White Paper –
Obtaining Optimal Performance in ANSYS 11.0

Published: December 13, 2007

ANSYS, Inc.
Southpointe
275 Technology Drive
Canonsburg, PA 15317
ansysinfo@ansys.com
http://www.ansys.com
(T) 724-746-3304
(F) 724-514-9494

Copyright and Trademark Information

Disclaimer Notice

U.S. Government Rights

Third-Party Software

Published in the U.S.A.

Table of Contents

List of Tables

Chapter 1: Understanding ANSYS Computing Demands

This performance guide provides a comprehensive resource for ANSYS users who wish to understand factors that impact the performance of ANSYS on current hardware systems. The guide provides information on:

- ANSYS computing demands
- Hardware considerations
- Memory usage
- Parallel processing
- I/O considerations

The guide also includes general information on how to measure performance in ANSYS and an example-driven section showing how to optimize performance for several ANSYS analysis types and several ANSYS equation solvers. The guide provides summary information along with detailed explanations for users who wish to push the limits of performance on their hardware systems. Windows and UNIX/Linux operating system issues are covered throughout the guide.

The ANSYS program requires a computing resource demand that spans every major component of hardware capability. Equation solvers that drive the simulation capability of Workbench and ANSYS analyses are computationally intensive, require large amounts of physical memory, and produce very large files which demand I/O capacity and speed. To best understand the process of improving ANSYS performance, we begin by examining:

Memory requirements within ANSYS
Parallel processing
I/O requirements within ANSYS

1.1. Memory

Memory requirements within ANSYS are driven primarily by the requirement of solving large systems of equations. Details of solver memory requirements are discussed in more detail in a later section. Additional memory demands can come from meshing large components and from other pre- and post processing steps which require large data sets to be memory resident (also discussed in a later section).

Another often-overlooked component of memory usage in ANSYS comes from a hidden benefit of large physical memory systems. This hidden benefit is the ability of modern operating systems, both Windows 64-bit and UNIX/Linux, to use available physical memory to cache file I/O. Maximum system performance for ANSYS simulations occurs when there is sufficient physical memory to comfortably run the ANSYS solvers while also caching the large files used by the simulations. Affordable large memory systems are now available from the desktop to the high-end server systems, making high performance computing solutions available to virtually all ANSYS users.

1.1.1. Specifying Memory Allocation in ANSYS

ANSYS memory is divided into two blocks: the *database* space that holds in memory the current model data and the *scratch* space that is used for temporary calculation space (used, for example, for forming graphics images and by the solvers). The database space is specified by the -db command line option. The initial allocation of *total* workspace is specified by the -m command line option. The scratch space is the total workspace minus the

database space. Understanding how scratch space is used (as we will see in later chapters) is an important component of getting optimal performance out of some of the solver options.

Scratch space grows dynamically in ANSYS, provided the memory is available when the ANSYS memory manager tries to allocate additional memory from the operating system. If database space is too small, ANSYS automatically uses a disk file to spill the database to disk. See Memory Management and Configuration in the *Basic Analysis Guide* for additional details on ANSYS memory.

1.1.2. Memory Limits on 32-bit Systems

While it is generally no longer necessary for ANSYS users to use the -m command line option to set initial memory, it can be an important option, particularly when you attempt large simulations on a computer system with limited memory. Memory is most limited on 32-bit systems, especially Windows 32.

It is important that you learn the memory limits of your 32-bit systems. The ANSYS command line argument -m is used to request an initial contiguous block of memory for ANSYS. Use the following procedure to determine the largest -m setting you can use on your machine. The maximum number you come up with will be the upper bound on the largest contiguous block of memory you can get on your system.

1. Install ANSYS.
2. Open a command window and type:

```
ansys110 -m 1200 -db 64
```

3. If that command successfully launches ANSYS, close ANSYS and repeat the above command, increasing the -m value by 50 each time, until ANSYS issues an insufficient memory error message and fails to start. Be sure to specify the same -db value each time.

Ideally, you will be able to successfully launch ANSYS with a -m of 1700 or more, although 1400 is more typical. A -m of 1200 or less indicates that you may have some system DLLs loaded in the middle of your memory address space. The fragmentation of a user's address space is outside the control of the ANSYS program. User's who experience memory limitations on a Windows 32 system should seriously consider upgrading to Windows 64. However, many users will be able to run ANSYS simulations with excellent performance, provided the models are not so large that they use all available memory.

1.2. Parallel Processing

Multiple processors, and thus the ability to use parallel processing, are now widely available on all computer systems, from laptops to high-end servers. The benefits of parallel processing are compelling, but are also among the most misunderstood. This guide will explain the two types of parallel processing within ANSYS and will attempt to realistically set customer expectations for parallel processing performance.

No matter what form of parallel processing is used, the maximum benefit attained will always be limited by the amount of work in the code that cannot be parallelized. If just 10 percent of the runtime is spent in nonparallel code, the maximum theoretical speedup is only 10, assuming the time spent in parallel code is reduced to zero. However, parallel processing can be an effective way to reduce wall clock elapsed time, thereby providing significant value when performing simulations.

In many parallel regions in ANSYS, the speedups obtained from parallel processing are nearly linear as the number of processors are increased, making very effective use of multiple processors. The total benefit measured by elapsed time is problem dependent and is influenced by many different factors.

1.2.1. Shared Memory Parallel vs. Distributed Memory Parallel

Shared memory parallel (SMP) is distinguished from *distributed memory parallel* (DMP) by a different memory model. SMP systems share a single global memory image that may be distributed physically across multiple nodes or processors, but is globally addressable. These systems are the easiest to program for parallel processing, but they have also historically been the most expensive systems to build. The new generation multicore systems are examples of SMP systems.

Distributed memory processing assumes that physical memory for each process is separate from all other processes. Parallel processing on such a system requires some form of message passing software to exchange data between the processors. The prevalent software used today is called MPI (Message Passing Interface). MPI software uses a standard set of routines to send and receive messages and synchronize processes. The DMP programming model can be used on any computer system that supports a message passing programming model, including cluster systems and single-box shared memory systems. A major attraction of the DMP model is that very large parallel systems can be built using commodity priced components or assembled using a cluster of idle desktop systems. The disadvantage of the DMP programming model is that programming in this model requires significantly more code changes for large existing applications such as ANSYS.

Although SMP systems share the same memory, there is a limitation to how many processors can access shared memory without performance degradation. Some systems use a memory architecture in which the shared memory is accessed at different rates by different processors or cores. These NUMA (Non-Uniform Memory Architecture) systems, therefore, have limited scalability in SMP mode. Other SMP systems use a bus architecture with nearly flat memory speed for all processors, but the total memory bandwidth is limited by the capacity of the bus. Running in DMP mode can be beneficial for attaining maximum scalability on both types of systems; it helps improve the use of the faster memory references on NUMA machines and reduces message-passing costs on all SMP systems.

1.2.2. Parallel Processing in ANSYS

ANSYS simulations are very computationally intensive. Most of the computations are performed within the solution phase of the analysis. During the solution, three major steps are performed:

1. Forming the element matrices and assembling them into a global system of equations
2. Solving the global system of equations
3. Using the global solution to derive the requested set of element and nodal results

Each of these three major steps involves many computations and, therefore, has many opportunities for exploiting multiple processors through use of parallel processing.

All three steps of the ANSYS solution phase can take advantage of SMP processing, including most of the equation solvers. However, the speedups obtained in ANSYS are limited by requirements for accessing globally shared data in memory, I/O operations, and memory bandwidth demands in computationally intensive solver operations. SMP programming in ANSYS uses the OpenMP programming standard.

Distributed ANSYS (DANSYS) is the DMP version of ANSYS. While DANSYS is a separate executable from ANSYS, it shares the same source code as ANSYS with the addition of the MPI software for communication. For this reason, the capability of DANSYS to solve ANSYS jobs is identical within the areas of ANSYS where DANSYS implementation is complete. DANSYS parallelizes the entire ANSYS solution phase, including the three steps listed above. However, the maximum speedup obtained by DANSYS is limited by how well the computations are balanced among the processors, the speed of messages being passed between processors, and the amount of work that cannot be done in a distributed manner.

Users may now choose SMP or DMP processing using 2 processors with the standard ANSYS license. To achieve additional benefit from parallel processing, users must acquire additional ANSYS Mechanical HPC licenses. Nearly

all ANSYS analyses will see reduced time to solution using parallel processing. While speedups vary due to many factors, users should expect to see the best time to solution results using DANSYS on properly configured systems having 8-12 processors.

1.3. I/O in ANSYS

The final major computing demand in ANSYS is file I/O. The use of disk storage extends the capability of ANSYS to solve large model simulations and also provides for permanent storage of results.

One of the most acute file I/O bottlenecks in ANSYS occurs in the direct equation solvers (sparse and distributed sparse) and block Lanczos eigensolver, where very large files are read forward and backward multiple times. For block Lanczos, average-sized runs can easily perform a total data transfer of 1 terabyte from disk. At a typical disk I/O rate of 30-50 MB/sec on most desktop PCs, this I/O demand can add hours of elapsed time to a simulation. Another expensive I/O demand in ANSYS is saving results files for multiple step analyses. Results files that are tens to hundreds of GB in size are common if all results are saved for all time steps in a transient analysis of a large model.

This guide will discuss ways to minimize the I/O time in ANSYS. Important breakthroughs in desktop I/O performance that have been added to ANSYS 11.0 will be described later in this document.

Chapter 2: Hardware Considerations

This chapter discusses important issues related to hardware when considering a high performance computing (HPC) system. The following topics are covered:

2.1. What Is an HPC System?
2.2. CPU, Memory, and I/O Balance
2.3. CPU Choices
2.4. Memory
2.5. I/O Hardware

2.1. What Is an HPC System?

The definition of high performance computing (HPC) systems in a continually changing environment is difficult to achieve if measured by any fixed performance metric. However, if defined in terms of improving simulation capability, high performance computing is an ongoing effort to remove computing limitations from engineers who use computer simulation. This goal is never completely achieved because computers will never have infinite speed and unlimited capacity. Therefore, HPC systems attempt to deliver maximum performance by eliminating system bottlenecks that restrict model size or increase execution time.

This document describes current hardware capabilities and recommended system configurations that maximize the performance of every computer system running ANSYS simulations. This is achieved by avoiding unbalanced configurations and sizing systems to meet the demands of the user's expected model size.

2.2. CPU, Memory, and I/O Balance

The key factor in assembling an HPC system is the correct balance of CPU, memory, and I/O. Processors are now multicore, operate at several GHz (Giga (10^9) Hertz) frequencies, and are capable of sustaining compute rates of 1 to 5 Gflops (Giga (10^9) floating point operations per second) per core in ANSYS equation solvers. The CPU speed is an obvious factor of performance. If the CPU performs computations at a very slow rate, having large amounts of fast memory and I/O capacity will not significantly help performance. The other two factors, memory and I/O, are not always so obvious. With the latest processors now being multicore and operating at several GHz frequencies, the speed and capacity of memory and I/O become much more important; they are both needed to feed these processors as quickly as possible.

2.3. CPU Choices

The least critical choice in HPC systems today is the processor. However, there are still important differences that effect system performance. Most computer users talk about their CPU speed in terms of GHz. However, the clock frequency of a CPU is only one indicator of the peak speed of the given processor. The fastest processors today run with clock speeds (clock ticks/second) from 1 GHz to nearly 4 GHz. The other factor in CPU speed is how many floating point operations (flops) can be performed per clock cycle. Most processors can achieve a maximum of one add and one multiply result per clock period, or 2 floating point operations per clock. Thus, a 3 GHz processor with 2 flops per clock is theoretically capable of 6 Gflops. Notable exceptions to 2 flops per clock are the Intel Itanium, the new line of Intel core micro-architecture dual- and quad-core Xeon processors, the IBM Power 5, and the newly announced AMD Quad-core Opteron processors. These processors are capable of 4 floating point operations per clock period, *per core*. Thus, a 1.6 GHz Itanium processor has the same theoretical peak performance as a 3.2 GHz Pentium processor. The new core micro-architecture Intel EM64T Xeon processors are 64-bit processors, run at a maximum of 3 GHz, and are capable of 4 flops per clock, making them among the fastest processors available today—at commodity prices! Not all Xeon branded processors are core micro-architecture, and not all AMD Opteron processors produce 4 flops per clock; check the processor type carefully to obtain the faster chips.

None of these processors sustain peak speed in typical applications, but both Intel and AMD processors can achieve sustain rates that are about half of the theoretical peak in compute intensive optimized code. Usually, the highest rates are obtained only in highly optimized math library routines that are written by experts from the hardware vendors. ANSYS uses these high performance math libraries on all systems where they are available. *Table A.1, "Comparison of CPU Characteristics and Performance"* gives a comparison of a few representative processors currently available.

2.3.1. Commodity Vs. Proprietary Processors

The computing world today is dominated by commodity processors that use the x86 instruction set. These include both Intel and AMD processors. All x86 compatible processors can run the Windows operating system. Due to the rise in x86 processor performance in recent years, extension to 64-bit address space, and the development of Windows 64-bit, these commodity processors are now a legitimate component of true HPC desktop and server systems.

In addition to x86 compatible processors, there are still some proprietary processors (for example, the IBM Power series processors). These processors are very high performance with excellent memory bandwidth capability, but at a higher cost to the user. They run best using the proprietary IBM AIX UNIX operating system; they do not run Windows. Sun also makes a line of proprietary processors that run the Sun Solaris UNIX operating system. The Intel Itanium processor is not an x86 instruction set processor and can be found in both name brand HPC systems, such as SGI Altix and HP Integrity servers, as well as systems from smaller companies that make very competitively priced HPC systems.

The proprietary systems mentioned support ANSYS and do offer very stable multi-user systems that have been tuned to run HPC applications. They are most cost effective in multi-user environments or where very large memory systems (above 128 GB) are desired. These systems still offer the highest absolute performance for customers who want to solve the largest models possible or who need to share a large computing resource in a central environment.

2.3.2. Multicore Processors

Parallel processing is now available on virtually every computer system sold, from commodity processors through high-end servers. With the advent of dual- and quad-core processors, the hardware manufacturers have introduced a confusion in nomenclature by referring to multicore processors. Each core in a multicore processor is a legitimate processor, independent from the other cores. A dual processor quad-core machine is really an 8 processor computing resource.

Multicore processors should not be confused with another parallel processing term from Intel: hyperthreading. Hyperthreading is not the same as adding a functioning core. Hyperthreading is an operating system form of parallel processing that uses extra virtual processors to share time on a smaller set of physical processors, or cores. This form of parallel processing is only effective when a system has many lightweight tasks. Giving each task a time slice of the physical processors can improve overall system response. However, for an HPC application the only useful form of parallel processors are the independent functioning processor cores that can each compute simultaneously.

While the multicore architectures are still evolving with each design cycle, they are already performing very well in ANSYS. Multicore processors share cache space and, in some cases, memory bandwidth. However, bus speeds have increased and other improvements to the memory architecture have been made to support the demands of multicore processors. Currently (as of mid 2007), dual-core processors sustain a higher compute rate per core than quad-core processors. For example, a 4 processor dual-core system runs faster than a 2 processor quad-core system if the clock speeds and bus speeds are equal. This is an expected result given that dual-core processors place less demand per processor on memory and cache bandwidth than quad-core processors.

2.4. Memory

Memory size for HPC systems is the single most critical factor when considering ANSYS performance. Fortunately, the density of memory parts continues to increase as prices fall, so it is now affordable to obtain desktop 64-bit workstations that have 8 GB of memory. The bus speed of desktop systems is a critical element of memory configurations; it is a key element to allowing very fast processors to access large memory fast enough to sustain high performance results. The details of memory architectures differ between manufacturers. However, an HPC desktop system should use the highest bus speed configuration available to achieve peak performance since ANSYS solvers will use all of the memory bandwidth available, particularly when using multicore processors.

For Intel processors, the current fastest bus speed is 1333 MHz for core micro-architecture processors. Comparing bus speeds for different processors can be confusing. AMD, Intel EM64T, and Intel Itanium all use different bus architectures, making a direct comparison of bus frequency meaningless. However, for any given processor family the fastest bus speed available is preferred.

HPC systems will also run best with larger memory. Motherboards with more memory slots will allow users to add large memory to support the processor speed and memory bandwidth demands. It is generally better to fill all open memory slots than to buy higher density memory, leaving open slots. Most systems run best if the memory parts are all identical. Currently, 2 GB memory SIMMS are priced the same per GB of memory as lower density parts, while higher density 4 GB memory SIMMS are often doubled in price *per GB*. Memory that is priced linearly in memory size is expected to extend to 4 GB SIMMS in the near future and beyond as the technology advances. Since memory speed is an important part of sustainable performance, premium memory parts are worth the investment to obtain an HPC system.

Finally, cache sizes can vary widely on processors. Larger cache sizes help processors to achieve a higher performance for HPC applications. Itanium processors have cache sizes from 6 to 24 MB, while most dual and quad core Intel processors share cache sizes ranging from 4 MB to 8 MB. Each core typically has 2 MB. AMD processors typically have 512k to 1 MB cache per core.

2.5. I/O Hardware

I/O capacity and speed are important parts of an HPC system. While disk storage capacity has grown dramatically in recent years, the speed at which data is transferred to and from disks has not increased nearly as much as processor speed. Processors compute at Gflops (billions of floating point operations per second) today, while disk transfers are measured in MB/sec (megabytes per seconds), a factor of 1,000 difference! This performance disparity can be hidden by the effective use of large amounts of memory to cache file accesses. However, the size of ANSYS files often grows much larger than the available physical memory so that system file caching is not able to hide the I/O cost.

Many desktop systems today have very large capacity disks that hold several hundred GB or more of storage. However, the transfer rates to these large disks can be very slow and are significant bottlenecks to ANSYS performance. Typical home systems have low cost, high capacity disks that typically rotate at 5000 to 7200 rpm and sustain transfer rates of 10 to 50 MB/sec. SATA and SCSI drives that rotate at 10k to 15k rpm and sustain 50 to 75 MB/sec transfer rates are available for desktop workstations. Many disk manufactures claim higher rates, called burst rates, but ANSYS I/O requires a sustained I/O stream to write or read files that can be hundreds of MB to several GB in length.

Historically, SCSI disk drives are used on higher-end systems while lower costing SATA drives have emerged as favorites on the desktop. SATA drives are becoming very widely used due to low cost and improvements in performance. The key to obtaining outstanding performance, however, is not finding a fast single drive, but rather using disk configurations that use multiple drives configured in a RAID setup that looks like a single disk drive to a user. RAID0 is the fastest configuration of multiple disk drives. Other RAID configurations maintain duplicate file systems for robustness or maintain a parity disk as a way to recover lost data if a single drive fails.

For fast ANSYS runs, the recommended configuration is a RAID0 setup using 4 or more disks and a fast RAID controller. These fast I/O configurations are inexpensive to put together for desktop systems and can achieve I/O rates in excess of 200 MB/sec. It is worth noting that not all software will automatically take advantage of a RAID configuration, and ANSYS has only recently begun to take advantage of this type of configuration.

Ideally, a dedicated RAID0 disk configuration for ANSYS runs is recommended. This dedicated drive should be regularly defragmented or reformatted to keep the disk clean. Using a dedicated drive for ANSYS runs also separates ANSYS I/O demands from other system I/O during simulation runs. Cheaper permanent storage can be used for files after the simulation runs are completed. *Table A.2, "I/O Hardware"* summarizes I/O hardware and disk recommendations for ANSYS users.

Another key bottleneck for I/O on many systems comes from using centralized I/O resources that are shared across a relatively slow company interconnect. Very few system interconnects can sustain 100 MB/sec or more for the large files that ANSYS reads and writes. Centralized disk resources can provide high bandwidth as well as very large capacity to multiple servers, but such a configuration requires an expensive high-speed interconnect and enough independent paths to the disk resource to supply each server independently if they are to be used at the same time for multiple jobs. Today, low cost RAID0 arrays can be added locally to most machines to achieve HPC I/O performance.

Finally, you should be aware of alternative solutions to I/O performance that may not work well. Some ANSYS users may have experimented with eliminating I/O in ANSYS by increasing the number and size of internal ANSYS file buffers. This strategy only makes sense when the amount of physical memory on a system is large enough so that all ANSYS files can be brought into memory. However, in this case file I/O is already in memory on both 64-bit Windows and Linux systems using the system buffer caches. This approach of adjusting the file buffers wastes physical memory because ANSYS requires that the size and number of file buffers for each file is identical, so the memory required for the largest files determines how much physical memory must be reserved for each file opened (over five ANSYS files are opened in a typical solution). All of this file buffer I/O comes from the user scratch memory in ANSYS, making it unavailable for other system functions or applications that may be running at the same time.

Another alternative approach to avoid is the so-called RAM disk. In this configuration a portion of physical memory is reserved, usually at boot time, for a disk partition. All files stored on this RAM disk partition are really in memory. Though this configuration will be faster than I/O to a real disk drive, it requires that the user have enough physical memory to reserve part of it for the RAM disk. Once again, if a system has enough memory to reserve a RAM disk, then it also has enough memory to automatically cache the ANSYS files. The RAM disk also has significant disadvantages in that it is a fixed size, and if it is filled up the job will fail with no warning.

The bottom line for minimizing I/O times in ANSYS and DANSYS is to use as much memory as possible to avoid I/O and use multiple disk RAID arrays in a separate work directory for the ANSYS working directory. HPC I/O is no longer a high cost addition if properly configured and understood.

Chapter 3: Parallel Processing

This chapter discusses how the ANSYS parallel processing capabilities can be used to maximize performance. The following topics are available:

3.1. SMP and DMP Processing
3.2. Recommended Number of Processors
3.3. Memory Considerations for Parallel Processing
3.4. I/O Considerations for Parallel Processing

3.1. SMP and DMP Processing

Shared memory computing systems have been available and in use for many years by HPC users. Over the years, SMP systems have developed sophisticated memory architectures that allow many processors to share memory across multiple nodes or processor boards while maintaining cache coherency. In recent years, SMP hardware systems have reached even higher levels of capabilities as the prices continue to fall. It is now possible to run ANSYS on high-end SMP servers that have hundreds of processors and up to several terabytes of real physical memory. At the same time, deskside SMP systems have emerged in the market with up to 16 processors and 16 to 128 GB of memory. Dual- and quad-core processors have made every desktop system a parallel SMP computer. The advantage of a single memory image is that users can solve the largest problems possible for a given amount of memory if ANSYS can access all of the memory to build the model, solve it, and visualize the results.

Distributed memory processing, on the other hand, can be used on any computer system that has an interconnect between physically separated processors. A major attraction of the DMP model is that very large parallel systems can be built using commodity priced components, or powerful computing resources can be assembled using a cluster of idle desktop systems. In practice, the promise of harnessing the power of idle desktops at night for a computationally demanding application like ANSYS has been unfulfilled due to difficulties in system setup and slow communication between processors. Instead, dedicated clusters, often put together in a single rack mounted server system, are used for running distributed applications. These systems include dedicated high speed interconnects which often transfer data between independent processors at speeds that approach the speed at which memory is moved on an SMP machine. The main benefit of these DMP systems is higher parallel efficiency due to better parallel programming models and reduced system cost. The Windows Cluster Computing environment has demonstrated that very low cost cluster resources can be deployed within an organization and used transparently as a compute engine for ANSYS. This extends the simulation capability of desktop users to a domain formerly known only to users of the very high-end, expensive SMP servers.

The above discussion describes SMP and DMP hardware. It is important to note that the SMP version of ANSYS can only run on a single machine (that is, it cannot run across machines). However, the DMP version of ANSYS, known as Distributed ANSYS (DANSYS), can run using multiple processors on a single machine (SMP hardware), and it can be run across multiple machines using one or more processors on each of those machines (DMP hardware).

3.2. Recommended Number of Processors

For SMP systems, ANSYS can effectively use 2 to 4 processors in most cases. For very large jobs and for higher-end servers (such as the HP Integrity series, IBM Power 5, SGI Altix, etc.), users may see reduced wall clock on 6 or even 8 processors for important compute bound portions of ANSYS (for example, sparse matrix factorization). In most cases, no more than 4 processors or cores should be used for a single ANSYS job using SMP parallel.

DANSYS performance often exceeds SMP ANSYS. The speedup achieved with DANSYS, compared to that achieved with SMP ANSYS, improves as more processors are used. DANSYS has been run using as many as 32 processors, but usually is most efficient with 8 to 12 processors. The best speedups in DANSYS are often obtained in the new iterative Lanczos solver, LANPCG. Speedups of over 6X have been obtained on 8 processors for this analysis type,

which relies on multiple linear system solves using the PCG iterative solver. For general static analyses or nonlinear runs, speedups between 3X and 6X on 8 processors are common across most DANSYS platforms.

3.3. Memory Considerations for Parallel Processing

Memory considerations for SMP ANSYS are essentially the same as for ANSYS on a single processor. All shared memory processors access the same user memory, so there is no major difference in memory demands for SMP ANSYS.

DANSYS memory usage requires more explanation. When running DANSYS using *n* processors, *n* DANSYS processes are started. The first of these processes is often referred to as the master process, while the remaining *n - 1* processes are often referred to as the slave processes.

In DANSYS, the master process always requires more memory than the slave processes. The master process reads the entire ANSYS input file and does the initial model decomposition. In addition, while much of the memory used for the **SOLVE** command scales across the processes, some additional solver memory requirements are unique to the master process. Generally, the machine that contains the master process should have twice as much memory as all other machines used for the run.

Although not a requirement for cluster systems, the memory available on the machine containing the master process will determine the size of problem that DANSYS can solve. A cluster of 8 nodes with 4 GB each cannot solve as large a problem as an SMP machine that has 32 GB of memory, even though the total memory in each system is the same. Upgrading the master node to 8 GB, however, will allow the cluster to solve a similar-sized problem as the 32 GB SMP system.

3.4. I/O Considerations for Parallel Processing

Reducing the I/O demands ANSYS places on a system is even more important when running in parallel, especially on systems that write to one disk, such as a cluster with a single NFS-mounted disk or a mulitcore SMP system. It is strongly recommended that you use the **OUTRES** command to limit the data written to the results file (`Jobname.RST`). Also, you may increase the file buffer size (**/CONFIG**,SZBIO and **/CONFIG**,NBUF) enough that the .EMAT and .ESAV files are kept in memory.

Chapter 4: Recommended System Configurations

This section recommends system configurations to define HPC systems for ANSYS users, from desktop systems running Windows up through high-end server recommendations. The following topics are covered:

4.1. Operating Systems
4.2. Single-box (SMP) Configurations
4.3. Cluster Configurations
4.4. Choosing Systems to Meet User Needs

All of the recommended system configurations have balanced CPU, memory, and I/O to deliver HPC performance at every price point. The system configurations are described for SMP configurations and true cluster systems. *Table A.3, "Recommended Single Box System Configurations (SMP ANSYS or DANSYS)"* and *Table A.4, "Recommended Cluster System Configurations (DANSYS)"* summarize the single-box and cluster configurations described here.

4.1. Operating Systems

With the increasing use of commodity processors in HPC systems and the release of 64-bit Windows, users now have a choice of HPC operating systems, each having legitimate merits. These choices include 64-bit Windows, 64-bit Linux, and vendor specific UNIX operating systems such as HPUX, IBM AIX, and Sun Solaris. Linux 64-bit is not a single choice because of multiple OS versions (Red Hat, SuSE, and others), but it does provide a common look and feel for Linux-based systems. We will discuss some of the issues with each choice.

4.1.1. 32-bit versus 64-bit Operating Systems

The amount of memory that each running ANSYS process can address (or utilize) is limited depending on the operating system being used. Thirty two-bit operating systems have a theoretical peak memory address space limitation of 4 GB per process. In practice, this memory limit is set even lower, at 2 GB or 3 GB per process, by the Windows or Linux 32-bit operating system. Sixty four-bit operating systems support memory address spaces that extend to several terabytes of address space and more, well beyond the availability of physical memory in systems today.

On 32-bit systems, the physical memory often exceeds the available address space. Thus, on these systems ANSYS may be limited to allocating less memory than is physically available. On 64-bit operating systems, users encounter the opposite case: more address space than physical memory. With these systems, users often run out of physical memory, and so either the user or the operating system itself will increase the virtual memory (or swap file) size. While this effectively increases the size of the largest job that can be solved on the system by increasing the amount of memory available to ANSYS, this virtual memory is actually hard drive space, and its usage can drastically slow down ANSYS performance. Those using 64-bit operating systems should note this effect and avoid using virtual memory as much as possible.

Most applications, including ANSYS, predominantly use 32-bit integer variables. This means that integer overflows can still occur in integer operations regardless of whether a 32-bit or 64-bit operating system is used. Thirty two bit integers overflow at 2^{32} (~2 billion) on most systems. Much of ANSYS code still uses 32-bit integer values, and so the number of nodes, elements, and equations is limited by design to under 2 billion. This is still well below the model sizes that users are solving today, although billion element models are now within the capability of many large memory computers.

Starting with release 10.0 of ANSYS, a special large memory version of ANSYS is available for which all integers used for the sparse solver are 64-bit integers. This allows the sparse solver work array to exceed its previous limit of 2 billion double precision values (16 GB). This version of ANSYS is used whenever the -lm command line

option is specified (see *Section 5.1.1.3: Using the Large Memory (-lm) Option*). In addition, for release 11.0 the distributed sparse solver is, by default, all 64-bit integers. ANSYS will continue an orderly migration to all 64-bit integers while supporting legacy file issues and legacy code.

Within ANSYS, memory allocations have been changed to support 64-bit allocation sizes, and file addressing has also been modified to support 64-bit addressing. In other words, memory use for ANSYS can exceed 16 GB, and files can be virtually unlimited in size. In fact, ANSYS has been used for solving models requiring over several hundred gigabytes of memory and file sizes.

4.1.2. Windows

In the past, 32-bit versions of Windows (for example, Windows 2000 and Windows XP) were the only realistic choice for desktop ANSYS users. While ANSYS runs well on these operating systems, it is not recommended for HPC systems because of the memory limitations of the 32-bit operating system. For models less than 250-300k DOFs, 32-bit Windows can still function very well using the sparse direct solver and can even extend to a few million DOFs using the PCG iterative solver. However, I/O performance is usually very poor, and these large runs will render the desktop system useless for any other activity during the simulation run.

Windows 64-bit has changed the landscape for desktop users, offering a true HPC operating system on the desktop. In ANSYS Release 11.0, the first use of Windows specific I/O is introduced, enabling desktop systems to obtain high performance I/O using very inexpensive RAID0 array configurations. Additional Windows OS features will be exploited in future releases that will continue to enhance Windows 64-bit performance on desktop systems.

4.1.2.1. Memory Usage on Windows 32-bit Systems

If you are running on a 32-bit Windows system, you may encounter memory problems due to Windows' limitation on the amount of memory programs can address (including ANSYS). Windows 32-bit systems limit the maximum memory address space to 2 GB. This 2 GB of memory space is typically not contiguous, as the operating system DLLs loaded by ANSYS often fragment this address space. Setting the /3GB switch in the system boot.ini file will add another gigabyte of memory address space for ANSYS to use; however, this extra memory is not contiguous with the initial 2 GB.

Note

The file boot.ini is a hidden system file in Windows. It should be modified with care. The /3GB switch should always be added as an extra line in the boot.ini file rather than adding it as the only boot option. This file can be modified directly or through the Windows system control panel (**Control Panel > System > Advanced tab >Startup and Recovery: Settings > Edit option**).

For example a typical boot.ini file contains the following line:

```
multi(0)disk(0)rdisk(0)partition(4)\WINDOWS="Microsoft
Windows XP Professional" /noexecute=optin /fastdetect
```

To enable the extra 1 GB of address space, the following line is added to the boot.ini file. Note that this line is identical to the previous line except that the label has been changed to add 3GB, and the switch /3GB is added at the end of the line. At boot time, users can select the 3GB option or use the standard boot process. if any OS stability issues are encountered, users can revert to the standard boot option.

```
multi(0)disk(0)rdisk(0)partition(4)\WINDOWS="Microsoft
Windows XP Professional 3GB " /noexecute=optin /fastdetect /3GB
```

4.1.3. UNIX

UNIX operating systems have long been the choice of the HPC community. They are mature multi-user HPC operating systems that support high performance I/O, parallel processing, and multi-user environments. But, they are generally vendor specific and require some knowledge of specific vendor features to achieve the full benefit. ANSYS supports HPUX, IBM AIX, and Sun Solaris. These operating systems are still likely to be the choice of customers who purchase large high-end servers for multi-user environments.

4.1.4. Linux

Linux has replaced UNIX for many users in recent years. This operating system is increasingly used for cluster systems and is used on Intel Itanium and x86 processors as well as AMD Opteron processors. In addition, some traditionally proprietary UNIX vendors, such as IBM, now offer Linux operating systems for their own microprocessors. Other vendors use Linux as the operating system, but add additional HPC support. For example, SGI adds PowerPack software to their Linux OS to support additional MPI and I/O features.

The version issue for 64-bit Linux may cause problems for ANSYS users. Since Linux is an open source operating system, there is not a single point of control for maintaining compatibility between various libraries, kernel versions, and features. Before purchasing a Linux-based system, users should consult their ANSYS support provider to find what releases are supported and tested by ANSYS.

4.2. Single-box (SMP) Configurations

With the advent of multicore processors, almost all single-box configurations in use today have multiple processors. These processors all share the same physical memory. Larger servers may have more than one processor board, each with multiple CPU sockets and memory slots, but all processors access the same global memory image. These systems can be configured very inexpensively on the desktop with dual- and quad-core processors, but desktop systems will usually have poor I/O support for multiple processor runs, particularly for DANSYS runs. This disadvantage can be hidden by adding additional memory to improve system cache performance. One significant advantage of these systems is that they can use the global memory image to pre- and postprocess very large models.

Users should carefully weigh the pros and cons of using a powerful deskside server for DANSYS instead of a true cluster, if a system is being configured primarily to run ANSYS simulations. The deskside system will always have the advantage of a single large memory image for building and viewing very large model results. It is important to note that these machines also typically run DANSYS well, and the cost of deskside SMP servers is competitive with well configured clusters. See *Table A.3, "Recommended Single Box System Configurations (SMP ANSYS or DANSYS)"* for suggested configurations.

4.2.1. Desktop Windows 32-bit Systems and 32-bit Linux Systems

Desktop Windows 32-bit systems and 32-bit Linux systems, while quite effective for running small to medium-sized problems, are not considered true HPC systems due to their memory limit of 2 GB or 3 GB set by the Windows or Linux operating system. A recommended configuration for these systems is a dual-core processor, 4 GB of memory, and a single drive for system and applications software, with a separate RAID0 4-disk configuration for ANSYS work space.

4.2.2. Desktop Windows 64-bit Systems and 64-bit Linux Systems

Desktop Windows 64-bit systems and 64-bit Linux systems should have a minimum of 8 GB of memory. A minimum configuration would have a single dual-core processor. However, 2 dual-core processors is a very affordable upgrade and is recommended for ANSYS. The amount of memory should be chosen to comfortably run the ANSYS simulation for typically expected model sizes. Ideally, the system should have enough memory to run ANSYS with enough left over to cache the solver files in memory. A workstation with 8 GB of memory should run

models up to 300k to 500k DOFs with excellent I/O performance. For models approaching 1 million DOFs or more, doubling memory to 16 GB is highly recommended. Disk recommendations are the same as the RAID0 recommendation stated above for 32-bit desktop systems. However, if sufficient physical memory is purchased to avoid the I/O bottleneck, the disk I/O rate is not as critical for systems with 64-bit operating systems as it is for systems with 32-bit operating systems.

4.2.3. Deskside Servers

The deskside server is a new class of system that can have multiple functions for ANSYS users. These systems have 4 to 8 dual- or quad-core processors with recommended memory of 16 to 64 GB and a fast RAID0 array using 4 to 8 disks. They can be 64-bit Windows or 64-bit Linux systems. They can serve a single power user or serve as a shared compute resource for remote execution of ANSYS simulations. Deskside servers can run SMP ANSYS and distributed ANSYS well and are cost effective ways to deliver true supercomputer performance for very large ANSYS models.

Linux 64-bit systems can handle multiple parallel jobs at once, up to the limit of physical processors and memory available, and most can handle the I/O demands if they are configured as recommended. Windows 64-bit or Windows Server 2003 can be used on these deskside servers as well, with very good performance. Multiple-user experience with Windows servers is currently not expected to be as fast as 64-bit Linux, but improvements are expected over time. Deskside servers can also be used as a remote solve engine for large Workbench models. A single deskside server used as a remote solve resource can be an effective addition to a small group that uses desktop Windows 64-bit , or even Windows 32-bit systems, to run Workbench simulations.

When considering a deskside server with larger memory versus a desktop system, keep in mind that many larger memory servers are currently being configured using slower bus speeds and processors because they are designed primarily for handling large data operations rather than intense processing demands. Large memory servers using x86 commodity processors often have slower bus speeds (800 MHz to 1066 MHz) and use older Xeon processors. First generation dual-core processors are no match for the new core micro-architecture Xeons. This is true for both single core performance and parallel performance. The faster bus speeds used with core micro-architecture processors also include an extra path to memory to support multicore processors. While large memory will greatly reduce I/O times in ANSYS, the slower processors in these large memory commodity-priced servers may mean that a desktop system using the latest dual- or quad- core processors may outperform the large memory, more expensive deskside system. It is very important to understand the performance value of the new x86 processor technologies.

For all but the most time critical simulation requirements, the most cost effective HPC systems today should use the latest multicore processors with all memory slots filled using memory parts that are reasonably priced. At any given time, the latest high capacity memory is always more expensive and is 2 to 4 times higher in cost per GB of memory. Cost effective memory is priced linearly per GB of memory. In the current market, memory parts up to 2 GB per SIMM are priced linearly, but 4 GB SIMMS will soon follow. The memory saved using 2 GB SIMMS is more than enough to purchase a good RAID0 disk array, which will extend good balanced performance for larger models that require a large amount of I/O. Processor cost is also directly related to the clock frequency. It is more cost effective to use a slower processor that is within the latest core micro-architecture and invest the savings in filling all memory slots, than to buy the fastest processors to put in a memory-starved system with a slow disk.

4.2.4. High-end Centralized SMP Servers

Large servers are the most expensive, but also the most powerful resource for solving the largest and most challenging ANSYS simulations. They can easily serve a typical workload of multiple parallel jobs. These systems typically have 64 GB of memory or more, 8 processors or more, and a substantial disk configuration—usually 8 or more disks arranged to deliver fast multiuser I/O capacity. They would typically have multiple I/O controllers to support the multiuser, multijob demands.

These machines make excellent parallel processing resources for SMP ANSYS or Distributed ANSYS (DANSYS). They also work well as remote solve resources for ANSYS Workbench users who want a larger resource for faster solve runs, freeing desktop systems for other work. Typically, large SMP servers do not have the graphics capabilities that desktop systems have. The operating system for large SMP servers is almost always UNIX or 64-bit Linux. It is possible to configure large memory Windows servers, but the support and performance for multiple users in Windows HPC systems is not yet proven.

4.3. Cluster Configurations

True cluster systems have more than one independent processor per memory unit, usually in a separate chassis rack-mounted unit or CPU board, using some sort of interconnect to communicate between the independent processors through MPI software calls. Many large companies are investing in cluster computing resources. ANSYS will run on a single node of these systems if the system is a supported ANSYS platform. Distributed ANSYS (DANSYS) will run in parallel on a cluster system that is a supported ANSYS platform if a supported interconnect is used and the appropriate MPI software is properly installed.

Due to increased memory demands and I/O requirements for DANSYS, cluster configurations that work well for DANSYS must have sufficient memory and I/O capacity. Quite often, cluster systems are created using identical hardware configurations for all machines (often referred to as nodes) in the cluster. This setup would work well for parallel software that is able to run in a perfectly distributed fashion with all tasks being perfectly parallel. This is not the case for Distributed ANSYS simulations.

When launching Distributed ANSYS, there are *n* processes that are started. As discussed earlier, the first of these processes is often referred to as the master process. The machine or node on which the master process runs is often referred to as the head node, or node 0. It is highly recommended that this node contain at least twice as much memory as the other nodes in the cluster. The reason is that the master process performs all of the pre- and postprocessing operations, and it handles all operations involving the database. Also, some operations in both the distributed direct and iterative solvers are not parallelizable and, thus, can require extra amounts of memory. The recommended minimum configuration for a cluster system used to solve large models of up to a few million DOFs would be 8 GB on node 0 and 4 GB on the other nodes. More modest memory sizes can be used if the simulation models are smaller.

Each DANSYS process does its own I/O to a unique set of files. If each processing node has independent disk resources, a natural scaling of I/O performance will be realized. Some cluster configurations assume minimal I/O at processing nodes and, therefore, have a shared I/O resource for all of the nodes in a cluster. This configuration is not recommended for use with DANSYS because, as more processors are used with DANSYS, more of a burden is put on this I/O resource to keep up with all of the I/O requests. A cluster sized to run DANSYS should have large memory and good I/O capability at each node. Some hardware vendors have recognized these demands and recommend systems targeted for structural applications such as ANSYS separate from other less I/O and memory intensive applications.

All cluster systems must have an interconnect, which is used to pass messages between the DANSYS process. The cheapest interconnect, often using on-board built-in communications hardware, is Ethernet. Many company networks run today using either Fast Ethernet or Gigabit Ethernet (GigE). Fast Ethernet can reach transfer speeds around 10 MB/sec, while GigE typically runs around 100 MB/sec. A new version of Ethernet known as 10GigE is becoming available, but is still somewhat expensive. Transfer rates for this interconnect are reportedly around 1 GB/sec.

GigE is a minimum configuration for DANSYS, but it is adequate to obtain solution time improvements. Other faster interconnects are Infiniband and Myrinet. Infiniband interconnects are capable of inter-processor communication in ANSYS solvers of over 800 MB/sec, and Myrinet is capable of 200+ MB/sec. High-end interconnects still cost more than the processors and require additional hardware expenditures in most cases.

While a rate improvement from 100 MB/sec to nearly 1 GB/sec sounds significant, the data movement between MPI processes is usually only a few GB, even on a larger model, so the time saved with an expensive interconnect is often insignificant. For DANSYS performance, it is wiser to invest in more memory and faster local I/O than to add expensive interconnects. This surprising conclusion is due to the observation of DANSYS performance on cluster systems, and the fact that DANSYS jobs do not currently scale past 8 or 16 cores and are typically not run on processors with hundreds of cores. On larger systems, faster interconnects are important for overall system performance, but such systems typically already have faster interconnects.

See *Table A.4, "Recommended Cluster System Configurations (DANSYS)"* for suggested cluster configurations.

4.3.1. Windows or Linux Personal Cluster

Windows and Linux personal clusters are relatively new in the HPC community, but may be cost effective ways to improve performance for Windows users. The personal cluster system is becoming an increasingly powerful resource for higher-end users. These systems are small, quiet, and have modest power requirements. They have 4 to 8 dual-core processors and may be configured as 4 small CPU boards with memory and an interconnect, or as a single CPU board system with multiple CPU sockets. The interconnect for systems that have multiple, separate boards would usually be built-in on each board using GigE, and would come already configured and setup. Recommended memory for these systems is at least 4 GB per processor, with 8 GB on node 0, if possible. These systems would function best as a remote solve resource for Workbench users.

You can configure the Windows cluster using Windows XP, along with an interconnect and the MPICH MPI software that comes with Distributed ANSYS. An alternative configuration, however, would be to install Microsoft Compute Cluster Server 2003, which allows ANSYS and Workbench users to easily setup and configure a personal cluster. This version of Windows is a 64-bit operating system meant for users seeking an affordable high performance computing system. It contains a job scheduler and has the Microsoft MPI software built into the operating system. These features allow you to launch a Distributed ANSYS job seamlessly from the ANSYS product launcher or from the Workbench Remote Solve Manager (RSM).

Although presented above as a Windows solution, this hardware configuration can obviously support the Linux operating system as well. In addition, since the Remote Solve Manager supports Linux servers, this type of system with Linux installed could function as a remote solve resource for Workbench users.

4.3.2. Rack Cluster

Rack-mounted cluster systems use 64-bit Linux, Windows 64-bit, or Windows Compute Cluster Server 2003 operating systems. They are usually 1 or 2 socket processor boards with a local disk and memory of 4 to 8 GB per rack unit. Inexpensive rack systems that use 32-bit Windows or 32-bit Linux operating systems or that have no local I/O capability are not recommended for DANSYS. Infiniband interconnect is best for these systems, but GigE is sufficient for 4 to 8 core runs using DANSYS.

Each rack unit must be connected to other units in the cluster using fast switches. Local disk resources should be 40 to 80 GB. They do not need to be RAID0 arrays, since the I/O performance of DANSYS will naturally scale as each processor does I/O to separate disks. However, one cost effective I/O configuration for a rack cluster is to use dual drives in each node, configured using RAID0 as a scratch disk for DANSYS jobs. Dual- and quad-core configurations in rack systems will present more I/O demands on each rack unit because each DANSYS process will do independent I/O. Modest RAID0 configurations using 2 drives on each node improve the local I/O performance at a very reasonable cost.

4.3.3. Cluster of Large Servers

One way to reduce the cost of interconnects and solve the demand for large memory on the master process in DANSYS is to use a cluster of two or more large servers. Each server would contain two dual- or quad-core processors with 16 GB or more of memory. The MPI interconnect between these processors is very fast since the

communication can use shared memory within a single box. The processing power on a 4-core box can be multiplied by connecting two or more similarly configured SMP boxes using a fast interconnect. However, instead of multiple expensive fast switches between every processor, only one switch is required for each box. I/O requirements on these systems are concentrated within each SMP box. A RAID0 configured array with 200 GB or more of capacity is recommended for each server. Disk space should include additional space reserved for a swap space file equal to the size of physical memory.

4.3.4. Large Corporate-wide Cluster

Many large companies now have large cluster compute resources with 64 or more processors. Individual applications run on a subset of the large cluster resource. It is generally not possible to specify which nodes a DANSYS run will run on. However, a 4 to 8 processor DANSYS job can run effectively on such a system using remote job execution. To maximize DANSYS performance, the processing nodes should have 4 to 8 GB of memory and should be capable of high performance I/O on each node. Many large cluster systems are not configured with sufficient memory per processor or I/O capacity to run DANSYS well, even though the total memory of a large cluster system may seem very large. For example, a 128 processor cluster with 128 GB of memory would not be able to run DANSYS for large models, but a 16 node cluster with 128 GB of memory would be an excellent configuration for large DANSYS models.

4.4. Choosing Systems to Meet User Needs

The variety of HPC systems available for ANSYS users has never been greater. Choosing the best system configuration often requires balancing competing demands. However, the availability of Windows 64-bit desktop HPC systems and deskside large memory SMP servers suggest a new strategy for meeting ANSYS computing demands. It is no longer the case that only the high end users have access to parallel processing, large memory, and high performance I/O. The choice of HPC hardware can now begin with single-user desktop configurations that maximize system performance. Larger deskside, server, and cluster solutions should be added as simulation demand increases or extends to multiple users within an organization. *Table A.5, "HPC Recommendations for Various ANSYS User Scenarios"* lists several ANSYS user scenarios with hardware recommendations for each situation.

Chapter 5: ANSYS Memory Usage and Performance

This section explains memory usage in ANSYS and gives recommendations on how to manage memory in order to maximize ANSYS performance. System memory has already been described as the single most important factor in HPC system performance. Since solver memory usually drives the memory requirement for ANSYS, the details of solver memory usage are discussed here. Solver memory for direct and iterative solvers, as well as ANSYS and DANSYS implementations, are described and summarized in *Table A.6, "ANSYS and DANSYS Direct Sparse Solver Memory and Disk Estimates "* and *Table A.7, "ANSYS and DANSYS Iterative PCG Solver Memory and Disk Use Estimates"*. The chapter concludes with a brief discussion of ANSYS memory requirements for pre- and postprocessing.

5.1. Linear Equation Solver Memory Requirements

ANSYS offers two types of linear equation solvers: direct and iterative. There are SMP and DMP differences for each of these solver types. This section describes the important details for each solver type and presents, in tabular form, a summary of solver memory requirements. Recommendations are given for managing ANSYS memory use to maximize performance.

The following topics are covered:

5.1.1. Direct (Sparse) Solver Memory Usage
5.1.2. Iterative (PCG) Solver Memory Usage
5.1.3. Modal (Eigensolvers) Solver Memory Usage

5.1.1. Direct (Sparse) Solver Memory Usage

The sparse solver in ANSYS is still the default solver for virtually all analyses. It is the most robust solver in ANSYS, but it is also computationally and I/O intensive. The sparse solver used in ANSYS is designed to run in different modes of operation, depending on the amount of memory available. Understanding how to influence which mode of operation is used can have a significant impact on runtime.

Memory usage for a direct sparse solver is determined by several steps. In ANSYS, the matrix that is input to the sparse solver is assembled entirely in memory before being written to the FULL file. The sparse solver then reads the FULL file and inputs the matrix, processes the matrix, factors the matrix, and computes the solution. Direct method solvers factor the input matrix into the product of a lower and upper triangular matrix in order to solve the system of equations. For symmetric input matrices (most of ANSYS matrices are symmetric), only the lower triangular factor is required since it is equivalent to the transpose of the upper triangular factor. Still, the process of factorization produces matrix factors which are 10 to 20 times larger than the input matrix. The computation of this factor is very compute intensive. In contrast, the solution of the triangular systems is I/O or memory access-dominated with few computations required.

The following are rough estimates for the amount of memory needed for each step when using the sparse solver for most 3-D analyses. For non-symmetric matrices or for complex value matrices (as found in harmonic analyses), these estimates approximately double.

- The amount of memory needed to assemble the matrix in memory is approximately 1 GB per million DOFs.
- The amount of memory needed to hold the factored matrix in memory is approximately 10 to 20 GB per million DOFs.

It is important to note that the sparse solver in ANSYS is not the same as the distributed sparse solver in DANSYS. While the fundamental steps of each solver are the same, each sparse solver is actually a different solver, and there are subtle differences in the modes of operation. These differences will be explained in the following sections. *Table A.6, "ANSYS and DANSYS Direct Sparse Solver Memory and Disk Estimates "* summarizes the direct solver memory requirements.

5.1.1.1. Out-of-Core Factorization

For out-of-core factorization, the factored matrix is held on disk. There are two possible out-of-core modes when using the sparse solver in ANSYS. The first of these modes is a minimum memory method that is dominated by I/O. However, this mode is rarely used and should be avoided whenever possible. The minimum memory mode occurs whenever the sparse solver memory available is smaller than the largest front processed by the sparse solver. Fronts are dense matrix structures within the large matrix factor. Fronts are processed one at a time, and as long as the current front fits within available sparse solver memory, the only I/O required for the front is to write out finished matrix columns from each front. However, if insufficient memory is available for a front, a temporary file in ANSYS is used to hold the complete front, and multiple read/write operations to this file (`Jobname.LN32`) occur during factorization. The minimum memory mode occurs most often with models that have very large fronts, either due to the use of constraint equations that couple many nodes to a single node, or due to bulky 3-D models that use predominantly higher-order elements. The total amount of I/O performed for minimum memory mode is often 10 times greater then the size of the entire matrix factor file. Note that this minimum memory mode does not exist for the distributed sparse solver found in ANSYS.

Fortunately, if sufficient memory is available for the assembly process, it is almost always more than enough to run the sparse solver factorization in the other out-of-core mode known as optimal out-of-core mode. This mode uses some additional memory to avoid the excessive I/O done in the minimum memory mode, but still writes the factored matrix to disk and, thus, attempts to achieve an optimal balance between memory usage and I/O. On Windows 32-bit systems, if the optimal out-of-core memory exceeds 1200 to 1500 MB, minimum core mode may be required. Sometimes the default memory path for large models will use minimum core mode if the starting memory for the sparse solver is below the optimal out-of-core mode. In most cases ANSYS will automatically run the sparse solver using optimal out-of-core memory. The optimal out-of-core memory mode is the default for the distributed sparse solver.

5.1.1.2. Incore Factorization

Incore factorization requires that the factored matrix be held in memory and, thus, often requires 10 to 20 times more memory than out-of-core factorization. However, with larger memory systems readily available, many users will benefit from incore factorization. A model with 250k DOFs can, in many cases, be factored using 2.5 GB of memory—easily achieved on the recommended desktop HPC systems with 8 GB of memory. Users can run incore in ANSYS Release 11.0 and earlier using several different methods. The simplest way to set up an incore run is to use the **BCSOPTION**,,INCORE command. This option tells the sparse solver interface routines to try allocating a block of memory sufficient to run incore after solver preprocessing of the input matrix has determined this value. However, this method requires preprocessing of the input matrix using an initial allocation of memory to the sparse solver.

Another way to get incore performance with the sparse solver in ANSYS Release 11.0 and earlier is to start the sparse solver with enough memory to run incore. Users can start ANSYS with an initial large -m allocation (see *Section 1.1.1: Specifying Memory Allocation in ANSYS*) such that the largest block available when the sparse solver begins is large enough to run incore. This method is subject to the limitation of 16 GB of total sparse solver memory in standard ANSYS runs, but is virtually unlimited in the large memory (-lm) version. This method will obtain incore memory with a lower peak memory usage than the simpler method described above, but it requires prior knowledge of how much memory to allocate to obtain incore memory for the sparse solver.

The incore factorization should be used only when the computer system has enough memory to easily factor the matrix incore. Users should avoid using all of the available system memory or extending into virtual memory to obtain an incore factorization. However, users who have long-running nonlinear simulations with less than 500k DOFs and who have 8 GB or more of system memory should understand how to use the incore factorization to improve elapsed time performance.

The **BCSOPTION** command in ANSYS controls sparse solver memory modes and also enables performance debug summaries. See the documentation on this command for usage details. Sparse solver memory usage statistics are usually printed in the output file and can be used to determine the memory requirements for a given model, as well as the memory obtained from a given run. Below is an example output.

```
Memory available for solver =              322.05 MB
Memory required for in-core =             1458.89 MB
Optimal memory required for out-of-core =  280.04 MB
Minimum memory required for out-of-core =   26.96 MB
```

This sparse solver run required 1459 MB to run incore, 281MB to run in optimal out-of-core mode, and just 27 MB to run in minimum out-of-core mode. "Memory available for solver" shows the amount of memory that was used for this run to be just above the optimal out-of-core memory requirement.

The distributed sparse solver can also be run in the incore mode. However, currently, the only way to force this mode is to use the **DSPOPTION**,,INCORE command. Similar memory usage statistics for this solver are also printed in the output file for each DANSYS process. When running the distributed sparse solver using multiple processors on a single box or when running on a cluster with poor I/O performance, using the incore mode can significantly improve the solver time as the costly I/O time is avoided.

5.1.1.3. Using the Large Memory (-lm) Option

The large memory version of ANSYS was created beginning with Release 10.0 and extending to Release 11.0 to provide an all 64-bit integer version of the sparse solver. The large memory version of ANSYS is used whenever the -lm argument is added to the ANSYS command line. This version uses a sparse solver library that is compiled using 64-bit integers instead of the default library that uses 32-bit integers (as in the regular version of ANSYS). This change allows the sparse solver to access and address a work array that is subject to the limits of 32-bit integer addressing. By increasing the integer size, the large memory version uses more memory for each integer stored, but allows users with very large memory systems to access virtually unlimited amounts of memory for the sparse solver. Thus, the sparse solver (and Block Lanczos eigensolver) runs incore, eliminating costly I/O for larger jobs.

The -lm version is most effective for large Block Lanczos runs used to compute 100 or more modes. Running large modal analyses incore may save more than a terabyte of I/O for large models. However, the incore -lm version also competes with system cache performance on very large memory machines. If a large memory machine has sufficient memory to cache the large matrix factor files, the performance of out-of-core solves will be very close to incore performance. The -lm version should not be used on a system when incore runs will use up all available memory. For most multi-user or multi-job systems, the large memory is better used for reducing I/O costs using the system cache rather than running very large ANSYS jobs incore. If the system is not heavily loaded and the incore memory requirement fits comfortably within the available physical memory, then the -lm version can be used to deliver maximum performance for a single large job.

Because the distributed sparse solver in DANSYS uses 64-bit integers by default, DANSYS does not use the -lm option. DANSYS runs as the -lm option does for the SMP sparse solver and can allocate and access work arrays exceeding 2 Gwords (GB). In version 12.0 of ANSYS, all sparse solvers will use 64-bit integers, and the -lm version will no longer be necessary.

5.1.1.4. Partial Pivoting

Sparse solver partial pivoting is an important detail that may inhibit incore factorization. Pivoting in solvers refers to a dynamic reordering of rows and columns to maintain numerical stability. This reordering is based on a test of the size of the diagonal (called the pivot) in the current matrix factor column during factorization. Pivoting is not required for most ANSYS analysis types, but it is enabled when certain element types and options are used (for example, Lagrange contact and mixed u-P formulation). When pivoting is enabled, the size of the matrix factor cannot be known before the factorization; thus, the incore memory requirement cannot be accurately computed. As a result all pivoting, enabled factorizations in ANSYS are out-of-core.

5.1.2. Iterative (PCG) Solver Memory Usage

ANSYS iterative solvers offer a powerful alternative to more expensive sparse direct methods. They do not require a costly matrix factorization of the assembled matrix, and they always run in memory and do only minimal I/O. However, iterative solvers proceed from an initial random guess to the solution by an iterative process and are dependent on matrix properties that can cause the iterative solver to fail to converge in some cases. Hence, the iterative solvers are not the default solvers in ANSYS.

The most important factor determining the effectiveness of the iterative solvers for ANSYS simulations is the preconditioning step. The preconditioned conjugate gradient (PCG) iterative solver in ANSYS now uses two different proprietary preconditioners which have been specifically developed for a wide range of element types used in ANSYS. The newer node-based preconditioner (added at Release 10.0) requires more memory and uses an increasing level of difficulty setting, but it is especially effective for problems with poor element aspect ratios.

The specific preconditioner option can be specified using the *Lev_Diff* argument on the **PCGOPT** command. *Lev_Diff* = 1 selects the original element-based preconditioner for the PCG solver, and *Lev_Diff* values of 2, 3, and 4 select the new node-based preconditioner with differing levels of difficulty. Finally, *Lev_Diff* = 5 uses a preconditioner that does require factorization of the assembled global matrix. This last option (which is discussed in more detail later) is mainly used for the PCG Lanczos solver (LANPCG) and is only recommended for smaller problems where there is sufficient memory to use this option. ANSYS uses heuristics to choose the default preconditioner option and, in most cases, makes the best choice. However, in cases where the heuristic choice is for a high level of difficulty, it may be necessary to reduce memory requirements by setting a lower level of difficulty (via the **PCGOPT** command) in order to obtain a solution on your existing hardware.

The basic memory formula for ANSYS iterative solvers is 1 GB per million DOFs. Using a higher level of difficulty preconditioner raises this amount, and higher order elements also increase the basic memory requirement. An important memory saving feature for the PCG solvers is implemented for several key element types in ANSYS. This option, invoked via the **MSAVE** command, avoids the need to assemble the global matrix by computing the matrix/vector multiplications required for each PCG iteration at the element level. The **MSAVE** option can save up to 70 percent of the memory requirement for the PCG solver if the majority of the elements in a model are elements that support this feature. **MSAVE** is automatically turned on for some linear static analyses when SOLID92/187 and/or SOLID95/186 elements that meet the **MSAVE** criteria are present. It is turned on because it often reduces the overall solution time in addition to reducing the memory usage. It is most effective for these analyses when dominated by SOLID95/186 elements using reduced integration, or by SOLID92/187 elements. For large deflection nonlinear analyses, the **MSAVE** option is not on by default since it increases solution time substantially compared to using the assembled matrix for this analysis type; however, it can still be turned on manually to achieve considerable memory savings.

The total memory usage of the DANSYS iterative solvers is higher than the corresponding SMP versions in ANSYS due to some duplicated data structures required on each processor for the DMP version. However, the total memory requirement scales across the processors so that memory use per processor reduces as the number of processors increases. The preconditioner requires an additional data structure that is only stored and used on the master processor, so the memory required for the master processor is larger than all other processors. Peak

memory usage for the PCG solvers usually occurs during preconditioner construction. *Table A.7, "ANSYS and DANSYS Iterative PCG Solver Memory and Disk Use Estimates"* summarizes the memory requirements for ANSYS iterative solvers, including DANSYS.

5.1.3. Modal (Eigensolvers) Solver Memory Usage

Finding the natural frequencies and mode shapes of a structure is one of the most computationally demanding tasks in ANSYS. Specific equation solvers, called eigensolvers, are used to solve for the natural frequencies and mode shapes. ANSYS offers two eigensolvers for modal analyses: the sparse solver-based Block Lanczos solver and the PCG Lanczos solver.

The memory requirements for each eigensolver are related to the memory requirements for the sparse and PCG solvers used in each method, as described above. However, there is additional memory required to store the mass matrices as well as blocks of vectors used in the Lanczos iterations. For the Block Lanczos solver, I/O is a critical factor in determining performance. For the PCG Lanczos solver, the choice of the preconditioner is an important factor.

5.1.3.1. Block Lanczos Solver

The Block Lanczos solver in ANSYS (**MODOPT**,LANB) uses the sparse direct solver. However, in addition to requiring a minimum of two matrix factorizations, the Block Lanczos algorithm also computes blocks of vectors that are stored on files during the Block Lanczos iterations. The size of these files grows as more modes are computed. Each Block Lanczos iteration requires multiple solves using the large matrix factor file (or in-memory factor if the incore option is used).

Users should be aware of some key concepts related to memory use for Block Lanczos. First, the total memory used for a Block Lanczos run is in one contiguous work array. This array is used from both ends towards the middle; when overlap occurs, I/O is required. Second, if the available memory is below recommended memory in a Block Lanczos run, the block size used internally for the Lanczos iterations will be reduced. Smaller block sizes will require more block solves, the most expensive part of the Lanczos algorithm for I/O performance. Finally, multiple matrix factorizations may be required for a Block Lanczos run.

The algorithm used in ANSYS decides dynamically whether to refactor using a new shift point or to continue Lanczos iterations using the current shift point. This decision is influenced by the measured speed of matrix factorization versus I/O dominated solves. This means that unexpected performance characteristics can occur when hardware is changed or when memory is changed from out-of-core to incore estimates. The best general rule for Block Lanczos is to be generous with the memory allocated for a Lanczos run (see *Section 1.1.1: Specifying Memory Allocation in ANSYS*). Start ANSYS with a large initial memory allocation (-m) that is estimated from guidelines in *Table A.8, "ANSYS and DANSYS Eigensolver Memory and Disk Space Estimates"*, if possible.

5.1.3.2. PCG Lanczos Solver

The new PCG Lanczos solver (**MODOPT**,LANPCG) represents a breakthrough in modal analysis capability because it allows users to extend the maximum size of models used in modal analyses well beyond the capacity of direct solver-based eigensolvers. The PCG Lanczos eigensolver works with the PCG options command (**PCGOPT**) as well as with the memory saving feature (**MSAVE**).

Both shared-memory parallel performance and distributed-memory parallel performance can be obtained by using this eigensolver. Note that in Release 11.0, Distributed ANSYS supports modal analyses and the PCG Lanczos eigensolver is the only eigensolver available that runs in a distributed fashion in this product.

Controlling PCG Lanczos Parameters

The PCG Lanczos eigensolver can be controlled using several options on the **PCGOPT** command. The first of these options is the Level of Difficulty value (*Lev_Diff*). In most cases, choosing a value of AUTO (which is the default) for *Lev_Diff* is sufficient to obtain an efficient solution time. However, in some cases you may find that manually adjusting the *Lev_Diff* value further reduces the total solution time. Setting a *Lev_Diff* value equal to 1 uses less memory compared to other *Lev_Diff* values; however, the solution time is longer in most cases. Setting higher *Lev_Diff* values (e.g., 3 or 4) can help for problems that cause the PCG solver to have some difficulty in converging. This typically occurs when elements are poorly shaped or are very elongated (that is, having high aspect ratios). A new *Lev_Diff* value of 5 is first available in version 11.0 and warrants further explanation.

A *Lev_Diff* value of 5 causes a fundamental change to the equation solver being used by the PCG Lanczos eigensolver. This *Lev_Diff* value makes the PCG Lanczos eigensolver behave more like the Block Lanczos eigensolver by replacing the PCG iterative solver with a direct solver similar to the Sparse direct solver. As with the Block Lanczos eigensolver, the numeric factorization step can either be done in an incore mode using memory or in an out-of-core mode using hard drive space. The Memory field on the **PCGOPT** command can allow the user to force one of these two modes or let ANSYS decide which mode to use. Due to the amount of computer resources needed by the direct solver, choosing a *Lev_Diff* value of 5 essentially eliminates the reduction in computer resources obtained by using the PCG Lanczos eigensolver compared to using the Block Lanczos eigensolver. Thus, this option is generally only recommended for problems that have less than one million degrees of freedom, though its efficiency is highly dependent on several factors such as the number of modes requested and I/O performance. Larger numbers of modes may increase the size of problem for which a value of 5 should be used, while slower I/O performance may decrease the size of problem for which a value of 5 should be used.

When to Choose PCG Lanczos

Determining when to choose the appropriate eigensolver can sometimes be a challenge. The Block Lanczos eigensolver is currently the recommended eigensolver for most applications, however the PCG Lanczos eigensolver can be more efficient than the Block Lanczos eigensolver for certain cases. Here are some guidelines to follow when deciding whether to use the Block Lanczos or PCG Lanczos eigensolver. Typically the following conditions must be met for the PCG Lanczos eigensolver to be most efficient.

1. The model would be a good candidate for using the PCG solver in a similar static or full transient analysis.
2. The number of requested modes is less than a hundred.
3. The beginning frequency input on the **MODOPT** command is zero (or near zero).

The PCG Lanczos eigensolver (like all iterative solvers) is most efficient when the solution converges quickly. So if the model would not converge quickly in a similar static or full transient analysis, it is expected that the PCG Lanczos eigensolver will also not converge quickly and thus be less efficient. The PCG Lanczos eigensolver is most efficient when finding the lowest natural frequencies of the system. As the number of requested modes begins to approach one hundred, or when requesting only higher modes, the Block Lanczos eigensolver becomes a better choice.

Other factors such as the size of the problem and the hardware being used can affect the eigensolver selection strategy. For example, when solving problems where the Block Lanczos eigensolver runs in an out-of-core mode on a system with very slow hard drive speed (that is, bad I/O performance) the PCG Lanczos eigensolver may be the better choice as it does significantly less I/O, assuming a *Lev_Diff* value of 1 through 4 is used. Another example would be solving a model with 15 million degrees of freedom. In this case, the Block Lanczos eigensolver would need approximately 500 GB of hard drive space. If computer resources are limited, the PCG Lanczos eigensolver may be the only viable choice for solving this problem since the PCG Lanczos eigensolver does much less I/O than the Block Lanczos eigensolver. The PCG Lanczos eigensolver has been successfully used on problems exceeding 100 million degrees of freedom.

5.2. Preprocessing and Postprocessing Memory Requirements

While optimal performance of the solution phase is obtained by minimizing the database memory (-db) and maximizing the available scratch space (-m), this is not true of the preprocessing and postprocessing phases of an analysis. For these activities, it is best to launch ANSYS with a -db large enough to hold the entire model in memory (that is, no `Jobname.PAGE` file is created). As a rule of thumb, you need approximately 30 MB for every 100,000 nodes and 40 MB for every 100,000 elements. You can also judge the required -db by the size of the database file `Jobname.DB`, adding 20% for postprocessing data.

A good practice to follow when solving models that are large for a given system (that is, models that use most of the available memory) is to separate the ANSYS or DANSYS run into preprocessing, solving, and postprocessing stages. After preprocessing is completed, the database file can be saved and a new job started by resuming the database file. For pre- and postprocessing, you should allocate additional memory to keep the database file in memory without creating a page file. For the solution phase, database space can be reduced to increase the amount of physical memory available for solver memory requirements. It is also good practice to use batch mode for large jobs so that an output file containing all of the memory usage statistics and other solution statistics is written, rather than letting all output disappear in an interactive window.

Chapter 6: Measuring ANSYS Performance

A key step in maximizing ANSYS performance on any hardware system is to properly interpret the ANSYS output. This section will describe how to use ANSYS output to measure performance for each of the commonly used solver choices in ANSYS. The performance measurements can be used to assess CPU , I/O, memory use, and performance for various hardware configurations. Armed with this knowledge, you can use the options previously discussed to improve performance the next time you run your current analysis or a similar one. Additionally, this data will help identify hardware bottlenecks that you can remedy.

The following performance topics are available:

Note

Sample program output listings shown in this chapter and elsewhere in this document use the abbreviation "Mb" to indicate megabytes.

6.1. Sparse Solver Performance Output

Performance information for the sparse solver is printed by default to the ANSYS file `Jobname.BCS`. Use the command **BCSOPTION**,,,,,,PERFORMANCE to print this same information, along with additional memory usage information, to the standard ANSYS output file. If this command is used, the information will be printed into the output file for every call to the sparse solver, not just the current call as in `Jobname.BCS`.

For jobs that call the sparse solver multiple times (nonlinear, transient, etc.), a good technique to use for studying performance output is to add the command **NCNV**,1,,*n*, where *n* specifies a fixed number of cumulative iterations. The job will run up to *n* cumulative iterations and then stop. For most nonlinear jobs, 3 to 5 calls to the sparse solver is sufficient to understand memory usage and performance for a long run. Then, the **NCNV** command can be removed, and the entire job can be run using memory settings determined from the test run.

Example 6.1, "Sparse Solver Performance Summary" shows an example of the output from the sparse solver performance summary. The times reported in this summary use CPU time and wall clock time. In general, CPU time reports only time that a processor spends on the user's application, leaving out system time and I/O wait time. When using a single processor, the CPU time is a subset of the wall time. However, when using multiple processors, some systems accumulate the CPU times from all processors, so the CPU time reported by ANSYS will exceed the wall time. Therefore, the most meaningful performance measure is wall clock time because it accurately measures total elapsed time. Wall times are reported in the second column of numbers in the sparse solver performance summary.

When comparing CPU and wall times for a single processor, if the wall time is excessively greater than the CPU time, it is usually an indication that a large amount of elapsed time was spent doing actual I/O. If this is typically the case, determining why this I/O was done (that is, looking at the memory mode and comparing the size of matrix factor to physical memory) can often have dramatic improvements on performance.

The most important performance information from the sparse solver performance summary is matrix factorization time and rate, solve time and rate, and the file I/O statistics (items marked A, B, and C in *Example 6.1, "Sparse Solver Performance Summary"*). Matrix factorization time and rate measures the performance of the computationally

intensive matrix factorization. The factorization rate provides the best single measure of peak obtainable speed for most hardware systems because it uses highly tuned math library routines that dominate the matrix factorization computations. The rate is reported in units of Mflops (millions of floating point operations per second) and is computed using an accurate count of the total number of floating point operations required for factorization (also reported in *Example 6.1, "Sparse Solver Performance Summary"*) in millions of flops, divided by the total elapsed time for the matrix factorization. While the factorization is typically dominated by a single math library routine, the total elapsed time is computed from the start of factorization until the finish. The compute rate includes all overhead (including any I/O required) for factorization. On modern hardware, the factorization rates typically observed in sparse matrix factorization range from 1000 Mflops (1 Gflop) to over 5500 Gflops on a single core. For parallel factorization, compute rates can now approach 20 Gflops using the fastest multicore processors, although rates of 6 to 10 Gflops are more common for parallel matrix factorization. Factorization rates do not always determine the fastest computer system for ANSYS runs, but they do provide a meaningful and accurate comparison of processor peak performance. I/O performance and memory size are also important factors in determining overall system performance.

Sparse solver I/O performance can be measured by the forward/backward solve required for each call to the solver; it is reported in the output in MB/sec. When the sparse solver runs incore, the effective I/O rate is really a measure of memory bandwidth, and rates of 2000 MB/sec or higher will be observed on most modern processors. When out-of-core factorization is used and the system buffer cache is large enough to contain the matrix factor file in memory, the effective I/O rate will be 500+ MB/sec. This high rate does not indicate disk speed, but rather indicates that the system is effectively using memory to cache the I/O requests to the large matrix factor file. Typical effective I/O performance for a single drive ranges from 30 to 70 MB/sec. Higher performance—over 100 MB/sec and up to 300 MB/sec—can be obtained from RAID0 drives in Windows (or multiple drive, striped disk arrays on high-end UNIX/Linux servers). With experience, a glance at the effective I/O rate will reveal whether a sparse solver analysis ran incore, out-of-core using the system buffer cache, or truly out-of-core to disk using either a single drive or a multiple drive fast RAID system.

The I/O statistics reported in the sparse solver summary list each file used by the sparse solver and shows the unit number for each file. For example, unit 20 in the I/O statistics reports the size and amount of data transferred to ANSYS file `Jobname.LN20`. The most important file used in the sparse solver is the matrix factor file, `Jobname.LN09` (see D in *Example 6.1, "Sparse Solver Performance Summary"*). In the example, the LN09 file is 1746 MB and is written once and read twice for a total of 6098 MB of data transfer. I/O to the smaller files does not usually contribute dramatically to increased wall clock time, and in most cases these files will be cached automatically by the system buffer cache. If the size of the LN09 file exceeds the physical memory of the system or is near the physical memory, it is usually best to run the sparse solver in optimal out-of-core memory mode, saving the extra memory for system buffer cache. It is best to use incore memory only when the memory required fits comfortable within the available physical memory.

This example shows a well-balanced system (high factor mflops and adequate I/O rate). Additional performance gains could be achieved by running incore or using more than one processor.

Example 6.1 Sparse Solver Performance Summary

```
     number of equations                     =         253700
     no. of nonzeroes in lower triangle of a =       18597619
     number of compressed nodes              =          74216
     no. of compressed nonzeroes in l. tri.  =        1685103
     amount of workspace currently in use    =        1904036
     max. amt. of workspace used             =       33667954
     no. of nonzeroes in the factor l        =      228832700.
     number of super nodes                   =           4668
     number of compressed subscripts         =        1748112
     size of stack storage                   =       20716944
     maximum order of a front matrix         =           5007
     maximum size of a front matrix          =       12537528
     maximum size of a front trapezoid       =         318368
     no. of floating point ops for factor    =     3.9077D+11
     no. of floating point ops for solve     =     9.1660D+08
     actual no. of nonzeroes in the factor l =      228832700.
     actual number of compressed subscripts  =        1748112
     actual size of stack storage used       =       21200111
     negative pivot monitoring activated
     number of negative pivots encountered   =             0.
     factorization panel size                =             64
     factorization update panel size         =             32
     solution block size                     =              2
     time (cpu & wall) for structure input   =       2.046875        2.150099
     time (cpu & wall) for ordering          =       5.015625        5.806626
     time (cpu & wall) for symbolic factor   =       0.093750        0.142739
     time (cpu & wall) for value input       =       8.296875       52.134713
     time (cpu & wall) for numeric factor    =     219.359375      229.913349 <---A (Factor)
     computational rate (mflops) for factor  =    1781.411726     1699.637555 <---A (Factor)
     condition number estimate               =     0.0000D+00
     time (cpu & wall) for numeric solve     =       5.203125       28.920544 <---B (Solve)
     computational rate (mflops) for solve   =     176.163229       31.693708 <---B (Solve)
     effective I/O rate (Mb/sec) for solve   =     671.181892      120.753027 <---C (I/O)

     i/o stats:    unit           file length             amount transferred
                                 words     mbytes            words     mbytes
                  ----           -----     ------            -----     ------
                    20      29709534.      227. Mb      61231022.      467. Mb
                    25       1748112.       13. Mb       6118392.       47. Mb
                     9     228832700.     1746. Mb     798083814.     6089. Mb <---D (File)
                    11      37195238.      284. Mb     111588017.      851. Mb

                -------     ----------     --------      ----------    --------
                Totals:     297485584.     2270. Mb     977021245.     7454. Mb

  Sparse Solver Call     1 Memory   ( Mb) =        256.9
  Sparse Matrix Solver     CPU Time (sec) =        240.406
  Sparse Matrix Solver ELAPSED Time (sec) =        320.689
```

6.2. Distributed Sparse Solver Performance Output

DSPARSE solver performance is reported in the ANSYS output file only for the first call to the solver. An example of the output is shown in *Example 6.2, "Distributed Sparse Solver Performance Summary for 2 Processes"*.

Example 6.2 Distributed Sparse Solver Performance Summary for 2 Processes

```
DISTRIBUTED SPARSE MATRIX DIRECT SOLVER.
NUMBER OF EQUATIONS  =     3133395
-------------------------------------------------------------
---------DISTRIBUTED SPARSE SOLVER RUN STATS------------------
-------------------------------------------------------------

Memory allocated on processor   0        =  1479.839 MB
Memory allocated on processor   1        =  1477.896 MB
Total Memory allocated by all processors =  2957.736 MB

Time(cpu & wall) for input matrix indices          2.30        2.30
Time(cpu & wall) for reordering & sym.factor      131.05      131.14
Time(cpu & wall) for input matrix values            9.91        9.91
Time(cpu & wall) for matrix redistribution        105.97      105.99
Time(cpu & wall) for matrix factorization         436.07      477.56 <---A (Factor)
Computational rate (mflops) for factor           3498.71     3194.71 <---A (Factor)
Time(cpu & wall) for matrix forwd/back solve       43.12      442.19 <---B (Solve)
Computational rate (mflops) for solve             150.11       14.64 <---B (Solve)
Effective I/O rate (Mb/sec) for solve             571.93       55.77 <---C (I/O)
Time(cpu & wall) for D-sparse total solution      728.42     1169.09
```

The memory information shows the amount of memory used on each process for matrix factorization and solves. There is additional memory required on the master process for solver preprocessing that is not currently reported. Just as for the SMP sparse solver, the factorization times (A) and solve times (B) are reported separately. The additional times reported are for solver preprocessing of the input matrix and are generally not parallelized. In this example, the factorization time is almost equal to the I/O dominated solve time. This output shows a very modest 55 MB/sec effective I/O rate (C). In this example both solver preprocessing and I/O performance significantly reduce the overall benefit of parallel processing. This example shows poor I/O system throughout. Running incore or using a RAID0 configuration could increase the performance.

6.3. Block Lanczos Solver Performance Output

Block Lanczos performance is reported in a similar manner to the SMP sparse solver. Use **BCSOPTION**,,,,,,PERFORMANCE command to add the performance summary to the ANSYS output file. The Block Lanczos method uses an assembled stiffness and mass matrix in addition to factoring a matrix that is a combination of the mass and stiffness matrices computed at various shift points. The memory usage heuristics for this algorithm must divide available memory for several competing demands. Various parts of the computations can be incore or out-of-core depending on the memory available at the start of the Lanczos driver. The compute kernels for Block Lanczos include matrix factorization, matrix vector multiplication using the mass matrix, multiple block solves, and some additional block vector computations. Matrix factorization is the most critical compute kernel for CPU performance, while the block solves are the most critical operations for I/O performance.

A key factor determining the performance of Block Lanczos is the block size actually used in a given run. Maximum performance for Block Lanczos in ANSYS is usually obtained when the block size is 8. This means that each block solve requiring a forward and backward read of the factored matrix completes 8 simultaneous solves. If the memory available for a given run is less than the amount required for recommended optimal out-of-core, the block size is reduced. Reducing the block size requires the algorithm to increase the number of block solves, which can greatly increase I/O. Each block solve requires a forward and backward read of the matrix factor file, `Jobname.LN07`.

The next two examples contain parts of the Block Lanczos performance summary. *Example 6.3, "Memory Information from Block Lanczos Performance Summary"* shows the memory usage information from a 500k DOF Block Lanczos run. The initial memory allocation of 823 MB of memory (A) is large enough to run this example using the recommended memory setting for optimal out-of-core (B). If a smaller initial memory allocation was used, the mass matrix could be stored out-of-core and/or the block size would be reduced. *Example 6.4, "Block Lanczos CPU and I/O Performance Summary"* gives details on the costs of various steps of the Lanczos algorithm. The im-

portant parts of this summary for measuring hardware performance are the times and rates for factorization (C) and solves (D) computed using CPU and wall times. Wall times are the most useful because they include all system overhead. In this parallel run, the factorization performance is measured at 6691 Mflops, while the solve rate was over 1000 Mflops and the matrix multiply was over 500 Mflops. These numbers indicate a balanced HPC system, well configured for this size problem.

Other statistics from *Example 6.4, "Block Lanczos CPU and I/O Performance Summary"* that are useful for comparing Lanczos performance are the number of factorizations (E), Lanczos steps (F), and block solves (G). The Block Lanczos algorithm in ANSYS always does at least 2 factorizations for robustness and to guarantee that the computed eigenvalues do not skip any values. For larger numbers of modes or for models with large holes in the spectrum of eigenvalues, the algorithm may require additional factorizations. Additionally, if users specify a range of frequencies using the **MODOPT** command, additional matrix factorizations will always be required. In most cases, we do not recommend that you specify a frequency range if the desired result is the first group of modes.

The final part of *Example 6.4, "Block Lanczos CPU and I/O Performance Summary"* shows the files used by the Block Lanczos run. The largest file is unit 7 (`Jobname.LN07`), the matrix factor file (H). If you specify the end points of the frequency interval, additional files will be written which contain copies of the matrix factor files computed at the 2 shift points. These copies of the matrix factors will be stored in units 20 and 22. This will significantly increase the disk storage requirement for Block Lanczos as well as add additional I/O time to write these files. For large models that compute 100 or more modes, it is common to see I/O statistics that show over 1 TB (1 Million MB) of I/O. In this example, the LN07 file is over 7 GB in size, and a total of 142 GB of I/O to this file was required (H).

Example 6.3 Memory Information from Block Lanczos Performance Summary

```
BLOCK LANCZOS CALCULATION OF UP TO   10 EIGENVECTORS.
  NUMBER OF EQUATIONS       =      525867
  MAXIMUM WAVEFRONT         =         195
  MAXIMUM MODES STORED      =          10
  MINIMUM EIGENVALUE        =  0.0000
  MAXIMUM EIGENVALUE        = 0.10000E+09
  EST. OF MEMORY NEEDED             =  823.30     (MB)
  MEMORY AVAILABLE FOR EIGENSOLVER =  823.30     (MB) <---A
  Incore factorization
            input matrix    63898547   487.51
           matrix factor  921474654.  7030.29
               max stack   121492395   926.91
       panel and update      1011744     7.72
                    misc      4542898    34.66
 ------------------------------------------
            Total incore  991716308.  7566.19

  Out-of-core factorization
            input matrix    63898547   487.51
               max stack   121492395   926.91
           largest front    55540530   423.74
       panel and update      1805814    13.78
                    misc      4537544    34.62
 ------------------------------------------
                Total OOC   48942389.   373.40
       Total OOC + stack  170434784.  1300.31
       Total OOC + front  104482919.   797.14
     OOC + front + stack  225975314.  1724.05

MEMORY TO PERFORM EIGENEXTRACTION: using opt Min core
   MIN. TO COMPLETE VALUE INPUT            =     32049382     244.52     (MB)
   MIN. FOR PART IN-CORE AND OUT-OF-CORE   =     90757370     692.42     (MB)
   MIN. FOR IN-CORE                        =   1247265347     9515.9     (MB)
   RECOM. FOR PART IN-CORE AND OUT-OF-CORE =    106851780     815.21     (MB) <---B
   RECOM. FOR IN-CORE                      =   1250946416     9544.0     (MB)

  Factorization is out-of-core
  Stiffness matrix is out-of-core
  Mass matrix is incore

  This run uses recommended optimal memory for out-of-core mode
```

Example 6.4 Block Lanczos CPU and I/O Performance Summary

```
 = sparse eigenanalysis statistics =
  ===================================

      number of equations                    =         525867
      no. of flt. pt. ops for a single factor =    4.163879D+12
      no. of flt. pt. ops for a block solve   =    3.688528D+08
      no. of flt. pt. ops for block mtx mult. =    4.529874D+08
      total no. of flt. pt. ops for factor    =    8.327757D+12
      total no. of flt. pt. ops for solve     =    2.655740D+11
      total no. of flt. pt. ops for mtx mult. =    9.512734D+09
      actual no. of nonzeroes in the factor l =       921474654.
      actual number of compressed subscripts  =         3772137
      actual size of stack storage used       =        68735964
      pivot tolerance used for factorization  =        0.000000
      factorization panel size                =              64
      factorization update panel size         =              32
      solution block size                     =               8
      total number of factorizations          =               2 <---E
      total number of lanczos runs            =               1
      total number of lanczos steps           =               8 <---F
      total number of block solves            =               9 <---G
      time (cpu & wall) for structure input   =       12.594727       12.587932
      time (cpu & wall) for ordering          =       22.123047       22.112111
      time (cpu & wall) for symbolic factor   =        0.382812        0.382070
      time (cpu & wall) for value input       =       28.284180       51.806799
      time (cpu & wall) for numeric factor    =     2364.491211     1244.471642 <---C (Factor)
      computational rate (mflops) for factor  =     3522.008044     6691.801391 <---C (Factor)
      time (cpu & wall) for numeric solve     =      156.026367      260.662024 <---D (Solve)
      computational rate (mflops) for solve   =     1702.109825     1018.844283 <---D (Solve)
      time (cpu & wall) for matrix multiply   =       17.574219       17.564429
      computational rate (mflops) for mult.   =      541.289176      541.590876

      cost for sparse eigenanalysis
      -----------------------------
      lanczos run start up cost               =       20.955078
      lanczos run recurrence cost             =      138.989258
      lanczos run reorthogonalization cost    =       27.198242
      lanczos run internal eigenanalysis cost =        0.002930
      lanczos eigenvector computation cost    =        3.216797
      lanczos run overhead cost               =        0.083008

      total lanczos run cost                  =      190.445312
      total factorization cost                =     2364.491211
      shift strategy and overhead  cost       =        2.619141

      total sparse eigenanalysis cost         =     2557.555664

      i/o stats:    unit           file length              amount transferred
                                  words     mbytes            words     mbytes
                   ----          -----     ------           -----     ------
                     20      85593538.      653. Mb     256780614.     1959. Mb
                     21       3772138.       29. Mb      18860690.      144. Mb
                     22      28051940.      214. Mb      56103880.      428. Mb
                     25      37862424.      289. Mb     159863568.     1220. Mb
                     28      33655488.      257. Mb      84138720.      642. Mb
                      7     921474654.     7030. Mb   18672861468.   142463. Mb <---H (File)
                      9      83128215.      634. Mb    1274336942.     9722. Mb
                     11       5784537.       44. Mb       5784537.       44. Mb

                   Total:    1199322934.     9150. Mb   20528730419.   156622. Mb
```

6.4. PCG Solver Performance Output

The PCG solver performance summary information is not written to the ANSYS output files. Instead, a separate file named `Jobname.PCS` is written when the PCG solver is used. This file contains useful information about the computational costs of the iterative PCG solver. Iterative solver computations typically involve sparse matrix

operations rather than the dense block kernels that dominate the sparse solver factorizations. Thus, for the iterative solver, performance metrics reflect measures of memory bandwidth rather than peak processor speeds. The information in `Jobname.PCS` is also useful for identifying which preconditioner option was chosen for a given simulation and allows users to try other options to eliminate performance bottlenecks. *Example 6.5, "Jobname.PCS Output File"* shows a typical `Jobname.PCS` file.

The memory information in *Example 6.5, "Jobname.PCS Output File"* shows that this 500k DOF PCG solver run requires less than 200 MB of memory (A). This model uses SOLID95 elements which, by default, use the **MSAVE**,ON feature in ANSYS. This feature uses an "implicit" matrix-vector multiplication algorithm that avoids using a large "explicitly" assembled stiffness matrix. (See the **MSAVE** command description for more information.) The PCS file reports the number of elements assembled and the number that use the memory-saving option (B).

The PCS file also reports the number of iterations (C) and level of difficulty (D). The level of difficulty is automatically set, but can be user-controlled by the *Lev_Diff* option on the **PCGOPT** command. As the value of *Lev_Diff* increases, more expensive preconditioner options are used that often increase memory requirements and computations. However, increasing *Lev_Diff* also reduces the number of iterations required to reach convergence for the given tolerance.

As a rule of thumb, when using the default tolerance of 1.0e-8 and a level of difficulty of 1 (*Lev_Diff* = 1), a static or full transient analysis with the PCG solver that requires more than 2000 iterations per equilibrium iteration probably reflects an inefficient use of the iterative solver. In this scenario, raising the level of difficulty to bring the number of iterations closer to the 300-750 range will usually result in the most efficient solution. If increasing the level of difficulty does not significantly drop the number of iterations, then it is probably not an efficient option, and the matrix could possibly require the use of the sparse direct solver for an efficient solution time.

The CPU performance reported in the PCS file is divided into (E) matrix multiplication using the stiffness matrix and (F) the various compute kernels of the preconditioner. For parallel processing, the preconditioner step is not as scalable when run in parallel as the matrix multiplication step; thus, higher levels of difficulty have less speedup for parallel runs. It is normal that the Mflop rates reported in the PCS file are a lot lower than matrix factorization kernels, but they provide a good measure to compare relative performance of memory bandwidth on different hardware systems.

The I/O reported (G) in the PCS file is much less than that required for matrix factorization in the sparse solver. This I/O occurs only during solver preprocessing before the iterative solution and is generally not a performance factor in the PCG solver. The one exception to this rule is when the *Lev_Diff* = 5 option on the **PCGOPT** command is specified, and the factored matrix used for this preconditioner is out-of-core. Normally, this option is only used for the iterative LANPCG modal solver and only for smaller problems (under 1 million DOFs) where the factored permutations fit in memory.

Example 6.5 Jobname.PCS Output File

```
Number of Processors: 1
 Degrees of Freedom: 532491
 DOF Constraints: 8832
 Elements: 40297  <---B
  Assembled: 0 <---B
  Implicit: 40297 <---B (MSAVE,ON)
 Nodes: 177497
 Number of Load Cases: 1

 Nonzeros in Upper Triangular part of
             Global Stiffness Matrix : 0
 Nonzeros in Preconditioner: 9984900
  *** Precond Reorder: MLD ***
  Nonzeros in V: 6644052
  Nonzeros in factor: 2275866
  Equations in factor: 9684
 *** Level of Difficulty: 1   (internal 0) *** <---D (Preconditioner)

 Total Operation Count: 3.90871e+011
 Total Iterations In PCG: 613  <---C (Convergence)
 Average Iterations Per Load Case:  613.0
 Input PCG Error Tolerance: 1e-006
 Achieved PCG Error Tolerance: 9.89056e-007


 DETAILS OF PCG SOLVER SETUP TIME(secs)        Cpu        Wall
      Gather Finite Element Data              0.75        0.85
      Element Matrix Assembly                 3.66        6.10

 DETAILS OF PCG SOLVER SOLUTION TIME(secs)     Cpu        Wall
      Preconditioner Construction             5.67       13.57
      Preconditioner Factoring                2.19        2.51
      Apply Boundary Conditions               0.30        0.44
      Preconditioned CG Iterations          464.13      466.98
           Multiply With A                  410.39      411.50 <---E (Matrix Mult. Time)
                Multiply With A22           410.39      411.50
           Solve With Precond                43.64       43.76 <---F (Preconditioner Time)
                Solve With Bd                 8.41        8.48
                Multiply With V              27.03       27.04
                Direct Solve                  6.25        6.27
*******************************************************************************
      TOTAL PCG SOLVER SOLUTION CP TIME      =   473.00 secs
      TOTAL PCG SOLVER SOLUTION ELAPSED TIME =   487.45 secs
*******************************************************************************
 Total Memory Usage at CG         :   189.74 MB
 PCG Memory Usage at CG           :   124.60 MB
 Peak Memory Usage                :   189.74 MB (at symPCCG) <---A (Memory)
 Memory Usage for MSAVE Data      :    47.16 MB
 *** Memory Saving Mode Activated : Jacobians Precomputed ***
*******************************************************************************
 Multiply with A MFLOP Rate       :   875.93 MFlops
 Solve With Precond MFLOP Rate    :   586.05 MFlops
 Precond Factoring MFLOP Rate     :   416.23 MFlops
*******************************************************************************
 Total amount of I/O read         :   400.52 MB <---G
 Total amount of I/O written      :   290.37 MB <---G
*******************************************************************************
```

6.5. PCG Lanczos Solver Performance Output

The PCG Lanczos eigensolver uses the PCG iterative solver with the Lanczos algorithm to compute eigenvalues (frequencies) of linear systems. The performance summary for PCG Lanczos is contained in the file `Jobname.PCS`, with additional information related to the Lanczos solver. The important details in the this file are the number of load cases (A), total iterations in PCG (B), level of difficulty (C), and the total elapsed time (D).

The PCG Lanczos solver uses the Lanczos algorithm to compute eigenvalues and eigenvectors for modal analyses, but replaces matrix factorization and multiple solves with multiple iterative solves. An iterative solution is faster than matrix factorization, but usually takes longer than a single block solve. Each successive increase in level of difficulty (`Lev_Diff` value on **PCGOPT** command) increases the cost per iteration, but also reduces the total iterations required. For `Lev_Diff` = 5, a direct matrix factorization of the assembled shifted stiffness matrix is used so that the number of total iterations is the same as the number of load cases. This option is best for smaller problems where the memory required for factoring the assembled shifted stiffness matrix is available, and the cost of factorization is not dominant.

The real power of the PCG Lanczos method is experienced for very large models, usually above 1 million DOFs, where matrix factorization and solves become very expensive. The PCG Lanczos method will choose a good default level of difficulty, but experienced users may improve solution time by changing the level of difficulty via the **PCGOPT** command.

In the PCS file the number of load cases corresponds to the number of Lanczos steps required to obtain the specified number of eigenvalues. The number of load cases is usually 2 to 3 times more than the number of eigenvalues desired, unless the Lanczos algorithm has difficulty converging. PCG Lanczos will be increasingly expensive relative to Block Lanczos as the number of desired eigenvalues increases. PCG Lanczos is best for obtaining a few modes (up to 100) for large models (over 1 million DOF).

The next two examples show parts of the PCS file that report performance statistics described above for a 500k DOF modal analysis that computes 10 modes. The difference between the two runs is the level of difficulty used (`Lev_Diff` on the **PCGOPT** command). *Example 6.6, "PCS File for PCG Lanczos, Level of Difficulty = 3"* uses **PCGOPT**,3. The output shows that `Lev_Diff` = 3 (C), and the total iterations required for 30 Lanczos steps (A) is 3688 (B), or 122.9 iterations per step (E). *Example 6.7, "PCS File for PCG Lanczos, Level of Difficulty = 5"* shows that increasing `Lev_Diff` to 5 (C) on **PCGOPT** reduces the iterations required per Lanczos to just one (E).

Though the solution time difference in these examples is insignificant (see (D) in both examples), `Lev_Diff` = 5 can be much faster for more difficult models where the average number of iterations per load case is much higher. The average number of PCG iterations per load case for efficient PCG Lanczos solutions is generally around 100 to 200. If the number of PCG iterations per load case begins to exceed 500, then either the level of difficulty should be increased in order to find a more efficient solution, or it may be more efficient to use the Block Lanczos eigensolver (assuming the problem size does not exceed the limits of the system).

Example 6.6 PCS File for PCG Lanczos, Level of Difficulty = 3

```
 Number of Processors: 2
 Degrees of Freedom: 532491
 DOF Constraints: 6624
 Elements: 40297
  Assembled: 40297
  Implicit: 0
 Nodes: 177497
 Number of Load Cases: 30 <---A (Lanczos Steps)

 Nonzeros in Upper Triangular part of
             Global Stiffness Matrix : 43701015
 Nonzeros in Preconditioner: 72867697
  *** Precond Reorder: MLD ***
  Nonzeros in V: 3194385
  Nonzeros in factor: 68608330
  Equations in factor: 53333
 *** Level of Difficulty: 3   (internal 2) *** <---C (Preconditioner)

 Total Operation Count: 1.90773e+12
 Total Iterations In PCG: 3688   <---B (Convergence)
 Average Iterations Per Load Case:  122.9 <---E (Iterations per Step)
 Input PCG Error Tolerance: 0.0001
 Achieved PCG Error Tolerance: 9.96657e-05

*****************************************************************************
      TOTAL PCG SOLVER SOLUTION CP TIME      =  3366.23 secs
      TOTAL PCG SOLVER SOLUTION ELAPSED TIME =  1877.77 secs <---D (Total Time)
*****************************************************************************
 Total Memory Usage at Lanczos    :  1073.20 MB
 PCG Memory Usage at Lanczos      :  1002.48 MB
 Peak Memory Usage                :  1214.22 MB (at computeShift)
 Memory Usage for Matrix          :   419.72 MB
*****************************************************************************
 Multiply with A MFLOP Rate       :  1046.56 MFlops
 Solve With Precond MFLOP Rate    :  1019.91 MFlops
 Precond Factoring MFLOP Rate     :  3043.02 MFlops
*****************************************************************************
 Total amount of I/O read         :  1453.77 MB
 Total amount of I/O written      :  1363.37 MB
*****************************************************************************
```


Example 6.7 PCS File for PCG Lanczos, Level of Difficulty = 5

```
Number of Processors: 2
 Degrees of Freedom: 532491
 DOF Constraints: 6624
 Elements: 40297
  Assembled: 40297
  Implicit: 0
 Nodes: 177497
 Number of Load Cases: 30 <---A (Lanczos Steps)

 Nonzeros in Upper Triangular part of
             Global Stiffness Matrix : 43701015
 Nonzeros in Preconditioner: 949238571
  *** Precond Reorder: MLD ***
  Nonzeros in V: 0
  Nonzeros in factor: 949238571
  Equations in factor: 532491
 *** Level of Difficulty: 5   (internal 0) *** <---C (Preconditioner)

 Total Operation Count: 4.73642e+12
 Total Iterations In PCG: 30   <---B (Convergence)
 Average Iterations Per Load Case:    1.0 <---E (Iterations per Step)
 Input PCG Error Tolerance: 0.0001
 Achieved PCG Error Tolerance: 1e-10
*************************************************************************
      TOTAL PCG SOLVER SOLUTION CP TIME      =  1799.21 secs
      TOTAL PCG SOLVER SOLUTION ELAPSED TIME =  1709.58 secs <---D (Total Time)
*************************************************************************
 Total Memory Usage at Lanczos    :   901.26 MB
 PCG Memory Usage at Lanczos      :   830.55 MB
 Peak Memory Usage                :  1554.84 MB (at formIVTV)
 Memory Usage for Matrix          :   423.00 MB
 *** Precond Factoring Out-of-core activated ***
 *** Precond Solve Out-of-core activated ***
*************************************************************************
 Solve With Precond MFLOP Rate    :   424.13 MFlops
 Precond Factoring MFLOP Rate     :  3569.16 MFlops
 Factor I/O Rate                  :   396.64 MB/sec
 Precond Solve I/O Rate           :  1533.35 MB/sec
*************************************************************************
 Total amount of I/O read         : 224928.24 MB
 Total amount of I/O written      :  7074.71 MB
*************************************************************************
```

6.6. Identifying CPU, I/O, and Memory Performance

Table A.9,"Obtaining Performance Statistics from ANSYS Solvers" summarizes the information in the previous sections for the most commonly used ANSYS solver choices. CPU and I/O performance are best measured using sparse solver statistics. Memory information from the sparse solver and file size information are important because they set boundaries indicating which problems can run efficiently incore and which problems should use optimal out-of-core memory.

The expected results summarized in *Table A.9,"Obtaining Performance Statistics from ANSYS Solvers"* are for current systems. Continual improvements in processor performance are expected, although single processor rates have recently plateaued due to power requirements and heating concerns. I/O performance is also expected to improve as wider use of inexpensive RAID0 configurations is anticipated. The sparse solver effective I/O rate statistic can determine whether a given system is getting incore performance, in-memory buffer cache performance, RAID0 speed, or is limited by single disk speed.

PCG solver statistics are mostly used to tune preconditioner options, but they also measure memory bandwidth in an indirect manner by comparing times for the compute portions of the PCG solver. The computations in the iterative solver place demand on memory bandwidth, thus comparing performance of the iterative solver is a good way to compare the effect of memory bandwidth on processor performance for different systems.

Chapter 7: Examples and Guidelines

This chapter presents several examples to illustrate how different ANSYS solvers and analysis types can be tuned to maximize performance. The chapter is broken into two main sections:

7.1. ANSYS Examples
7.2. Distributed ANSYS Examples

You can find a complete set of ANSYS and Distributed ANSYS benchmark examples at:

ftp://ftp.ansys.com/pub/devsvcs/benchmarks/bench110/

Along with the benchmark problems, you will find a summary document that explains how to run the examples and interpret the results. You may find these examples useful when evaluating hardware performance.

7.1. ANSYS Examples

The following topics are discussed in this section:

7.1.1. SMP Sparse Solver Static Analysis Example
7.1.2. Block Lanczos Modal Analysis Example
7.1.3. Summary of Lanczos Performance and Guidelines

7.1.1. SMP Sparse Solver Static Analysis Example

The first usage example is a simple static analysis using a parameterized model that can be easily modified to demonstrate the performance of ANSYS solvers for models of different size. It is a basic model that does not include contact, multiple elements types, or constraint equations, but it is effective for measuring system performance. The model is a wing-shaped geometry filled with SOLID95 elements. Two model sizes are used to demonstrate expected performance for incore and out-of-core HPC systems. This model also demonstrates expected performance from the recommended HPC system configuration for a desktop system. This system runs 64-bit Windows, has Intel 5160 Xeon dual core processors, 8 GB of memory and a RAID0 configured disk system using four 73 GB SAS drives. The Windows 64-bit runs are also compared with a Windows 32-bit system.

Two model sizes are used for the sparse solver example. Each run solves 3 load steps. The first load step includes matrix factorization; the next two load steps require only the forward/backward solve steps. The first model size has 250k DOFs. *Example 7.1, "SMP Sparse Solver Statistics Comparing Windows 32-bit and Windows 64-bit"* shows the performance statistics for this model on both Windows 32-bit and Windows 64-bit systems. The 32-bit Windows I/O statistics in this example show that the matrix factor file is 1746 MB (A), and the total file storage for the sparse solver in this run is 2270 MB. This problem does not run incore on a Windows 32-bit system, but easily runs incore on the recommended 8 GB Windows 64-bit HPC system. The CPU performance is doubled on the Windows 64-bit system, (from 1918 Mflops (B) to 4416 Mflops (C)) reflecting the performance of the latest Intel Xeon core micro architecture processors. These processors can achieve 4 flops per clock cycle, compared to the previous generation Xeon processors that achieve only 2 flops per clock cycle. Effective I/O performance on the Windows 32-bit system is 48 MB/sec for the solves (D), reflecting typical single disk drive performance. The Windows 64-bit system delivers 2818 MB/sec (E), reflecting the speed of incore solves—a factor of over 50X speedup compared 32-bit Windows! The overall time for the sparse solver on the Windows 64-bit system is one third what it was on the Windows 32-bit system (F).

Example 7.1 SMP Sparse Solver Statistics Comparing Windows 32-bit and Windows 64-bit

250k DOF Model Run on Windows 32-bit and Windows 64-bit

Windows 32-bit system, Out-of-core run, 2 GB memory, single disk drive

```
     time (cpu & wall) for numeric factor    =      168.671875      203.687919
     computational rate (mflops) for factor  =     2316.742865     1918.470986 <---B (Factor)
     time (cpu & wall) for numeric solve     =        1.984375       72.425809
     computational rate (mflops) for solve   =      461.908309       12.655700
     effective I/O rate (Mb/sec) for solve   =     1759.870630       48.218216 <---D (I/O)

     i/o stats:    unit           file length              amount transferred
                                 words     mbytes            words     mbytes
                 ----           -----     ------            -----     ------
                   20       29709534.      227. Mb       61231022.      467. Mb
                   25        1748112.       13. Mb        6118392.       47. Mb
                    9      228832700.     1746. Mb      817353524.     6236. Mb <---A (File)
                   11       37195238.      284. Mb      111588017.      851. Mb

               -------     ----------     --------      ----------    --------
               Totals:     297485584.     2270. Mb      996290955.     7601. Mb

  Sparse Solver Call     1 Memory   ( Mb) =        205.1
  Sparse Matrix Solver     CPU Time (sec) =        189.844
  Sparse Matrix Solver ELAPSED Time (sec) =        343.239 <---F (Total Time)
```

Windows 64-bit system, incore run, 8 GB memory, 3 Ghz Intel 5160 processors

```
     time (cpu & wall) for numeric factor    =       87.312500       88.483767
     computational rate (mflops) for factor  =     4475.525993     4416.283082 <---C (Factor)
     condition number estimate               =      0.0000D+00
     time (cpu & wall) for numeric solve     =        1.218750        1.239158
     computational rate (mflops) for solve   =      752.081477      739.695010
     effective I/O rate (Mb/sec) for solve   =     2865.430384     2818.237946 <---E ((I/O)

     i/o stats:    unit           file length              amount transferred
                                 words     mbytes            words     mbytes
                 ----           -----     ------            -----     ------
                   20        1811954.       14. Mb        5435862.       41. Mb
                   25        1748112.       13. Mb        4370280.       33. Mb

               -------     ----------     --------      ----------    --------
               Totals:       3560066.       27. Mb        9806142.       75. Mb

  Sparse Solver Call     1 Memory   ( Mb) =       2152.2
  Sparse Matrix Solver     CPU Time (sec) =         96.250
  Sparse Matrix Solver ELAPSED Time (sec) =         99.508 <---F (Total Time)
```

Example 7.2, "SMP Sparse Solver Statistics Comparing Incore vs. Out-of-Core" shows an incore versus out-of-core run on the same Windows 64-bit system. Each run uses 2 cores on the machine. The larger model, 750k dofs, generates a matrix factor file that is nearly 12 GB. Both incore and out-of-core runs sustain high compute rates for factorization—nearly 7 Gflops (A) for the smaller 250k DOF model and almost 7.5 Gflops (B) for the larger model. The incore run solve time (C) is only 1.26 seconds, compared to a factorization time of almost 60 seconds. The larger model, which cannot run incore on this system, still achieves a very impressive effective I/O rate of almost 300 MB/sec (D). Single disk drive configurations usually would obtain 30 to 70 MB/sec for the solves. Without the RAID0 I/O for the second model, the solve time in this example would be 5 to 10 times longer and would approach half of the factorization time. Poor I/O performance would significantly reduce the benefit of parallel processing speedup in the factorization.

This sparse solver example shows how to use the output from the **BCSOPTION**,,,,,,PERFORMANCE command to compare system performance. It provides a reliable, yet simple test of system performance and is a very good starting point for performance tuning of ANSYS. Parallel performance for matrix factorization should be evident

using 2 and 4 cores. However, there is a diminishing effect on solution time because the preprocessing times and the I/O and solves are not parallel; only the matrix factorization is parallel in the SMP sparse solver. The factorization time for models of a few hundred thousand equations has now become just a few minutes or even less. Still, for larger models (particularly dense 3-D geometries using high order solid elements) the factorization time can be hours, and parallel speedup for these models is significant. For smaller models, running incore when possible minimizes the nonparallel overhead in the sparse solver. For very large models, optimal out-of-core I/O is often more effective than using all available system memory for an incore run. Only models that run "comfortably" within the available physical memory should run incore.

Example 7.2 SMP Sparse Solver Statistics Comparing Incore vs. Out-of-Core

250k/750k DOF Models Run on Windows 64-bit System,

Windows 64-bit system, 250k DOFs, using 2 cores and running incore

```
     time (cpu & wall) for numeric factor    =       94.125000       57.503842
     computational rate (mflops) for factor  =     4151.600141     6795.534887 <---A (Factor)
     condition number estimate               =      0.0000D+00
     time (cpu & wall) for numeric solve     =        1.218750        1.260377 <---C (Solve Time)
     computational rate (mflops) for solve   =      752.081477      727.242344
     effective I/O rate (Mb/sec) for solve   =     2865.430384     2770.793287

     i/o stats:    unit           file length              amount transferred
                                 words     mbytes            words     mbytes
                   ----          -----     ------            -----     ------
                     20       1811954.       14. Mb       5435862.       41. Mb
                     25       1748112.       13. Mb       4370280.       33. Mb

                -------     ----------     --------      ----------    --------
                Totals:       3560066.       27. Mb       9806142.       75. Mb

  Sparse Solver Call     1 Memory   ( Mb) =       2152.2
  Sparse Matrix Solver     CPU Time (sec) =        103.031
  Sparse Matrix Solver ELAPSED Time (sec) =         69.217
```

Windows 64-bit system, 750K DOFs, using 2 cores running out-of-core

```
     time (cpu & wall) for numeric factor    =     1698.687500      999.705361
     computational rate (mflops) for factor  =     4364.477853     7416.069039 <---B (Factor)
     condition number estimate               =      0.0000D+00
     time (cpu & wall) for numeric solve     =        9.906250       74.601405
     computational rate (mflops) for solve   =      597.546004       79.347569
     effective I/O rate (Mb/sec) for solve   =     2276.650242      302.314232 <---D (I/O)

     i/o stats:    unit           file length              amount transferred
                                 words     mbytes            words     mbytes
                   ----          -----     ------            -----     ------
                     20      97796575.      746. Mb     202290607.     1543. Mb
                     25       5201676.       40. Mb      18205866.      139. Mb
                      9    1478882356.    11283. Mb    4679572088.    35702. Mb
                     11     121462510.      927. Mb     242929491.     1853. Mb

                -------     ----------     --------      ----------    --------
                Totals:    1703343117.    12995. Mb    5142998052.    39238. Mb

  Sparse Solver Call     1 Memory   ( Mb) =       1223.6
  Sparse Matrix Solver     CPU Time (sec) =       1743.109
  Sparse Matrix Solver ELAPSED Time (sec) =       1133.233
```

7.1.2. Block Lanczos Modal Analysis Example

Modal analyses in ANSYS using the Block Lanczos algorithm share the same sparse solver technology used in the previous example. However, the Block Lanczos algorithm for modal analyses includes additional compute kernels using blocks of vectors, sparse matrix multiplication using the assembled mass matrix, and repeated forward/backward solves using multiple right-hand side vectors. The memory requirement for optimal performance

balances the amount of memory for matrix factorization, storage of the mass and stiffness matrices, and the block vectors. The fastest performance for Block Lanczos occurs when the matrix factorizations and block solves can be done incore, *and* when the memory allocated for the Lanczos solver is large enough so that the block size used is not reduced. Very large memory systems which can either contain the entire matrix factor in memory or cache all of the files used for Block Lanczos in memory show a significant performance advantage.

Understanding how the memory is divided up for Block Lanczos runs can help users improve solution time significantly in some cases. The following examples illustrate some of the steps for tuning memory use for optimal performance in modal analyses.

The example considered next has 500k DOFs and computes 40 modes. *Example 7.3, "Windows 32-bit System Using Default Memory"* contains output from the **BCSOPTION**,,,,,,PERFORMANCE command. The results in this example are from a desktop Windows 32-bit system with 4 GB of memory, using default memory settings. The default memory for Windows 32-bit systems at Release 11.0 is -m 512 -db 256. The Block Lanczos default memory strategy for Windows 32-bit systems tries to run within the available memory. This memory strategy allows users to solve very large problems on a desktop system, but with less than optimal performance. In this example, the output file shows that the Block Lanczos run starts with an initial memory allocation of just 191 MB (A). Eventually, after preprocessing the input matrices, the initial memory allocation is raised to 338 MB (B). This amount is just above the minimum allowed for out-of-core solution in Block Lanczos.

The performance summary in *Example 7.3, "Windows 32-bit System Using Default Memory"* shows that there are 2 factorizations (C), the minimum for Block Lanczos, but there are 26 block solves (D). The number of block solves is directly related to the cost of the solves which exceeds factorization time by almost 3 times (5609 seconds (E) vs 1913 seconds (F)). The very low computational rate for the solves (46 Mflops (G)) also reveals the I/O imbalance in this run. For out-of-core Lanczos runs in ANSYS, if there is less memory available than the recommended memory size, the block size is automatically reduced to reduce memory requirements. This will always increase the number of solves and I/O cost. ANSYS will also try to keep the mass matrix in memory to avoid I/O during the mass matrix multiply operations, but in this case all matrices were out-of-core. The best solution for this performance problem is to move to a Windows 64-bit system with 8 GB of memory. However, the performance on the Windows 32-bit system can be significantly improved in this case simply by increasing the initial ANSYS memory allocation. While users cannot directly control the blocksize used in the Lanczos solver, they can provide additional memory to the Lanczos solver thereby reducing the possibility of the solver running with a smaller than optimal blocksize.

Example 7.3 Windows 32-bit System Using Default Memory

500k DOFs Block Lanczos Run Computing 40 Modes

```
 MEMORY AVAILABLE FOR EIGENSOLVER =  191.37     (MB) <---A (Initial Memory)
   MEMORY TO PERFORM EIGENEXTRACTION: using Min core
    MIN. TO COMPLETE VALUE INPUT            =      6095557     46.505     (MB)
    MIN. FOR PART IN-CORE AND OUT-OF-CORE   =     40228399     306.92     (MB)
    MIN. FOR IN-CORE                        =    749364879     5717.2     (MB)
    RECOM. FOR PART IN-CORE AND OUT-OF-CORE =     58223834     444.21     (MB)
    RECOM. FOR IN-CORE                      =    752950566     5744.6     (MB)

   Factorization is out-of-core  <---B
   Stiffness matrix is out-of-core <---B
   Mass matrix is out-of-core  <---B

  Memory available for Block Lanczos eigensolver= 338 MB; <---B (Memory Increased)

      total number of factorizations          =              2 <---C
      total number of lanczos runs            =              1
      total number of lanczos steps           =             25
      total number of block solves            =             26 <---D
      time (cpu & wall) for structure input   =       7.093750       32.272710
      time (cpu & wall) for ordering          =      17.593750       18.044067
      time (cpu & wall) for symbolic factor   =       0.250000        2.124651
      time (cpu & wall) for value input       =       7.812500       76.157348

      time (cpu & wall) for numeric factor    =    1454.265625     1913.180302 <---F (Factor)
      computational rate (mflops) for factor  =    2498.442914     1899.141259
      time (cpu & wall) for numeric solve     =     304.515625     5609.588772 <---E (Solve Time)
      computational rate (mflops) for solve   =     842.142367       45.715563 <---G (Solve Rate)
      time (cpu & wall) for matrix multiply   =      30.890625       31.109154
      computational rate (mflops) for mult.   =     233.318482      231.679512

      cost for sparse eigenanalysis
      -----------------------------
      lanczos run start up cost               =      12.843750
      lanczos run recurrence cost             =     295.187500
      lanczos run reorthogonalization cost    =      88.359375
      lanczos run internal eigenanalysis cost =       0.000000
      lanczos eigenvector computation cost    =      11.921875
      lanczos run overhead cost               =       0.031250

      total lanczos run cost                  =     408.343750
      total factorization cost                =    1454.265625
      shift strategy and overhead  cost       =       0.328125

      total sparse eigenanalysis cost         =    1862.937500

      i/o stats:    unit           file length              amount transferred
                                  words     mbytes            words     mbytes
                   ----          -----     ------           -----     ------
                     20      27018724.      206. Mb      81056172.      618. Mb
                     21       3314302.       25. Mb      16571510.      126. Mb
                     22      12618501.       96. Mb     920846353.     7026. Mb
                     25      53273064.      406. Mb     698696724.     5331. Mb
                     28      51224100.      391. Mb     592150596.     4518. Mb
                      7     615815250.     4698. Mb   33349720462.   254438. Mb
                      9      31173093.      238. Mb     437892615.     3341. Mb
                     11      20489640.      156. Mb      20489640.      156. Mb

                  Total:     814926674.     6217. Mb   36117424072.   275554. Mb
  Block Lanczos         CP Time (sec) =       1900.578
  Block Lanczos    ELAPSED Time (sec) =       7852.267
  Block Lanczos    Memory Used  ( Mb) =        337.6
```

Example 7.4, "Windows 32-bit System Using Larger Memory" shows results from a run on the same Windows 32-bit system using an initial ANSYS memory allocation -m 1200 -db 256. With this change, the initial block of memory available for Block Lanczos is now over 800 MB (A). The final memory used for this Lanczos run is 588 MB (B). Note the initial memory allocation of 800 MB was actually reduced in this case because it was more than enough

to obtain optimal block size. The initial Lanczos memory allocation is reduced from 800 to 588 MB, and the memory saved is used to improve the solve performance.

This run uses a larger block size such that the number of block solves is reduced from 26 to 17 (C), resulting in a noticeable reduction in time for solves (5609 to 3518 (D)) and I/O to unit 7 (254 GB down to 169 GB (E)). The I/O performance on this desktop system is still poor, but increasing the initial memory reduces the Lanczos solution time from 7850 seconds to 5628 seconds (F). This example shows a clear case where you can obtain significant performance improvements just by increasing the initial memory allocation with the -m command line setting.

Example 7.4 Windows 32-bit System Using Larger Memory

500k DOFs Block Lanczos Run Computing 40 Modes

```
 MEMORY AVAILABLE FOR EIGENSOLVER =  813.69     (MB) <---A (Memory)
    MIN. TO COMPLETE VALUE INPUT           =     16333512     124.61     (MB)
    MIN. FOR PART IN-CORE AND OUT-OF-CORE  =     52074192     397.29     (MB)
    MIN. FOR IN-CORE                       =    761979034     5813.4     (MB)
    RECOM. FOR PART IN-CORE AND OUT-OF-CORE =     70069627     534.59     (MB)
    RECOM. FOR IN-CORE                     =    765564721     5840.8     (MB)

   Factorization is out-of-core
   Stiffness matrix is out-of-core
   Mass matrix is incore
   Optimal block size will be used

 Memory available for Block Lanczos eigensolver= 588 MB;  <---B (Memory Reduced)

      total number of factorizations         =              2
      total number of lanczos runs           =              1
      total number of lanczos steps          =             16
      total number of block solves           =             17 <---C (Num. Block Solves)
      time (cpu & wall) for structure input  =       5.093750        5.271888
      time (cpu & wall) for ordering         =      15.156250       16.276968
      time (cpu & wall) for symbolic factor  =       0.218750        0.745590
      time (cpu & wall) for value input      =       5.406250       23.938993

      time (cpu & wall) for numeric factor   =    1449.734375     1805.639936
      computational rate (mflops) for factor =    2506.251979     2012.250379
      time (cpu & wall) for numeric solve    =     221.968750     3517.910371 <---D (Solve)
      computational rate (mflops) for solve  =    1510.806453       95.326994
      time (cpu & wall) for matrix multiply  =      26.718750       27.120732
      computational rate (mflops) for mult.  =     369.941364      364.458107

      cost for sparse eigenanalysis
      -----------------------------
      lanczos run start up cost              =      14.812500
      lanczos run recurrence cost            =     213.703125
      lanczos run reorthogonalization cost   =      91.468750
      lanczos run internal eigenanalysis cost =       0.000000
      lanczos eigenvector computation cost   =      17.750000
      lanczos run overhead cost              =       0.078125

      total lanczos run cost                 =     337.812500
      total factorization cost               =    1449.734375
      shift strategy and overhead  cost      =       0.406250

      total sparse eigenanalysis cost        =    1787.953125

      i/o stats:    unit           file length              amount transferred
                                  words     mbytes            words     mbytes
                   ----          -----     ------            -----     ------

                     20      27014504.      206. Mb      81051952.      618. Mb
                     21       3314302.       25. Mb       9942906.       76. Mb
                     25      69664776.      532. Mb     692549832.     5284. Mb
                     28      65566848.      500. Mb     553220280.     4221. Mb
                      7     615815250.     4698. Mb   22169349000.   169139. Mb <---E (File)
                     11      20489640.      156. Mb      20489640.      156. Mb

                  Total:     801865320.     6118. Mb   23526603610.   179494. Mb
  Block Lanczos         CP Time (sec) =       1819.656
  Block Lanczos    ELAPSED Time (sec) =       5628.399 <---F (Total Time)
  Block Lanczos    Memory Used  ( Mb) =        588.0
```

A good rule of thumb for Block Lanczos runs is to always make sure the initial memory allocation follows the general guideline of 1 GB of memory per million DOFs; be generous with that guideline because Lanczos always uses more memory than the sparse solver. Most Windows 32-bit systems will not allow an initial memory allocation

larger than about 1200 MB, but this is enough to obtain a good initial Block Lanczos memory allocation for most problems up to 1 million DOFs.

Example 7.5, "Windows 32-bit System with 2 Processors and RAID0 I/O" shows that a Windows 32-bit system with a RAID0 I/O configuration and parallel processing (along with an initial memory allocation as used in *Example 7.4, "Windows 32-bit System Using Larger Memory"*) can reduce the modal analysis time even further. The initial Windows 32-bit system took over 2 hours to compute 40 modes; but when using RAID0 I/O and parallel processing with a larger initial memory allocation, this job ran in just over half an hour (A). This example shows that, with a minimal investment of around $1000 to add RAID0 I/O, the combination of adequate memory, parallel processing, and a RAID0 I/O array makes this imbalanced Windows 32-bit desktop system into an HPC resource. The only significant tuning required to obtain this performance was increasing the initial memory setting.

Example 7.5 Windows 32-bit System with 2 Processors and RAID0 I/O

500k DOFs Block Lanczos Run Computing 40 Modes

```
      total number of factorizations         =              2
      total number of lanczos runs           =              1
      total number of lanczos steps          =             16
      total number of block solves           =             17
      time (cpu & wall) for structure input  =       5.125000        5.174958
      time (cpu & wall) for ordering         =      14.171875       15.028077
      time (cpu & wall) for symbolic factor  =       0.265625        1.275690
      time (cpu & wall) for value input      =       5.578125        7.172501

      time (cpu & wall) for numeric factor   =    1491.734375      866.249038
      computational rate (mflops) for factor =    2435.688087     4194.405405
      time (cpu & wall) for numeric solve    =     213.625000      890.307840
      computational rate (mflops) for solve  =    1569.815424      376.669512
      time (cpu & wall) for matrix multiply  =      26.328125       26.606395
      computational rate (mflops) for mult.  =     375.430108      371.503567

  Block Lanczos         CP Time (sec) =       1859.109
  Block Lanczos    ELAPSED Time (sec) =       1981.863 <---A (Total Time)

  Block Lanczos    Memory Used  ( Mb) =        641.5
```

A final set of runs on a large memory desktop Windows 64-bit system demonstrates the current state of the art for Block Lanczos performance in ANSYS. The runs were made on an HP dual CPU quad-core system (8 processing cores total) with 32 GB of memory. This system does not have a RAID0 I/O configuration, but it is not necessary for this model because the system buffer cache is large enough to contain all of the files used in the Block Lanczos run.

Example 7.6, "Windows 64-bit System Using Default Memory" shows some of the performances statistics from a run which uses the same initial memory setting as the Windows 32-bit system. However, a careful comparison with *Example 7.3, "Windows 32-bit System Using Default Memory"* shows that the initial Lanczos memory allocation on this Windows 64-bit system is 465 MB (A) instead of 191 MB used on the Windows 32-bit system. ANSYS uses a more conservative initial memory allocation on systems limited by 32-bit address space. The larger initial memory allocation on the Windows 64-bit system is still not enough to obtain the optimum block size, but it is sufficient to keep the mass matrix in memory, and this run uses a larger block size than the Windows 32-bit system. Even without an optimal memory setting, *Example 7.6, "Windows 64-bit System Using Default Memory"* shows that this Windows 64-bit system using only 2 of 8 cores is over 2 times faster (B) than the best Windows 32-bit system results after tuning memory and using a RAID0 disk array. It is over 7 times faster than the initial Windows 32-bit system using a single processor, default memory, and a standard single disk drive.

Example 7.6 Windows 64-bit System Using Default Memory

500k DOFs Block Lanczos Run Computing 40 Modes; 8 Core, 32 MB Memory System

```
MEMORY AVAILABLE FOR EIGENSOLVER =  465.24     (MB) <---A (Memory)

  MEMORY TO PERFORM EIGENEXTRACTION: using opt Min core
   MIN. TO COMPLETE VALUE INPUT            =     16336724     124.64     (MB)
   MIN. FOR PART IN-CORE AND OUT-OF-CORE   =     52075873     397.31     (MB)
   MIN. FOR IN-CORE                        =    760866076     5804.9     (MB)
   RECOM. FOR PART IN-CORE AND OUT-OF-CORE =     70066163     534.56     (MB)
   RECOM. FOR IN-CORE                      =    764450734     5832.3     (MB)
  Factorization is out-of-core
  Stiffness matrix is out-of-core
  Mass matrix is incore

  This run uses less than optimal memory for out-of-core mode
  Increasing memory may improve performance

 *** NOTE ***                            CP =      56.078   TIME= 22:38:28
 Memory available for Block Lanczos eigensolver= 465 MB; required for
 in-core solution= 5805 MB.  Proceeding with part in-core and part
 out-of-core solution (achieves similar CPU times as in-core).
  Just before calculating eigenvalues, etc.

     total number of factorizations          =               2
     total number of lanczos runs            =               1
     total number of lanczos steps           =              19
     total number of block solves            =              20
     time (cpu & wall) for structure input   =        2.796875        2.802142
     time (cpu & wall) for ordering          =        8.343750        9.330948
     time (cpu & wall) for symbolic factor   =        0.156250        1.578073
     time (cpu & wall) for value input       =        5.484375       14.554428

     time (cpu & wall) for numeric factor    =      954.218750      483.430871
     computational rate (mflops) for factor  =     3792.080754     7484.988596
     time (cpu & wall) for numeric solve     =      321.890625      321.734786
     computational rate (mflops) for solve   =      917.619143      918.063610
     time (cpu & wall) for matrix multiply   =       10.921875       10.986604
     computational rate (mflops) for mult.   =      777.532347      772.951382

 Block Lanczos         CP Time (sec) =       1386.500
 Block Lanczos    ELAPSED Time (sec) =        998.024 <---B (Total Time)

 Block Lanczos    Memory Used  ( Mb) =        465.2
```

Example 7.7, "Windows 64-bit System Using Optimal Out-of-Core Memory" shows the effect of increasing the initial memory setting for the ANSYS run. The number of block solves is reduced to 17 (A), and the elapsed time to solution is reduced from 998 seconds to 955 (B). In *Example 7.8, "Windows 64-bit System Using Incore Memory"*, a further reduction in Lanczos solution time (C) is achieved using the **BCSOPTION**,,INCORE option. The solve time is reduced to just 127 seconds (D). This compares with 5609 seconds in the original 32-bit Windows runs and 890 seconds for the solve time using a fast RAID0 I/O configuration. Clearly, large memory is the best solution for I/O performance. It is important to note that all of the Windows 64-bit runs were on a large memory system. The performance of the out-of-core algorithm is still superior to any of the Windows 32-bit system results and is still very competitive with full incore performance on the same system. As long as the memory size is larger than the files used in the Lanczos runs, good balanced performance will be obtained, even without RAID0 I/O.

Example 7.7 Windows 64-bit System Using Optimal Out-of-Core Memory

500k DOFs Block Lanczos Run Computing 40 Modes; 8 Core, 32 MB Memory System

Windows 64-bit system run using –m 1200 –db 256 initial memory setting

```
  Factorization is out-of-core
  Stiffness matrix is out-of-core
  Mass matrix is incore
  Optimal block size will be used

  This run uses recommended optimal memory for out-of-core mode


 *** NOTE ***                            CP =      51.375   TIME= 22:55:56
 Memory available for Block Lanczos eigensolver= 588 MB; required for
 in-core solution= 5805 MB.  Proceeding with part in-core and part
 out-of-core solution (achieves similar CPU times as in-core).
  Just before calculating eigenvalues, etc.

     total number of lanczos steps           =             16
     total number of block solves            =             17 <---A (Num. Block Solves)
     time (cpu & wall) for structure input   =       2.656250        2.645897
     time (cpu & wall) for ordering          =       8.390625        9.603793
     time (cpu & wall) for symbolic factor   =       0.171875        1.722260
     time (cpu & wall) for value input       =       3.500000        5.428131

     time (cpu & wall) for numeric factor    =     943.906250      477.503474
     computational rate (mflops) for factor  =    3833.510539     7577.902054
     time (cpu & wall) for numeric solve     =     291.546875      291.763562
     computational rate (mflops) for solve   =    1148.206668     1147.353919
     time (cpu & wall) for matrix multiply   =      12.078125       12.102867
     computational rate (mflops) for mult.   =     801.106125      799.468441

     Block Lanczos         CP Time (sec) =      1336.828
     Block Lanczos    ELAPSED Time (sec) =       955.463 <---B (Total Time)

     Block Lanczos    Memory Used  ( Mb) =       588.0
```

Example 7.8 Windows 64-bit System Using Incore Memory

500k DOFs Block Lanczos Run Computing 40 Modes; 8 Core, 32 MB Memory System

Windows 64-bit system run specifying BCSOPTION,,incore

```
     total number of lanczos steps           =             16
     total number of block solves            =             17
     time (cpu & wall) for structure input   =       2.828125        2.908010
     time (cpu & wall) for ordering          =       8.343750        9.870719
     time (cpu & wall) for symbolic factor   =       0.171875        1.771356
     time (cpu & wall) for value input       =       2.953125        3.043767

     time (cpu & wall) for numeric factor    =     951.750000      499.196647
     computational rate (mflops) for factor  =    3801.917055     7248.595484
     time (cpu & wall) for numeric solve     =     124.046875      127.721084 <---D (Solve)
     computational rate (mflops) for solve   =    2698.625548     2620.992983
     time (cpu & wall) for matrix multiply   =      12.187500       12.466526
     computational rate (mflops) for mult.   =     793.916711      776.147235

     Block Lanczos         CP Time (sec) =      1178.000
     Block Lanczos    ELAPSED Time (sec) =       808.156 <---C (Total Time)

     Block Lanczos    Memory Used  ( Mb) =      6123.9
```

7.1.3. Summary of Lanczos Performance and Guidelines

The examples described above demonstrate that Block Lanczos performance is influenced by competing demands for memory and I/O. *Table A.10, "Summary of Block Lanczos Memory Guidelines"* summarizes the memory usage guidelines illustrated by these examples. The critical performance factor in Block Lanczos is the time required for the block solves. Users should observe the number of block solves as well as the measured solve rate. The block size actually used in each run is not reported in the performance statistics in ANSYS Release 11.0 and earlier (the block size reported in these tables is the starting block size, not the actual size used after memory is divided up). Users can compare the number of block solves to the number of modes requested to get some indication of the block size. Generally, the number of block solves times the block size used will be at least 2.5 times the number of modes computed. Increasing the initial ANSYS memory allocation for large Lanczos runs using the 1 GB/MDOF guideline is a good strategy to obtain optimal out-of-core mode and optimal block size in most cases.

The 500k DOF Lanczos example is a large model for a Windows 32-bit desktop system, but an easy incore run for a large memory Windows 64-bit system like the 32 GB memory system described above. When trying to optimize ANSYS solver performance, it is important to know the expected memory usage for a given ANSYS model and compare that memory usage to the physical memory of the computer system. It is better to run a large Block Lanczos job in optimal out-of-core mode with enough memory allocated to easily run with the mass matrix in memory and a block size of 8 (this is the recommended out-of-core memory setting reported in the performance statistics) than to attempt running incore using up all or nearly all of the physical memory on the system.

For users who have more than 16 GB of memory available, the -lm version of ANSYS allows sparse solver and Block Lanczos runs that exceed 16 GB of memory (see *Section 5.1.1.3: Using the Large Memory (-lm) Option*). With the -lm version, even modal analyses with over 1 million DOFs can run incore on very large memory systems. The easiest way to run an incore Lanczos run on a large memory machine is to start ANSYS with a default memory setting that is consistent with the 1 GB per million DOFs rule, and then use **BCSOPTION**,,INCORE to direct the Block Lanczos routines to allocate whatever is necessary to run incore. Once the incore memory is known for a given problem, it is possible to start ANSYS with enough memory initially so that the Block Lanczos solver run will run incore automatically, and the matrix assembly phase can use part of the same memory used for Lanczos. This trial and error approach is helpful in conserving memory, but is unnecessary if the memory available is sufficient to easily run the given job.

A common error made by users on large memory systems is to start ANSYS with a huge initial memory allocation that is not necessary. This initial allocation limits the amount of system memory left to function as a buffer to cache ANSYS files in memory. It is also common for users to increase the memory allocation at the start of ANSYS, but just miss the requirement for incore sparse solver runs. In that case, the sparse solver will still run out-of-core, but often at a reduced performance level because less memory is available for the system buffer cache. Large memory systems function well using default memory allocations in most cases. Incore solver performance is not required on these systems to obtain very good results, but it is a valuable option for time critical runs when users can dedicate a large memory system to a single large model.

7.2. Distributed ANSYS Examples

The following topics are discussed in this section:

7.2.1. Distributed ANSYS Memory and I/O Considerations

Distributed ANSYS (DANSYS) is a distributed memory parallel version of ANSYS that uses MPI (message passing interface) for communication between DANSYS processes. Each MPI process is a separate ANSYS process, with each opening ANSYS files and allocating memory. In effect, each MPI process functions as a separate ANSYS job for much of the execution time. The global solution obtained in a DANSYS run is obtained through communication using an MPI software library. The master process initiates DANSYS runs, decomposes the global model into separate domains for each process, and collects results at the end to form a single results file for the entire model. ANSYS memory requirements are always higher for the master process than for the remaining processes because of the requirement to store the database for the entire model. Also, some additional data structures are maintained only on the master process, thus increasing the memory requirements for this process.

Since each process has separate I/O to its own set of files, the I/O demands for a cluster system can be substantial. Some cluster systems use a centralized I/O setup where all processing nodes write to one file system using the same interconnects that are responsible for MPI communication. While this setup has cost advantages and simplifies some aspects of cluster file management, it can lead to a significant performance bottleneck for DANSYS. If each MPI process in DANSYS has fast local disks, then a natural scaling of I/O performance can be expected. For cluster systems that do not have fast independent I/O for each process, it is important to know how to avoid I/O when possible, particularly with the distributed sparse direct solver. Note that current multicore systems also write to a single file system and fall into this category. It is important to understand the I/O and memory requirements for each solver type, direct and iterative; they are discussed separately with examples.

7.2.2. Distributed ANSYS Sparse Solver Example

The distributed sparse solver used in DANSYS, referred to as DSPARSE, is not the same sparse solver used for SMP ANSYS runs. It operates both incore and out-of-core, just as the SMP sparse solver; but there are important differences. Three important factors affect the parallel performance of DSPARSE: memory, I/O, and load balance.

For out-of-core memory mode runs, the optimal memory setting is determined so that the largest fronts are incore during factorization. The size of this largest front is the same for all processes, thus the memory required by each process to factor the matrix out-of-core is often similar. This means that the total memory required to factor the matrix out-of-core also grows as more processors are used.

By contrast, the total memory required for the incore memory mode for DSPARSE is essentially constant as processors are added. This means that the memory per process actually shrinks when more processors are used. Interestingly, when enough processors are used, the incore and optimal out-of-core memory modes become equivalent. This usually occurs with more than 4 processors. In this case DSPARSE runs may switch from I/O dominated out-of-core performance to incore performance. Even in cases where incore memory is larger than optimal out-of-core memory, adding additional processors may make the incore memory option possible where it was not possible using fewer processors.

The load balance factor cannot be directly controlled by the user. The DSPARSE solver internally decomposes the input matrix so that the total amount of work done by all processes to factor the matrix is minimal. However, this decomposition does not necessarily result in a perfectly even amount of work for all processes. Thus, some processes may finish before others, resulting in a load imbalance.

Parts 1, 2, and 3 of *Example 7.9, "DSPARSE Solver Run for 750k DOF Static Analysis "* show DSPARSE performance statistics that illustrate the memory, I/O and load balance factors for a 750k DOF static analysis. In these runs the single processor times come from the sparse solver in ANSYS. The memory required to run this model incore is over 13 GB on one processor. The cluster system used in this example has 4 blades, each with 6 GB of memory, two dual core processors, and a standard, inexpensive GigE interconnect. The total system memory of 24 GB is more than enough to run this model incore, except that the total memory is not globally addressable. Therefore, only runs that use all 4 nodes can run incore. All I/O goes through the system interconnect, and all files used are

accessed from a common file system located on the host processor. The disk performance in this system is less than 50 MB/sec. This is typical of disk performance on systems that do not have a RAID0 configuration. For this cluster configuration, all processors share the same disk resource and must transfer I/O using the system interconnect hardware.

Part 1 of *Example 7.9, "DSPARSE Solver Run for 750k DOF Static Analysis "* shows that I/O cost increases both factorization and the solves for a 2 processor DANSYS run. The solve I/O rate measured in the DANSYS run was just over 30 MB/sec (A), while the I/O rate for the SMP sparse solver was just over 50 MB/sec (B). This I/O performance difference reflects the increased cost of I/O from two processors sharing a common disk through the standard, inexpensive processor interconnect. The memory required for a 2 processor DANSYS run exceeds the 6 GB available on each processor; thus, the measured effective I/O rate is a true indication of I/O performance on this configuration. Clearly, having local disks on each blade would significantly speed up the 2 processor job as the I/O rate would be doubled and less communication would be done over the interconnect.

Part 2 of *Example 7.9, "DSPARSE Solver Run for 750k DOF Static Analysis "* shows performance statistics for runs using the out-of-core and incore memory modes with the DSPARSE solver and 4 processors on the same system. In this example, the parallel runs were configured to use all available blades as processors were added, rather than using all available cores on the first blade before adding a second blade. Note that the 4 processor out-of-core run shows a substantial improvement compared to the 2 processor run and also shows a much higher effective I/O rate—over 260 MB/sec (C). This higher performance reflects the fact that the 4 processor out-of-core run now uses the total system memory, and each processor has enough local memory to cache its part of the large matrix factor. The 4 processor incore run is now possible because the incore memory requirement per processor, averaging just over 4 GB per process (D), is less than 6 GB per available processor. The performance gain is nearly 2 times over the out-of-core runs, and the solve time drops to just 3 seconds. The I/O cost, reflected in the wall times for the forward/backward solve, is reduced from 710 seconds on 2 processors (E) to just 3 seconds on 4 processors (F).

Part 3 of *Example 7.9, "DSPARSE Solver Run for 750k DOF Static Analysis "* shows incore runs using 6 and 8 processors for the same model. The 6 processor run shows the effect of load balancing on parallel performance. Though load balancing is not directly measured in the statistics reported, the amount of memory required on each processor is an indirect indicator that some processes have more of the matrix factor to store (and compute) than other processes. Load balance for the DSPARSE solver is never perfectly even and is dependent on the problem and the number of processors. In some cases, 6 processors will provide an almost perfect load balance, while in others 4 or 8 is better.

For all of the incore runs in Parts 2 and 3, the amount of memory required per processor decreases, even though total memory usage is roughly constant as the number of processors increases. It is this phenomenon that provides a speedup of over 6X on 8 processors in this example (1903 seconds down to 307 seconds). This example illustrates all three performance factors and shows the effective use of memory on a multi-blade cluster configuration.

Example 7.9 DSPARSE Solver Run for 750k DOF Static Analysis

Part 1: Out-of-Core Performance Statistics on 1 and 2 Processors

```
Sparse solver in ANSYS using 1 processor, optimal out-of-core mode

     time (cpu & wall) for structure input   =        4.770000        4.857412
     time (cpu & wall) for ordering          =       18.260000       18.598295
     time (cpu & wall) for symbolic factor   =        0.320000        0.318829
     time (cpu & wall) for value input       =       16.310000       57.735289
     time (cpu & wall) for numeric factor    =     1304.730000     1355.781311
     computational rate (mflops) for factor  =     5727.025761     5511.377286
     condition number estimate               =      0.0000D+00
     time (cpu & wall) for numeric solve     =       39.840000      444.050982
     computational rate (mflops) for solve   =      149.261126       13.391623
     effective I/O rate (Mb/sec) for solve   =      568.684880       51.022082 <---B (I/O)
```

```
     i/o stats:    unit           file length              amount transferred
                                words     mbytes            words     mbytes
                  ----          -----     ------           -----     ------
                    20      97789543.      746. Mb     202276543.     1543. Mb
                    25       5211004.       40. Mb      18238514.      139. Mb
                     9    1485663140.    11335. Mb    4699895648.    35857. Mb
                    11     125053800.      954. Mb     493018329.     3761. Mb

                -------     ----------     --------      ----------    --------
                Totals:    1713717487.    13075. Mb    5413429034.    41301. Mb

  Sparse Solver Call     1 Memory   ( Mb) =       1223.6
  Sparse Matrix Solver     CPU Time (sec) =       1385.330
  Sparse Matrix Solver ELAPSED Time (sec) =       1903.209 <---G (Total Time)
```

DSPARSE solver in DANSYS using 2 processors, out-of-core mode

```
---------------------------------------------------------------
  ---------DISTRIBUTED SPARSE SOLVER RUN STATS-------------------
  ---------------------------------------------------------------

  Memory allocated on processor   0        =   831.216 MB
  Memory allocated on processor   1        =   797.632 MB
  Total Memory allocated by all processors =  1628.847 MB

  Time(cpu & wall) for input matrix indices          0.67        0.69
  Time(cpu & wall) for reordering & sym.factor      15.27       16.38
  Time(cpu & wall) for input matrix values           2.30        2.33
  Time(cpu & wall) for matrix redistribution        12.89       25.60
  Time(cpu & wall) for matrix factorization        791.77     1443.04
  Computational rate (mflops) for factor          5940.07     3259.21
  Time(cpu & wall) for matrix forwd/back solve      23.48      710.15 <---E (Solve)
  Computational rate (mflops) for solve            253.25        8.37
  Effective I/O rate (Mb/sec) for solve            964.90       31.90 <---A (I/O)
  Time(cpu & wall) for D-sparse total solution      846.38     2198.19
```

Part 2: Out-of-Core and Incore Performance Statistics on 4 Processors

DSPARSE solver in DANSYS using 4 processors, out-of-core mode

```
  ---------------------------------------------------------------
  ---------DISTRIBUTED SPARSE SOLVER RUN STATS-------------------
  ---------------------------------------------------------------

  Memory allocated on processor   0        =   766.660 MB
  Memory allocated on processor   1        =   741.328 MB
  Memory allocated on processor   2        =   760.680 MB
  Memory allocated on processor   3        =   763.287 MB
  Total Memory allocated by all processors =  3031.955 MB

  Time(cpu & wall) for matrix factorization        440.33      722.55
  Computational rate (mflops) for factor         10587.26     6451.98
  Time(cpu & wall) for matrix forwd/back solve       9.11       86.41
  Computational rate (mflops) for solve            652.52       68.79
  Effective I/O rate (Mb/sec) for solve           2486.10      262.10 <---C (I/O)
  Time(cpu & wall) for D-sparse total solution      475.88      854.82
```

DSPARSE solver in DANSYS using 4 processors, incore mode

```
  ---------------------------------------------------------------
  ---------DISTRIBUTED SPARSE SOLVER RUN STATS-------------------
  ---------------------------------------------------------------

  Memory allocated on processor   0        =  4734.209 MB <---D
  Memory allocated on processor   1        =  4264.859 MB <---D
  Memory allocated on processor   2        =  4742.822 MB <---D
  Memory allocated on processor   3        =  4361.079 MB <---D
  Total Memory allocated by all processors = 18102.970 MB
```

```
  Time(cpu & wall) for matrix factorization         406.11      431.13
  Computational rate (mflops) for factor          11479.37    10813.12
  Time(cpu & wall) for matrix forwd/back solve        2.20        3.27 <---F (Solve)
  Computational rate (mflops) for solve            2702.03     1819.41
  Effective I/O rate (Mb/sec) for solve           10294.72     6931.93
  Time(cpu & wall) for D-sparse total solution      435.22      482.31
```

Part 3: Out-of-Core and Incore Performance Statistics on 6 and 8 Processors

```
 DSPARSE solver in DANSYS using 6 processors, incore  mode

  ---------------------------------------------------------------
  ---------DISTRIBUTED SPARSE SOLVER RUN STATS-------------------
  ---------------------------------------------------------------

  Memory allocated on processor   0        =  2468.203 MB
  Memory allocated on processor   1        =  2072.919 MB
  Memory allocated on processor   2        =  2269.460 MB
  Memory allocated on processor   3        =  2302.288 MB
  Memory allocated on processor   4        =  3747.087 MB
  Memory allocated on processor   5        =  3988.565 MB
  Total Memory allocated by all processors = 16848.523 MB

  Time(cpu & wall) for matrix factorization         254.85      528.68
  Computational rate (mflops) for factor          18399.17     8869.37
  Time(cpu & wall) for matrix forwd/back solve        1.65        6.05
  Computational rate (mflops) for solve            3600.12      981.97
  Effective I/O rate (Mb/sec) for solve           13716.45     3741.30
  Time(cpu & wall) for D-sparse total solution      281.44      582.25


 DSPARSE solver in DANSYS using 8 processors, incore  mode

  ---------------------------------------------------------------
  ---------DISTRIBUTED SPARSE SOLVER RUN STATS-------------------
  ---------------------------------------------------------------

  Memory allocated on processor   0        =  2382.333 MB
  Memory allocated on processor   1        =  2175.502 MB
  Memory allocated on processor   2        =  2571.246 MB
  Memory allocated on processor   3        =  1986.730 MB
  Memory allocated on processor   4        =  2695.360 MB
  Memory allocated on processor   5        =  2245.553 MB
  Memory allocated on processor   6        =  1941.285 MB
  Memory allocated on processor   7        =  1993.558 MB
  Total Memory allocated by all processors = 17991.568 MB

  Time(cpu & wall) for matrix factorization         225.36      258.47
  Computational rate (mflops) for factor          20405.51    17791.41
  Time(cpu & wall) for matrix forwd/back solve        2.39        3.11
  Computational rate (mflops) for solve            2477.12     1903.98
  Effective I/O rate (Mb/sec) for solve            9437.83     7254.15
  Time(cpu & wall) for D-sparse total solution      252.21      307.06 <---H (Solve)
```

7.2.3. Guidelines for Iterative Solvers in Distributed ANSYS

Iterative solver performance in DANSYS does not require the I/O resources that are needed for out-of-core direct sparse solvers. There are no memory tuning options required for PCG solver runs, except in the case of LANPCG modal analysis runs which use the *Lev_Diff* = 5 preconditioner option on the **PCGOPT** command. This option uses a direct factorization, similar to the sparse solver, so additional memory and I/O requirements are added to the cost of the PCG iterations.

Performance of the PCG solver can be improved in some cases by changing the preconditioner options using the **PCGOPT** command. Increasing the *Lev_Diff* value on **PCGOPT** will usually reduce the number of iterations required for convergence, but at a higher cost per iteration.

Since the bulk of this higher cost is spent on the head node (node 0), the higher *Lev_Diff* settings tend not to scale as well. If the iterative solver is run on a parallel system with more than 8 processors, users may choose *Lev_Diff* = 1 because this option will normally exhibit the best parallel speedups. However, if a model exhibits very slow convergence, evidenced by high iteration count in Jobname.PCS, then *Lev_Diff* = 2 or 3 may give the best time to solution even though speedups are not as high as the less expensive preconditioners.

Memory requirements for the PCG solver will increase as the *Lev_Diff* value increases. The default heuristics that choose which preconditioner option to use are based on element types and element aspect ratios. The heuristics are designed to use the preconditioner option that results in the least time to solution. Users may wish to experiment with different choices for *Lev_Diff* if a particular type of model will be run over and over.

Parts 1 and 2 of *Example 7.10, "BMD-7 Benchmark - 5 MDOF, PCG Solver "* show portions of Jobname.PCS for the BMD-7 DANSYS benchmark problem run on a 4-blade cluster of 2 dual-core processors. In Part 1, the command **PCGOPT**,1 is used to force the preconditioner level of difficulty equal to 1. Part 2 shows the same segments of Jobname.PCS using the command **PCGOPT**,2. This output shows the model has 5.3 million degrees of freedom and uses the memory saving option for the entire model (all elements are implicit and 0 nonzeros in the assembled matrix). For Lev_Dif = 1 (Part 1), the size of the preconditioner matrix is just over 90 million coefficients (A), while in Part 2 the *Lev_Diff* = 2 preconditioner matrix is 165 million coefficients (B). Part 1 demonstrates higher parallel speedup than Part 2 (over 7 on 16 processors versus 5.5 in Part 2). However, the total elapsed time is lower in Part 2 for both 1 and 16 processors, respectively. The reduced parallel speedups in Part 2 result from the higher preconditioner cost and lesser scalability for the preconditioner used when *Lev_Diff* = 2. This example illustrates that sometimes a trade-off in parallel performance to improve preconditioner performance results in the fastest time to solution. ANSYS heuristics attempt to automatically optimize time to solution for the PCG solver preconditioner options, but in some cases users may obtain better performance by changing the level of difficulty manually.

The outputs shown in *Example 7.10, "BMD-7 Benchmark - 5 MDOF, PCG Solver "* report memory use for both preconditioner options. The peak memory usage for the PCG solver occurs only briefly during preconditioner construction, and using virtual memory for this short time does not significantly impact performance. However, if the PCG memory usage value reported in the PCS file is larger than available physical memory, each PCG iteration will require slow disk I/O and the PCG solver performance will be much slower than expected. This 5.3 Million DOF example shows the effectiveness of the **MSAVE**,ON option in reducing the expected memory use of 5 GB (1 GB/MDOF) to just over 1 GB (C) for **PCGOPT**,1 and 1.3 GB (D) for **PCGOPT**,2. Unlike the sparse solver memory requirements, memory grows dynamically during the matrix assembly portion of the PCG solver, and performance is not dependent on the initial memory allocation size in most cases. This characteristic is especially important for smaller memory systems, particularly Windows 32-bit systems. Users with 4 GB of memory can effectively extend their PCG solver memory capacity by nearly 1 GB using the /3GB switch described earlier. This 5 million DOF example could easily be solved using one processor on a Windows 32-bit system with the /3GB switch enabled.

Example 7.10 BMD-7 Benchmark - 5 MDOF, PCG Solver

Part 1: Lev_Diff = 1 on 1 and 16 Processors

```
Number of Processors: 1
   Degrees of Freedom: 5376501
  DOF Constraints: 38773
  Elements: 427630
   Assembled: 0
   Implicit: 427630
  Nodes: 1792167
  Number of Load Cases: 1

  Nonzeros in Upper Triangular part of
              Global Stiffness Matrix : 0
  Nonzeros in Preconditioner: 90524940 <---A (Preconditioner Size)

  *** Level of Difficulty: 1   (internal 0) ***

  Total Iterations In PCG: 1138

  DETAILS OF PCG SOLVER SOLUTION TIME(secs)     Cpu        Wall
       Preconditioned CG Iterations         2757.19     2780.98
            Multiply With A                 2002.80     2020.13
                 Multiply With A22          2002.80     2020.12
            Solve With Precond               602.66      607.87
                 Solve With Bd               122.93      123.90
                 Multiply With V             371.93      375.10
                 Direct Solve                 75.35       76.07
 ********************************************************************************
       TOTAL PCG SOLVER SOLUTION CP TIME      =  2788.09 secs
       TOTAL PCG SOLVER SOLUTION ELAPSED TIME =  2823.12 secs
 ********************************************************************************
  Total Memory Usage at CG         :  1738.13 MB
  PCG Memory Usage at CG           :  1053.25 MB <---C (Memory)
 ********************************************************************************

  Number of Hosts/Processors: 16

  Total Iterations In PCG: 1069

  DETAILS OF PCG SOLVER SOLUTION TIME(secs)     Cpu        Wall
       Preconditioned CG Iterations          295.00      356.81
            Multiply With A                  125.38      141.60
                 Multiply With A22           121.40      123.27
            Solve With Precond               141.05      165.27
                 Solve With Bd                19.69       20.12
                 Multiply With V              31.46       31.83
                 Direct Solve                 77.29       78.47
 ********************************************************************************
       TOTAL PCG SOLVER SOLUTION CP TIME      =  5272.27 secs
       TOTAL PCG SOLVER SOLUTION ELAPSED TIME =   390.70 secs
 ********************************************************************************
    Total Memory Usage at CG         :  3137.69 MB
    PCG Memory Usage at CG           :  1894.20 MB
 ********************************************************************************
```

Part 2: Lev_Diff = 2 on 1 and 16 Processors

```
Number of Processors: 1
   Degrees of Freedom: 5376501
  Elements: 427630
   Assembled: 0
   Implicit: 427630
  Nodes: 1792167
  Number of Load Cases: 1

  Nonzeros in Upper Triangular part of
              Global Stiffness Matrix : 0
  Nonzeros in Preconditioner: 165965316 <---B (Preconditioner Size)
```



```
 *** Level of Difficulty: 2   (internal 0) ***

 Total Iterations In PCG: 488

 DETAILS OF PCG SOLVER SOLUTION TIME(secs)     Cpu        Wall
      Preconditioned CG Iterations         1274.89     1290.78
           Multiply With A                  860.62      871.49
           Solve With Precond               349.42      353.55
                Solve With Bd                52.20       52.71
                Multiply With V             106.84      108.10
                Direct Solve                176.58      178.67
*******************************************************************************
      TOTAL PCG SOLVER SOLUTION CP TIME      =  1360.11 secs
      TOTAL PCG SOLVER SOLUTION ELAPSED TIME =  1377.50 secs
*******************************************************************************
 Total Memory Usage at CG         :  2046.14 MB
 PCG Memory Usage at CG           :  1361.25 MB <---D (Memory)
*******************************************************************************

 Number of Hosts/Processors: 16

 Total Iterations In PCG: 386

 DETAILS OF PCG SOLVER SOLUTION TIME(secs)     Cpu        Wall
      Preconditioned CG Iterations          148.54      218.87
           Multiply With A                   45.49       50.89
           Solve With Precond                94.29      153.36
                Solve With Bd                 5.67        5.76
                Multiply With V               8.67        8.90
                Direct Solve                 76.43      129.59
*******************************************************************************
      TOTAL PCG SOLVER SOLUTION CP TIME      =  2988.87 secs
      TOTAL PCG SOLVER SOLUTION ELAPSED TIME =   248.03 secs
*******************************************************************************
 Total Memory Usage at CG         :  3205.61 MB
 PCG Memory Usage at CG           :  1390.81 MB
*******************************************************************************
```

7.2.4. Summary

Table A.11, "Performance Guidelines for ANSYS and DANSYS Solvers" summarizes performance guidelines for ANSYS and DANSYS solvers. The examples presented in the previous sections illustrate how experienced users can tune memory and other solver options to maximize system performance. The best first step is to maximize ANSYS performance on an HPC system with a balance of fast processors, generous memory, and fast disks. On a balanced HPC system, experienced users can become aware of the demands of the solver used for a given class of problems. Sparse solver users will benefit most from large memory, but they can also significantly improve I/O performance using new RAID0 I/O capabilities on Windows desktop systems. PCG solver users do not need to worry so much about I/O performance, and memory usage is often less than the sparse solver. Choosing the best preconditioner option may reduce solution time significantly.

Obtaining maximum performance in ANSYS or DANSYS is an optimization problem with hardware and software components. This document explains how to obtain true HPC systems at every price point, including single user desktops, personal clusters, multicore desk side large memory systems, large cluster systems, and traditional large memory shared memory servers. ANSYS has been designed to run in parallel and to exploit large memory resources to handle the huge computing demands of today's sophisticated simulations. User expertise with ANSYS solver parameters can result in significant improvements in simulation times. Future releases of ANSYS will continue to endeavor to make default solver options optimal in most cases, but it will always be possible to tune performance because of the variety of system configurations available.

Appendix A. Tables

This chapter contains all the tables referenced in previous chapters.

Table A.1 Comparison of CPU Characteristics and Performance

Vendor	Processor	Cache	Clock Freq.	Flops/Clock	Peak Flop Rate	Observed Peak Rate
Intel	Xeon core micro-architecture	4MB (2MB/core)	3.0 GHz	4	12 Gflops	5-6 Gflops
IBM	Power 5	1.9 MB	1.9-2.3 GHz	4	7.6-9.2 Gflops	4-5 Gflops
Intel	Itanium	6-24 MB	1.5-1.6 GHz	4	6-6.4 Gflops	4-4.5 Gflops
Intel	Pentium/Xeon	2MB	3.6 GHz	2	7.2 Gflops	3-3.5 Gflops
AMD	Opteron Dual-Core	2MB (1MB/core)	2.8 GHz	2	5.6 Gflops	2.2-2.7 Gflops
Sun	UltraSparc		1.3 GHz	2	2.6 Gflops	1.5 Gflops
This table is only a representative list of current CPU offerings. For more detailed comparisons and characteristics, consult manufacturers. Observed peak rates are taken from sparse solver factorization times in ANSYS using 1 processor.						

Table A.2 I/O Hardware

Hardware	Type	Comments
Disk	SCSI	More expensive, faster, reliable.
	SATAII	Cheaper, higher capacity, more common on desktop.
	SAS	Newer version of SCSI drives found in some HPC systems.
	Maximum sustainable I/O rate for single drive on most desktop systems is 30 - 70 MB/sec.	
Controller	RAID0	Maximum speed, multiple drives as one disk partition, no backup of data, recommended for ANSYS jobs.
	RAID1	Duplicate copies of files, used for critical data not speed.
	RAID5	Compromise between speed and backup, uses a single parity disk rather than duplication of all files.

Recommended configuration:

- Use a single large drive for system and permanent files.
- Use a separate disk partition of 4 or more identical physical drives for ANSYS working directory. Use a striped array (on UNIX/Linux) or RAID0 across the physical drives.
- Size of ANSYS working directory should be 100-200 GB, preferably <1/3 of total RAID drive capacity.
- Keep working directory clean, and defragment or reformat regularly.
- Set up a swap space equal to physical memory size.

Table A.3 Recommended Single Box System Configurations (SMP ANSYS or DANSYS)

System Use	CPU	Memory	I/O
Desktop Windows 32-bit	1 dual-core processor	4 GB using /3GB switch for large models, swap space at least 1 GB	Separate 200 GB disk partition RAID0 array of 4 disks
Desktop Windows 64-bit / Linux 64	2 dual- or quad-core processors	8 GB <500K DOFs; 16 GB >500K DOFs	Same as Windows 32-bit, but RAID0 not as critical if memory is sufficient
Deskside Server Windows 64-bit / Linux 64	4-8 dual- or quad-core processors	16 GB single users; 64 GB multi-user	RAID0 array of 4-8 disks; 500 GB - 1 Tbyte space; swap space file equal to physical memory size
Department or Corporate Level Server	8 or more dual- or quad-core processors	64 GB and up, 8 GB per processor recommended	Multi-disk, multi-controller striped working directory space; 1 Tbyte minimum recommended size; swap space equal to physical memory size.

Table A.4 Recommended Cluster System Configurations (DANSYS)

System Use	CPU	Memory	I/O	Interconnect
Windows personal cluster	4 dual-core processors. Can be 4 small dual-core boxes or one 4 processor board system.	4 GB/processor 8 GB preferred for processor 0	Potential weak spot; separate disks per processor preferred; 40-80 GB each node	GigE
Rack cluster	1 or 2 dual-core processors per board; Total cores 8 to 16.	8 GB per dual-core processor	80 GB per processor board	Infiniband preferred; GigE is sufficient
Large corporate - wide cluster	32 processors and up; DANSYS will use a subset of the total processors	8 GB per processor; prefer 16 GB on processor 0	Local disk on each processor board preferred; Fast central disk resource possible on high-end systems	Infiniband or other high speed interconnect
Cluster of large servers	2 dual- or quad-core processors in an SMP system	16 GB per SMP system	200 GB on each SMP box; potential weak spot if SMP box has a single slow drive; RAID0 array is best	Shared memory inside SMP box; one fast switch between the boxes; Infiniband recommended
The configurations shown in this table are applicable to DANSYS, but can also be used for Workbench remote execution (Remote Solve Manager).				

Table A.5 HPC Recommendations for Various ANSYS User Scenarios

Scenario	Hardware Recommendation	Usage Guidelines
ANSYS/WB user.	Windows platform, dual core processor, Windows 64-bit is recommended to support growth of model sizes. Though not recommended for optimal performance, Windows 32-bit is adequate with 2-4 GB memory.	• Learn to monitor system performance. • Start using parallel processing; watch for signs of growing memory use or sluggish system performance.
ANSYS/WB Windows 32-bit user starting to run out of memory occasionally or experiencing sluggish system performance.	Upgrade to Windows 64-bit desktop with 1 or 2 dual-core processors, 8 GB of memory. Consider remote job execution on a personal cluster or Linux 64 remote server.	• Use parallel processing routinely. • Monitor system performance and memory usage. • Consider using Mechanical HPC licenses to increase parallel processing beyond 2 cores.
Several WB users in a company experiencing increased demand for simulation capabilities.	Individual users upgrade to Windows 64-bit, dual core with 8 GB memory. Consider adding a Windows Server 2003 cluster or Linux 64 cluster for remote execution.	• Setup remote execution from WB using Remote Solve Manager. • Use parallel processing; monitor system performance. • Use mechanical HPC licenses to run on the cluster system.
ANSYS users using large models and advanced simulation capabilities.	Deskside Windows 64-bit system, 16-64 GB memory. **or** Add deskside Windows 64-bit or Linux 64-bit server for remote job execution to off-load larger or long running jobs from a Windows 64-bit desktop system. **or** Use a cluster configuration for remote execution of large or long running jobs using DANSYS.	• Use parallel processing routinely; monitor system performance including CPU performance and disk usage. • Use mechanical HPC licenses to run 4-8 cores per job for maximum simulation performance. • Learn how to measure CPU and I/O performance from ANSYS output as well as memory usage.
Large company with ANSYS users, continual simulation load, and the demand to occasionally solve barrier-breaking sized models.	High-end large memory SMP server, 64 GB or more main memory, 8 or more processors	• Use parallel processing with mechanical HPC licenses to run DANSYS. • Monitor system performance and understand how to measure CPU and I/O performance as well as memory usage and affinity.

Table A.6 ANSYS and DANSYS Direct Sparse Solver Memory and Disk Estimates

Memory Mode	Memory Usage Estimate	I/0 Files Size Estimate
ANSYS Sparse Direct Solver		
Default (out-of-core)	1 GB/MDOFs	10 GB/MDOFs
Incore	10 GB/MDOFs	1 GB/MDOFs 10 GB/MDOFs if workspace is saved to `Jobname.LN22`.
DANSYS Dsparse Direct Solver Using p Processors		
Default (out-of-core)	1 GB/MDOF on master node .7 GB/MDOF on all other nodes	10 GB/MDOFs *1 / p Matrix factor is stored on disk evenly on p processors
Incore	10 GB/MDOF * 1/p Matrix factor is stored in memory evenly incore on p processors. Additional 1.5 GB/MDOFs required on master node to store input matrix	

Comments:

- Double memory estimates for non-symmetric systems.
- Double memory estimates for complex valued systems.
- Add 30 % to out-of-core memory for 3-D models with high order elements.
- Subtract 40 % to out-of-core memory for 2-D or beam/shell element dominated models.
- Add 50 -100 % to file size for incore memory for 3-D models with high order elements.
- Subtract 50 % to file size for incore memory for 2-D or beam/shell element dominated models.

Table A.7 ANSYS and DANSYS Iterative PCG Solver Memory and Disk Use Estimates

ANSYS PCG Solver Memory and Disk Estimates
• Basic Memory requirement is 1 GB/MDOFs • Basic I/O Requirement is 1.5 GB/MDOFs
DANSYS PCG Solver Memory and Disk Estimates Using p Processors
• Basic Memory requirement is 1.5 GB/MDOFs – Each processor uses 1.2 GB/MDOFs * 1/p – Add ~300 MB/MDOFs for master process • Basic I/O Requirement is 2.0 GB/MDOFs – Each process consumes 2 GB/MDOFs * 1/p file space – File sizes are nearly evenly divided on processors
Comments: • Add 30% for higher-order solid elements • Add 10-50% for higher level of difficulty preconditioners (PCGOPT 2-4) • Save up to 70% from **MSAVE** option by default when applicable – SOLID92 / SOLID95 / SOLID186 / SOLID187 elements – Static analyses with small deflections (**NLGEOM**,OFF) • Save up to 50% forcing **MSAVE**,ON – SOLID45 / SOLID185 elements (slower runtime) – **NLGEOM**,ON for SOLID45 / SOLID92 / SOLID95 elements (slower runtime)

Table A.8 ANSYS and DANSYS Eigensolver Memory and Disk Space Estimates

ANSYS Block Lanczos (LANB) Using Sparse Solver
• Default memory requirement is 1.5 GB/MDOFs (optimal out-of-core) • Basic I/O Requirement is 15-20 GB/MDOFs • Incore memory requirement is 12-17 GB/MDOFs
ANSYS Iterative PCG Lanczos Solver (LANPCG)
• Basic Memory requirement is 1.5 GB/MDOFs • Basic I/O Requirement is 2.0 GB/MDOFs
DANSYS Iterative PCG Lanczos Solver (LANPCG)
• Basic Memory requirement is 2.2 GB/ MDOFs – For p processors each uses 1.9 GB/MDOFs * 1/p – Add ~300 MB/MDOFs for master process • Basic I/O Requirement is 3.0 GB/MDOFs – Each process consumes 3.0 GB/MDOFs * 1/p file space – File sizes are nearly evenly divided on processors
Iterative Solver Adjustments
• Add 30% for higher-order solid elements • Add 10-50% for higher level of difficulty preconditioners (PCGOPT 2-4) • Save up to 70% from **MSAVE** option by default when applicable – SOLID92 / SOLID95 / SOLID186 / SOLID187 elements – Static Analyses with small deflections (**NLGEOM**,OFF) • Save up to 50% forcing **MSAVE**,ON – SOLID45 / SOLID185 elements (slower runtime) – **NLGEOM**,ON for SOLID45 / SOLID92 / SOLID95 elements (slower runtime)

Table A.9 Obtaining Performance Statistics from ANSYS Solvers

Solver	Performance Stats Source	Expected Results on a Balanced System
Sparse (including complex sparse)	**BCSOPTION**,,,,,,PERFORMANCE command or Jobname.BCS file	1000 – 5500 **Mflops factor rate** 1 CPU 30-70 MB/sec effective I/O rate single drive 150-300 MB/sec effective I/O rate Windows 64-bit RAID0, 4 drives or striped Linux configuration 500-2500 MB/sec effective I/O rate for incore or when system cache is larger than file size
LANB - Block Lanczos	**BCSOPTION**,,,,,,PERFORMANCE command or Jobname.BCS file	Same as sparse above but also add: **Number of factorizations** - 2 is minimum, more factorizations for difficult eigen-problems, many modes, or when frequency range is specified. **Number of block solves** - each block solve reads the LN07 file 3 times. More block solves required for more modes or when block size is reduced due to insufficient memory. **Solve Mflop rate** is not same as I/O rate but good I/O performance will yield solve Mflop rates of 500-1000 Mflops. Slow single drives yield 150 Mflops or less.
PCG	Jobname.PCS - always written by PCG iterative solver	**Total iterations** in hundreds for well conditioned problems. Over 2000 iterations indicates difficult PCG job, slower times expected. **Level of Difficulty** - 1 or 2 typical. Higher level reduces total iterations but increases memory and CPU cost per iteration. **Elements: Assembled** indicates elements are not using **MSAVE**,ON feature. Implicit indicates **MSAVE**,ON elements (reduces memory use for PCG).
LANPCG - PCG Lanczos	Jobname.PCS	Same as PCG above but add: **Number of Load Steps** - 2 - 3 times number of modes desired **Average iterations per load case** - few hundred or less is desired. **Level of Difficulty**: 2-4 best for very large models. 5 uses direct factorization - best only when system can handle factorization cost well.

Table A.10 Summary of Block Lanczos Memory Guidelines

Memory Mode	Guideline
Incore Lanczos runs	• Use only if incore memory < total physical memory of system ("comfortable" incore memory). • Use -lm version if incore memory exceeds 16 GB on a large memory system with more than 16 GB memory available.
Out-of-core Lanczos runs	• Use -m xxx, with xxx = 1 to 1.6 GB/MDOF guideline. • Consider adding RAID0 I/O array to improve I/O performance 3-4X.

General Guidelines:

- Use parallel performance to reduce factorization time, but total parallel speedup is limited by serial I/O and block solves.
- Monitor number of block solves. Expect number of block solves to be less then 2-3X number of modes computed.
- Increase memory for block Lanczos by increasing -m or using **BCSOPTION** command to try and reduce number of block solves.
- Don't use excessive memory for out-of-core runs (limits system caching of I/O to files).
- Don't use all of physical memory just to get an incore factorization (results in sluggish system performance and limits system caching of all other files).

Table A.11 Performance Guidelines for ANSYS and DANSYS Solvers

ANSYS Sparse Solver
• Monitor factorization rate for CPU performance. • Monitor effective I/O rate for I/O performance. • Use incore option (**BCSOPTION**,,INCORE) when system memory is large enough. • Use optimal out-of-core (default) with RAID0 I/O or on large memory multi-user systems.
ANSYS Block Lanczos
• Monitor number of block solves and number of factorizations. • Avoid specifying frequency endpoints to reduce number of factorizations. • Use generous out-of-core memory to increase block size and reduce number of block solves.
DANSYS Sparse Solver
• Incore is best for parallel performance on multi-machine clusters. • Out-of-core is default in Release 11.0 because incore memory requirement is too high for many systems. • DSPARSE solver performance is limited on multicore or single disk cluster systems in out-of-core mode. • DSPARSE out-of-core performance is recommended when each processes has independent disks or on large memory SMP machines with striped multi-disk I/O partitions. • DSPARSE is faster than SMP sparse solver for incore or when each processor has independent local disks.
PCG Solver
• Recommended choice for DANSYS when applicable. • All preconditioner options are parallel. • Reduce level of difficulty to improve parallel scalability for fast converging models. • Increase level of difficulty to reduce PCG iterations for slow converging models.