coupled networks of workstations and clusters running MPI in an NT or Unix environment. The domain solver operates seamlessly from an ANSYS execution by communicating via a file interface. Linear scaling of the domain solver has been observed on UNIX and NT platforms. Further parallelization is planned for future releases to enable scaling of the entire solution time. The current implementation of DDS uses a skyline solver within the subdomains.

A second parallel iterative solver has been added in ANSYS 5.7 that gives improved convergence for ill-conditioned problems and better parallel scaling. The new parallel iterative solver is a multi-level algebraic multi-grid solver (AMG). The AMG solver uses a sparse matrix assembly interface with direct elimination of constraint equations discussed in Section 5.3. The efficient sparse assembly coupled with a robust and parallel preconditioner combine to offer substantial improvements over the PCG solver in many cases.

4. Description of the AMG solver. The AMG solver in ANSYS is the Smoothed Aggregation Algebraic Multigrid method, with the specific selection of algorithm components guided by powerful heuristics. The implemented method exploits a number of opportunities for applying computational effort in parts of the problem to achieve fast convergence for hard problems. The actual algorithms used are selected automatically. The result is a powerful adaptive method that is lean and fast on easy problems and tough though justifiably expensive on hard ones.

The implemented algorithm is based on the method of Vaněk, Mandel, and Brezina [15], with further improvements by Vaněk and Mandel, not published previously. For theoretical analysis of convergence of a simple version of the algorithm, see [14]. See [10, 15] for a motivation and justification of the construction of the prolongation matrix P. The construction of P by smoothed aggregation was introduced by Vaněk [12, 13] for the case of scalar problems and generalized to elasticity by Vaněk, Mandel, and Brezina [15]. The choice of the smoothing blocks to follow dominant directions is similar as in a method by Mandel for p-version finite elements with high aspect ratios [9, 8]. The general scheme for using heuristics to selectively apply computational power to parts of the problem to build a preconditioner is from the work of Mandel on p-version elements [8, 7].

4.1. Basic multilevel scheme. The system to be solved is $Ax = b$ with A a real, $n \times n$, symmetric, positive definite matrix. The AMG algorithm as implemented in ANSYS requires the input of the matrix A, the load vector b, the $n \times 6$ matrix of rigid body modes Z, the node-degree of freedom adjacency, and the Cartesian coordinates of the nodes. The rigid body modes are calculated by ANSYS from the definition of the elements, and satisfy the equation $(AZ)_i = 0$ except at degrees of freedom i constrained by boundary conditions.

The overall algorithm is Preconditioned Conjugate Gradients, with preconditioning given by one multigrid cycle. A two-level multigrid algorithm is used as follows. Lowercase greek letters ω, ρ, σ,..., are parameters. The core of the algorithm is $n \times m$ matrix P, $m < n$, called a *prolongation*. The space $\mathbb{R}^m$ is called the *coarse space*, and $\mathbb{R}^n$ is *fine space*.

Algorithm 4.1. *Input:* f, x_0. *Output:* x.

```
1. $x \leftarrow x_0$ (initial approximation)
2. $x \leftarrow x - S(Ax - f)$ (pre-smoothing)
3. $r = P^T(b - Ax)$ (coarsening the residual)
4. solve $(P^T AP)y = r$ (coarse grid problem)
5. $x \leftarrow x + \sigma Py$ (coarse correction)
6. $x \leftarrow x - S^T(Ax - f)$ (post-smoothing)
```

The coarse grid problem $(P^T AP)y = r$ in step 4 is solved directly by a direct skyline solver if it is small enough, or iteratively by μ iterations of Algorithm 4.1 with zero initial approximation, using another yet smaller coarse space and zero initial approximation. The cases $\mu = 1, 2$ are known as the V cycle and W cycle, respectively. The mapping $M : f \mapsto x$, defined by such multilevel algorithm with $x_0 = 0$, is used as the preconditioner. It is well known and easy to see that M is symmetric and, for a suitable choice of σ, positive definite.

4.2. Prolongation. The prolongation P is constructed so that $\operatorname{Range} Z \subset \operatorname{Range} P$ except near boundary constraints, and the norm of $P^T AP$ is small while the norm of P is about one. First a *tentative prolongation* $\hat{P}$ is constructed using the matrix of rigid body modes Z as follows. The set of all degrees of freedom is divided into mutually disjoint *aggregates*,

$$\{1, \ldots, n\} = \mathcal{A}_1 \cup \ldots \cup \mathcal{A}_m.$$

For each aggregate $\mathcal{A}_i$, Z_i is the matrix obtained from Z by setting all rows with indices outside of $\mathcal{A}_i$ to zero, and orthonormalizing the resulting matrix. The tentative prolongation is then

$$\hat{P} = [Z_1, \ldots, Z_m],$$

and the prolongation P is obtained by smoothing,

$$P = (I - \omega DA)\hat{P},$$

where $\omega \leq 4/(3\lambda_{\max}(DA))$, D is a block diagonal matrix where each block D_i corresponds to a node and D_i is equal to the inverse of the corresponding diagonal block in A, or $D_i = 0$. The choice of $D_i = 0$, which avoids smoothing the prolongation for selected nodes, was presented and justified in the work of Brezina, Heberton, and Vaněk reported in [4].

4.3. Smoothing. The smoother S is Gauss-Seidel with overlapping blocks, that is, a multiplicative Schwarz method. The method is defined by an ordered covering of the set of degrees of freedom by *smoothing blocks*,

$$\{1, \ldots, n\} = \mathcal{S}_1 \cup \ldots \cup \mathcal{S}_k.$$

In the following algorithm, an array with set argument means an array section.

Algorithm 4.2. *Input f, x_0. Output: x.*

```
1. $x \leftarrow x_0$
2. for all $j = 1, \ldots, k$, $x(\mathcal{S}_j) \leftarrow (A(\mathcal{S}_j, \mathcal{S}_j)^{-1}[(f - Ax)(\mathcal{S}_j)]$
```

Algorithm 4.2 defines the smoother S in step 2 of Algorithm 4.1 by the equation

$$x = x_0 - S(Ax_0 - f).$$

The smoother S^T in Step 6 of Algorithm 4.1 is obtained by passing though the smoothing blocks in Algorithm 4.2 in the reverse order.

If the smoothing blocks are mutually disjoint, Algorithm 4.2 becomes the block Gauss-Seidel method. Parallelism is achieved by a multi-color generalization of the red-black Gauss-Seidel.

The smoothing blocks may be the same as the aggregates, they may be aggregates enlarged by one layer of nodes, or they may be constructed independently of $\mathcal{A}_j$ as chains of nodes in numerically dominant directions, or by a combination of these approaches.

4.4. Heuristics. The choice of the various parameters of the method and the aggregates and smoothing blocks is done by heuristic algorithms guided by numerical sensors that use the nodal coordinates as well as the entries of A. The algorithms range from simple threshold decisions to graph theoretical algorithms and neural networks. Specific heuristics used in the code include the following:

1. The strength of connection between two nodes is calculated from geometry using the distance between the nodes and their distance from other neighbors. Alternatively, numerical strength of connection is computed from global matrix data in a standard manner as the spectral norm of off-diagonal blocks after symmetric block scaling that changes the diagonal blocks to identity matrices.
2. The aggregates are grown preferably in the direction of strong connections.
3. Smoothing blocks are chosen to be same as aggregates and adaptively enlarged by a layer of nodes if the condition of the submatrix corresponding to the aggregate is high and if the increase in block size is not too large.
4. Another smoothing step may be added, with smoothing blocks determined as connected components of the graph of strong connections. The threshold for the connection strength is decreased until a limit on the complexity of the smoother is reached.
5. If a node has very strongly connected neighbors, $D_i = 0$ is chosen in the prolongator smoother.
6. The choice of algorithms and their parameters is governed by a neural network. The inputs to the neural network are global characteristics of the problem, such as problem size, number of degrees of freedom per node, average number of neigbors of a node, maximal anisotropy of the geometry, etc. The neural network was trained on a large set of industrial problems.

Generally, the heuristics strive to control the number of iterations at the cost of more memory use and more cpu time per iteration. For hard problems, the smoothing blocks dominate the memory usage.

5. Implementation considerations. The integration of new solver technology into existing codes presents many challenges. This section describes some of the implementation considerations related to adding new solver technology to the ANSYS program. The details described are not exhaustive but they do illustrate the kinds of changes made within the ANSYS program in order to add new solver technology to the program.

5.1. Interface and data structures. The initial consideration for adding any solver to an existing program is the solver interface and data structure. ANSYS, like many existing commercial finite element codes, was originally designed around an out-of-core frontal solver. The assembly of the linear system was combined with the factorization of the same matrix in such a way to avoid ever assembling the entire matrix in memory. In contrast to this approach, most solvers accept as input the assembled linear system, usually in a sparse matrix format. The implementation challenge is not switching to sparse assembly, a well known process, but uncoupling an integrated element computation, assembly and solution process that has been modified over many years of development to support many special analysis options and paths into a more modular process which supports multiple solver formats. The initial method for new solver interfacing in ANSYS was to leave the frontal assembly process functioning and add a sparse file interface for iterative methods. The PCG solver added in version 5.3 included a finite element interface that accepted element matrices and boundary conditions, including multi-point constraint equations. The elements were then assembled to form a node-blocked sparse matrix data structure used by the PCG solver. This approach proved to be much more efficient, particularly as problem size grows. A third approach is to develop a true sparse matrix assembly path which directly assembles sparse matrices into a sparse column format. This approach is also used in ANSYS and is currently under development.

In addition to issues related to matrix assembly, the actual data structures used by the various solvers are important. Increasingly, standard sparse matrix structures with sparse column or row storage are not sufficient. Sophisticated iterative solver preconditioners often require information about the association of matrix equations with nodes and/or elements in the finite element model, and the vectors of rigid body modes. Node blocking of the sparse matrix data structures is also used to improve cache performance and reduce integer storage of row and column locations for sparse matrices and is also used to reduce the cost of equation reordering. A straightforward association of user defined nodes in the finite element model with equations in the assembled linear systems is often required but is most often left as an implementation detail.

5.2. Robust and comprehensive solution. Some of the implementation considerations for solvers come from the use of a large library of varied elements in many different analysis types. Special purpose elements such as contact elements and rigid links may be extensively used in large models and are essential for modeling surfaces which come into contact under increasing loads or are joined to adjacent assemblies. Special elements that model uniform pressure within a volume may use Lagrange equation formulations which may require special treatment both in direct solvers and iterative solvers. While the details of heuristics to handle these special elements are proprietary, the changes required to new solvers generally require modifications in elimination ordering and/or attention to numerical properties of the special elements.

5.3. Boundary conditions and constraint equations. The definition of boundary conditions for finite element models is essential but can provide another source of implementation considerations. Standard Neuman and Dirichlet boundary conditions are easily handled, usually external to the solver at assembly time. Equations corresponding to fixed degrees of freedom can be eliminated and not assembled into the linear systems or they can be easily modified by zeroing all but the diagonal coefficients in each equation corresponding to a fixed degree of freedom.

Constraint equations that are used to couple groups of nodes or enforce some linear combination of displacement conditions between certain degrees of freedom are more difficult. Many large models are separately meshed assemblies and they rely on constraint equations to couple the separate assemblies into one global model. Constraint equations are also used extensively to enforce symmetry boundary conditions. Constraint equations can be imposed by using a Lagrange equation formulation or by direct elimination of constrained nodes prior to factorization. The Lagrange formulation causes stability problems for the direct and iterative methods and the direct elimination approach requires modification of the assembled matrix prior to solution.

Most solver libraries do not provide an interface for handling constraint equations so the direct elimination approach must be used or the solver library must be modified. The PCG solver in ANSYS does provide an interface for constraint equation. The preconditioner formulation includes the information of constraint equations so it is very effective in preconditioning with constraint equations. A sparse implementation of direct elimination of constraint equations has been developed for the AMG solver and will also be used with the sparse direct solver interface in ANSYS 6.0.

5.4. Singular systems, ill-conditioning and rigid body modes. A final source of implementation considerations for adding new solvers into existing finite element codes is the handling of singular systems. Most finite element codes that use a direct factorization method have developed heuristics for detecting singular systems. In some cases, modifications are made to the linear systems during factorization to remove the singularities. Most iterative solvers do not detect these singular systems and have no heuristic to modify the linear systems appropriately. Direct solvers may provide a means to test the diagonal pivots but code-specific heuristics are often added to the basic solver.

In simple static analyses, singular systems often occur if the boundary conditions imposed by a user fail to constrain all rigid body modes. The singularity in such a case will most often be detected during matrix factorization as a very small or identically zero diagonal coefficient. Continuing factorization with very small pivots will most likely produce garbage answers, at least in the unconstrained directions, or they may cause floating point exceptions. In these cases a warning message is often printed and users can change boundary conditions to remove the problem. Alternatively, when small pivots are detected the linear system can be modified to effectively remove the column causing the small pivot by constraining that degree of freedom. The details of this process vary from code to code and must be duplicated for any new solver added to the analysis program.

In iterative solvers, singulatities arising from the same rigid body modes may or may not cause problems for convergence, depending on whether the iterative solution has components in the rigid body modes. Iterative methods cannot be expected to duplicate the results of direct methods which heuristically modify small diagonal pivots. Some iterative solver packages detect singularities in the formation of the preconditioner. In most cases these solvers do not make modifications to continue with solution.

Nonlinear analyses may also result in a singular system when the solution is close to a bifurcation point. In these cases detection of the condition is important but heuristically modifying the diagonal pivots is not desired. The use of iterative methods in nonlinear analyses may not work when bisection due to matrix singularity is encountered. Within ANSYS, certain types of singularities are allowed, depending on the physics on the problem. In most cases monitoring of diagonal pivots during factorization is required with errors and warnings printed out depending on the analysis type.

Very few linear solver libraries handle all of the above mentioned implementation considerations. At the same time, existing codes which are designed around a single solver architecture do not provide clean interfaces and data structures for new solvers. Within ANSYS a steady migration away from the frontal solver assembly/solution loop is in progress. Within the BCS sparse solver changes have been made to duplicate heuristics that are used with the frontal solver to detect and report various error conditions that may occur during factorization. The sparse file interface coupled with frontal assembly that allowed initial implementations of iterative solvers and the BCS sparse solver library was essential to provide an efficient way to add new solver technology with minimal code changes. The migration of the solver interfaces from frontal assembly to a true sparse assembly procedure has greatly improved performance and reduced memory bottlenecks. Ultimately, a more component based style of programming should allow greater flexibility and ease of use for new solvers but the component design will have to include provisions that match the demands of the large scale applications codes.

6. Results. In this section results are presented from runs using ANSYS 5.7. Timings from industrial problems and ANSYS test problems are given for several of the solvers described above. Parallel processing results demonstrate current scalability of solvers in ANSYS. All of the parallel processing runs presented in this paper are from non-dedicated multi-processor systems. The timings demonstrate that routine analyses of finite element models with several hundred thousand up to a few million equations are now possible in ANSYS using both direct and iterative solvers.

6.1. Single processor solver comparisons. Table 6.1 contains a comparison of the PCG and AMG iterative solvers for a set of 37 ANSYS test problems. The first 16 problems in Table 6.1 are faster using AMG than PCG. Several problems show dramatic improvements for the AMG solver over PCG. These are problems using elements with high aspect ratios, such as aspec50_big, solidCE_20, solidCE_30, etc. or problems which combine shell and solid elements. For example, shbeamsolid_20, shbeamsolid_40, and shellsolid_32, etc. For these problems the AMG solver delivers substantial improvements for iterative solvers in ANSYS. On the other hand, shell dominated models are still faster using the PCG solver. For example, cyli_100 and cyli_400. The PCG solver also uses less memory than AMG. Generally, AMG will use about 30 percent more memory than the PCG solver in practice. For more difficult problems the AMG preconditioner automatically changes, requiring more memory but generally resulting in dramatically improved convergence compared to the PCG solver.

Table 6.2 shows timings for several runs using a parameterized wing model. This model is a solid model of a structure resembling an airplane wing shape. Solid brick elements are used to extrude a 3-D mesh from a 2-D cross sectional auto-meshed region. The model therefore has some characteristics of a solid model where dense meshes of solid elements dominate. This type of mesh produces uniformly shaped elements which can result in very fast convergence for iterative methods. The times in Table 6.2 compare the AMG and PCG iterative solver with the sparse direct solver, denoted SPAR. Both problem size and an aspect ratio parameter are varied to show the effects of problem size and matrix ill-conditioning on the relative performance of the AMG, PCG and SPAR solvers. The problem sizes vary from 134K DOFs to 489K DOFs. The aspect ratio parameters vary the element sizes from nearly cube shaped elements for $Aspect = 1$ to uniformly increased elongated elements for higher values of *Aspect*. For this problem both iterative solvers are faster than the direct solver for the well-conditioned problems. The times clearly increase at an expected higher rate for the sparse solver as the problems grow in size as compared to the iterative solvers. However, as the problems become more ill-conditioned the PCG solver time increases to eventually exceed the sparse solver time. Even for the largest problem the PCG solver is slower than the direct solver due to poor convergence.

6.2. Parallel processing solver comparisons. Table 6.3 shows parallel processing performance for a 245K DOF model of the wing model with varying aspect ratios as in Table 6.2. All three solvers run in parallel on an HP 4-processor L-Class system but the AMG solver scales the best for these runs. The SPAR times shown use a new parallel implementation of the sparse solver factorization routines available in ANSYS 5.7.1 on HP and IBM systems.

Table 6.4 shows timings for the same wing model used in Tables 6.2 and 6.3 comparing memory usage, convergence rates and parallel processing performance for the SPAR, AMG, PCG and the domain decomposition based solver, DDS. A 488K DOF model size was used in these runs. All of the runs were made on a large SGI O2000 server in a non-dedicated environment. The DSS solver was run using MPI distributed memory processing. The times shown are for the equation solution time only. For this job the AMG solver is the fastest solver. The DDS solver achieves the best scaling on this job and shows even better parallel performance for larger problems. The shared memory parallel implementations used for SPAR, AMG, and PCG are effective up to around 4 processors but do not scale higher due to both I/O bottlenecks as well as memory bandwidth limitations of shared memory machines. While this example gives a simple comparison of three shared memory parallel solvers with the domain decomposition based solver, it does not provide conclusive data on the relative merits of the two approaches.

The multi-level AMG preconditioner is much more effective than the PCG method for the ill-conditioning caused by elongated elements. The PCG iterations increase more than fourfold when the aspect ratio is increased while the AMG preconditioner automatically refines, using more memory but requiring only four additional iterations. The cost of the iterations for AMG clearly increases for the more ill-conditioned problem as well.

Memory requirements for each solver show that the memory requirements for the PCG and SPAR solver are nearly identical while the AMG solver requires about twice as much memory for this example and increases additionally when the more ill-conditioned problem uses a more expensive preconditioner. The DDS solver requires much more memory but is generally used in a distributed computer environment where the memory requirements are spread out across the available computer resources. Reducing the total memory requirement for the DDS solver is currently an area of research within ANSYS. Domain decomposition solvers are designed to exploit the availability and power of large numbers of processors. The increased memory requirement for domain based solvers is characterisic of solvers which are designed for scalable performance on large numbers of processors.

While the wing model is an example problem contrived to easily scale both mesh density, problem size and aspect ratios, Table 6.5 is a similar sized problem from an industrial customer. The 590k DOF model comes from an auto-meshed complex 3-D geometry that primarily combines tetrahedron-shaped elements with several thousand contact elements. Many industrial problems are well-conditioned and perform well for the iterative methods used in ANSYS, but this problem is a difficult problem for iterative solvers. At the same time, the more dense 3-D geometry proves very efficient for the SPAR direct solver. For this problem, the sparse direct solver is much faster than either PCG or AMG. The AMG preconditioner is again an improvement over PCG for this problem and shows much better parallel scaling as well. The DDS solver was unable to solve this industrial problem because of the use of contact elements in the model which are not supported for DDS in ANSYS 5.7.

Finally, Table 6.6 demonstrates the scaling of solver time that is possible with the domain decomposition based DDS solver. The 3.5 million DOF problem was run on a 32 processor SGI Origin system using 12 Gbytes of memory. The DDS solver offers a scalable solution for problems where direct methods are no longer feasible due to the size requirements for disk storage of the factored matrix or the prohibitive time for factorization.

These results demonstrate that with current solver technology in ANSYS many problems in the range of 500k DOFs to over 1 million DOFs can be solved on existing hardware systems in times from a few minutes to a few hours. The improvements in iterative solver convergence and parallel processing performance have advanced this capability. Sparse direct methods are still competitive with iterative methods, even at these large problem sizes, due to an implementation which uses I/O efficiently to reduce memory requirements combined with very effective equation reordering technology that reduces factorization operations and storage requirements. The domain decomposition solver offers a solution for the very largest problems that will scale to use larger numbers of available systems as an alternative to an expensive multi-processor system with very large memory. The use of the domain decomposition solver will continue to grow as a truly scalable solver option is developed in the ANSYS program that will allow parallel processing of the total solution process in ANSYS.

7. Summary and future solver directions. This paper has described the evolution of solvers in ANSYS from a single direct frontal solver to a collection of direct and iterative solvers. The addition of new solver technology has extended the capabilities of ANSYS by increasing the size of linear systems that can be solved within acceptable time limits using existing computer resources. The implementation of new solver technology has been described and results from some example problems have been presented.

Future solver improvements in ANSYS will seek to continue to extend the capabilities for larger simulations. One goal of solver improvements is to incorporate iterative solver technology into a robust eigensolver capable of solving models with several million degrees of freedom and above. The current block Lanczos solver is robust and capable of solving large models up to a few million degrees of freedom. Finite element models with 10 million degrees of freedom which use direct factorization solvers would most likely require hundreds of Gbytes of file storage and require prohibitively large memory and execution time, even at today's achievable Gflop performance. In addition to extending the maximum job size limits with new solver technology, ANSYS will continue to reduce minimum memory requirements for existing solvers through refinement of the solver interfaces. This important goal extends the capability of routine analyses on existing workstations, particularly workstations that are limited to 2 Gbytes or less of total address space.

REFERENCES

[1] C. C. Ashcraft, R. Grimes, J. G. Lewis, B. Peyton, and H. Simon, *Progress in sparse matrix methods for large linear systems on vector supercomputers*, International Journal of Supercomputer Applications, 1 (1987), pp. 10–30.

[2] C. Farhat, J. Mandel, and F.-X. Roux, *Optimal convergence properties of the FETI domain decomposition method*, Comput. Methods Appl. Mech. Engrg., 115 (1994), pp. 367–388.

[3] C. Farhat and F.-X. Roux, *An unconventional domain decomposition method for an efficient parallel solution of large-scale finite element systems*, SIAM J. Sci. Stat. Comput., 13 (1992), pp. 379–396.

[4] C. Heberton, *Eulerian-Lagrangian Localized Adjoint Method and Smoothed Aggregations Algebraic Multigrid*, PhD thesis, University of Colorado at Denver, May 2000. http://www-math.cudenver.edu/graduate/thesis/heberton.ps.gz.

[5] B. M. Irons, *A frontal solution program for finite element analysis*, International Journal of Numerical Methods in Engineering, 56 (1970), pp. 5–23.

[6] P. C. Kohnke, ed., *ANSYS Engineering Analysis System Theoretical Manual*, Swanson Analysis Systems, Inc., Houston, PA, 1989.

[7] J. Mandel, *Adaptive iterative solvers in finite elements*, in Solving Large Scale Problems in Mechanics. The Development and Application of Computational Solution Methods, M. Papadrakakis, ed., J. Wiley & Sons, London, 1993, pp. 65–88.

[8] ———, *Intelligent block iterative methods*, in FEM Today and the Future, J. Robinson, ed., Robinson and Associates, Okehampton, Devon EX20 4NT, England, 1993, pp. 471–477. Proceedings of the 7th World Congress on Finite Elements, Monte Carlo, November 1993.

[9] ———, *Iterative methods for p-version finite elements: Preconditioning thin solids*, J. Comput. Meth. Appl. Mech. Engrg., 133 (1996), pp. 247–257.

[10] J. Mandel, M. Brezina, and P. Vaněk, *Energy optimization of multigrid bases*, Computing, 62 (1999), pp. 205–228.

[11] E. L. Poole, M. Heroux, P. Vaidya, and A. Joshi, *Performance of iterative methods in ANSYS on CRAY parallel/vector supercomputers*, Computer Systems in Engineering, 3 (1995), pp. 251–259.

[12] P. Vaněk, *Acceleration of convergence of a two level algorithm by smoothing transfer operators*, Appl. Math., 37 (1992), pp. 265–274.

[13] ———, *Fast multigrid solver*, Appl. Math., 40 (1995), pp. 1–20.

[14] P. Vaněk, M. Brezina, and J. Mandel, *Convergence analysis of algebraic multigrid based on smoothed aggregation*, Numer. Math., 88 (2001), pp. 559–579.

[15] P. Vaněk, J. Mandel, and M. Brezina, *Algebraic multigrid based on smoothed aggregation for second and fourth order problems*, Computing, 56 (1996), pp. 179–196.

ANSYS Test problems Using shell, solid and beam elements with different aspect ratios and geometries		
Problem	**AMG (sec)**	**PCG (sec)**
1 BoxDS	555.5	1145.1
2 Ibeam	329.1	478.4
3 aspec50	124.6	1165.8
4 aspec50_big	406.6	2738.2
5 carrier1	770.8	819.5
6 casing3	217.8	286.8
7 runbig	58.5	85.6
8 runbig2	60.1	88.1
9 shbeamsolid_20	68.7	590.9
10 shbeamsolid_40	730.9	3875.7
11 shellsolid_32	484.0	1938.6
12 shellsolid_40	860.8	4080.5
13 solidCE_20	34.4	1541.5
14 solidCE_30	155.1	4541.3
15 solidCP_30	131.6	214.3
16 solidCP_40	307.1	483.8
17 beamj	217.5	153.3
18 carrier2	1166.3	1021.9
19 cyli	260.1	46.7
20 cyli_100	128.7	10.5
21 cyli_400	1911.3	290.7
22 heat1	66.1	27.6
23 nb30	117.6	69.8
24 plane02_500	207.8	136.9
25 plane42_1000	121.7	107.3
26 rcoarse2	115.5	95.2
27 s92_100	296.4	210.6
28 s92_130	794.5	543.1
29 s92_80	148.4	102.9
30 sh63_200	68.7	45.3
31 sh63_400	371.9	206.6
32 solid70b_100	80.2	34.5
33 solid87	123.1	58.8
34 solid92	366.4	237.3
35 voll2	2788.4	2592.5
36 ycoup3	405.1	346.7
37 ycoup5	523.8	518.9

Table 6.1
Comparison of AMG and PCG Iterative solvers for ANSYS test problems

Linear Static Analysis Using Wing Model Mesh refined to show effects of increasing problem size and element aspect ratio				
Solver	**Aspect Ratio**	Degrees of Freedom		
		134K	245K	489K
SPAR		303	1147	3633
AMG	1	98	204	817
AMG	10	133	325	1191
AMG	25	171	414	1484
PCG	1	126	289	952
PCG	10	365	649	1787
PCG	25	1532	1895	4691

- HP L-Class 4 550 Mhz Processors, 4 Gbytes memory
- Times are for linear solution time in seconds
- Increasing aspect uniformly elongates elements

Table 6.2
Solver Comparison for Increasing Size Problems

Linear Static Analysis using Wing Model Mesh refined to show effects of increasing problem size and element aspect ratio				
Solver	**Aspect Ratio**	245K DOFs Number of Processors		
		NP=1	NP=2	NP=3
SPAR		1147	909	843
AMG	1	204	152	142
AMG	10	325	225	197
AMG	25	414	288	238
PCG	1	289	244	231
PCG	10	649	564	544
PCG	25	1895	1769	1615

- HP L-Class 4 550 Mhz Processors, 4 Gbytes memory
- PCG CA&SI Iterative solver (ANSYS PowerSolver)
- AMG Algebraic Multigrid solver
- SPAR Boeing Sparse Direct solver 4.0

Table 6.3
Parallel Processing Solver Comparison for 245k DOF problem

Linear Static Analysis Using Wing Model 488K Degrees of Freedom					
Solver	**Memory**	**Iterations**	**Solver Elapsed Time)**		
	Mbytes	*tol* $< 10^{-6}$)	NP=1	NP=4	NP=8
Aspect Ratio Parameter = 1					
PCG	246	291	566.9	422.7	378.5
AMG	518	13	203.0	98.3	82.0
DDS	2113	53	547.2	176.0	125.4
SPAR	256	N/A	4392.7	2044.5	1728.2
Aspect Ratio Parameter = 10					
PCG	246	1293	1891.1	1524.9	1329.4
AMG	589	17	674.0	312.4	204.3
DDS	1956	259	1739.3	440.2	332.1
SPAR	256	N/A	4392.7	2044.5	1728.2

- Runs made on an SGI O2000 16 300 Mhz Processors, 16 Gbytes Memory
- PCG CA&SI Iterative solver (ANSYS PowerSolver)
- AMG Algebraic Multi-Grid solver
- DDS Domain Decomposition solver
- SPAR Boeing Sparse Direct solver 4.0

Table 6.4
Parallel Processing Performance for Iterative Solvers

Large Industrial Example, 590k Degrees of Freedom Static Analysis with Nonlinear Contact						
Solver	**Memory**	**Iterations**	**Solver Elapsed Time (sec)**			
	Mbytes	*tol* $< 10^{-6}$)	NP=1	NP=2	NP=4	NP=8
PCG	300	5301	8575	7675	6595	6891
AMG	722	200	5265	3831	2638	1884
SPAR	290	N/A	705	600	481	457

- Runs made on an SGI O2000 16 300 Mhz Processors, 16 Gbytes Memory
- PCG CA&SI Iterative solver (ANSYS PowerSolver)
- AMG Algebraic Multi-Grid solver
- SPAR Boeing Sparse Direct solver 4.0

Table 6.5
Iterative and Direct Solver comparison of linear system solution time

3.5 Million Dof Example 10-Node tetrahedron element mesh 2020 Subdomains, 12 Gbytes Memory Used		
Processors	**Elapsed Time**	**Speedup**
1	18806	1.0
2	9403	2.0
4	5074	3.6
6	3427	5.8
8	2660	7.0
10	2420	7.8
14	1790	10.6
18	1447	13.0
22	1374	13.6
26	1026	18.4
30	901	20.9

- Runs made on a 32-processor SGI O2000

Table 6.6
Scalability of Domain Decomposition Solver for linear system solution time