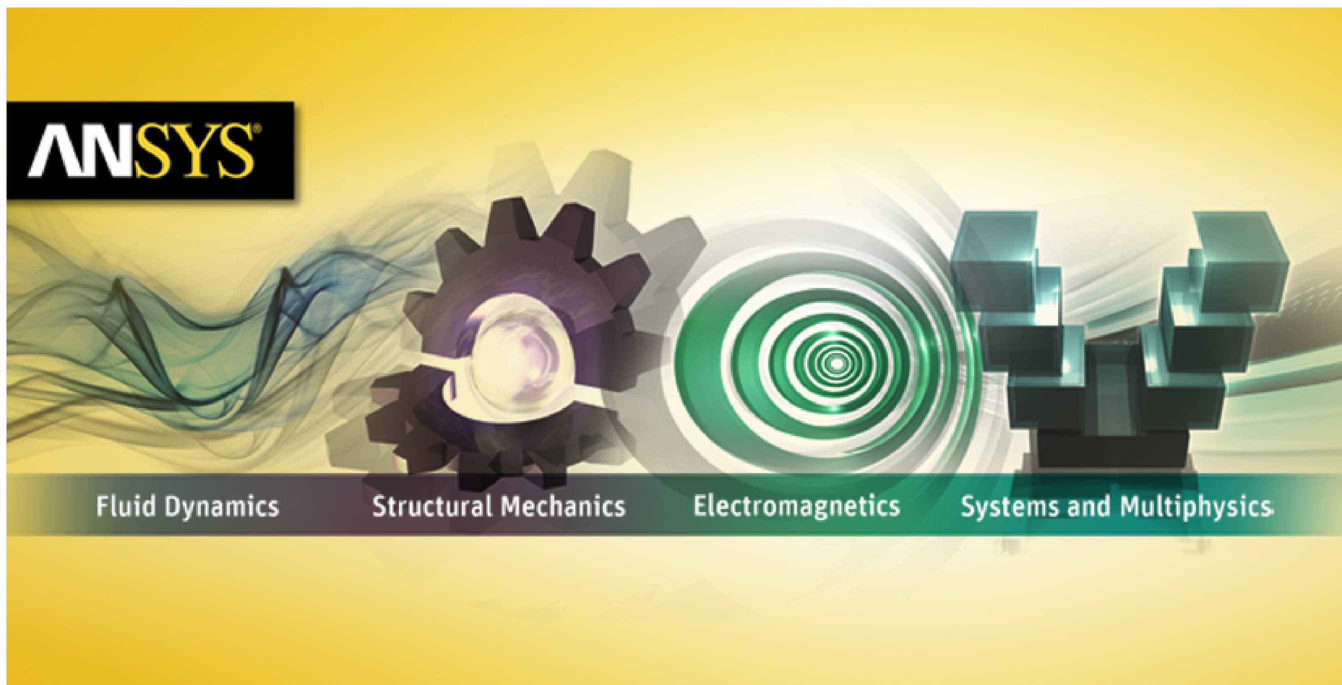


ANSYS Mechanical APDL Performance Guide

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

Release 15.0
November 2013

ANSYS, Inc. is
certified to ISO
9001:2008.

Copyright and Trademark Information

© 2013 SAS IP, Inc. All rights reserved. Unauthorized use, distribution or duplication is prohibited.

ANSYS, ANSYS Workbench, Ansoft, AUTODYN, EKM, Engineering Knowledge Manager, CFX, FLUENT, HFSS and any and all ANSYS, Inc. brand, product, service and feature names, logos and slogans are registered trademarks or trademarks of ANSYS, Inc. or its subsidiaries in the United States or other countries. ICEM CFD is a trademark used by ANSYS, Inc. under license. CFX is a trademark of Sony Corporation in Japan. All other brand, product, service and feature names or trademarks are the property of their respective owners.

Disclaimer Notice

THIS ANSYS SOFTWARE PRODUCT AND PROGRAM DOCUMENTATION INCLUDE TRADE SECRETS AND ARE CONFIDENTIAL AND PROPRIETARY PRODUCTS OF ANSYS, INC., ITS SUBSIDIARIES, OR LICENSORS. The software products and documentation are furnished by ANSYS, Inc., its subsidiaries, or affiliates under a software license agreement that contains provisions concerning non-disclosure, copying, length and nature of use, compliance with exporting laws, warranties, disclaimers, limitations of liability, and remedies, and other provisions. The software products and documentation may be used, disclosed, transferred, or copied only in accordance with the terms and conditions of that software license agreement.

ANSYS, Inc. is certified to ISO 9001:2008.

U.S. Government Rights

For U.S. Government users, except as specifically granted by the ANSYS, Inc. software license agreement, the use, duplication, or disclosure by the United States Government is subject to restrictions stated in the ANSYS, Inc. software license agreement and FAR 12.212 (for non-DOD licenses).

Third-Party Software

See the legal information in the product help files for the complete Legal Notice for ANSYS proprietary software and third-party software. If you are unable to access the Legal Notice, please contact ANSYS, Inc.

Published in the U.S.A.

Table of Contents

1. Introduction 1
- 1.1. A Guide to Using this Document 1

2. Hardware Considerations 3
- 2.1. Hardware Terms and Definitions 3
- 2.2. CPU, Memory, and I/O Balance 5

3. Understanding ANSYS Computing Demands 7
- 3.1. Computational Requirements 7
  - 3.1.1. Shared Memory Parallel vs. Distributed Memory Parallel 7
  - 3.1.2. Parallel Processing 8
  - 3.1.3. Recommended Number of Cores 8
  - 3.1.4. GPU Accelerator Capability 8
- 3.2. Memory Requirements 9
  - 3.2.1. Specifying Memory Allocation 9
  - 3.2.2. Memory Limits on 32-bit Systems 10
  - 3.2.3. Memory Considerations for Parallel Processing 10
- 3.3. I/O Requirements 11
  - 3.3.1. I/O Hardware 11
  - 3.3.2. I/O Considerations for Distributed ANSYS 13

4. ANSYS Memory Usage and Performance 15
- 4.1. Linear Equation Solver Memory Requirements 15
  - 4.1.1. Direct (Sparse) Solver Memory Usage 15
    - 4.1.1.1. Out-of-Core Factorization 17
    - 4.1.1.2. In-core Factorization 17
    - 4.1.1.3. Partial Pivoting 19
  - 4.1.2. Iterative (PCG) Solver Memory Usage 19
  - 4.1.3. Modal (Eigensolvers) Solver Memory Usage 21
    - 4.1.3.1. Block Lanczos Solver 21
    - 4.1.3.2. PCG Lanczos Solver 22
    - 4.1.3.3. Supernode Solver 24

5. Parallel Processing Performance 27
- 5.1. What Is Scalability? 27
- 5.2. Measuring Scalability 27
- 5.3. Hardware Issues for Scalability 28
  - 5.3.1. Multicore Processors 28
  - 5.3.2. Interconnects 28
  - 5.3.3. I/O Configurations 29
    - 5.3.3.1. Single Machine 29
    - 5.3.3.2. Clusters 29
  - 5.3.4. GPUs 30
- 5.4. Software Issues for Scalability 30
  - 5.4.1. ANSYS Program Architecture 30
  - 5.4.2. Distributed ANSYS 31
    - 5.4.2.1. Contact Elements 31
    - 5.4.2.2. Using the Distributed PCG Solver 31
    - 5.4.2.3. Using the Distributed Sparse Solver 31
    - 5.4.2.4. Combining Files 32
  - 5.4.3. GPU Accelerator Capability 32

6. Measuring ANSYS Performance 33
- 6.1. Sparse Solver Performance Output 33
- 6.2. Distributed Sparse Solver Performance Output 35

Release 15.0 - © SAS IP, Inc. All rights reserved. - Contains proprietary and confidential information of ANSYS, Inc. and its subsidiaries and affiliates.

6.3. Block Lanczos Solver Performance Output 37
6.4. PCG Solver Performance Output 39
6.5. PCG Lanczos Solver Performance Output 41
6.6. Supernode Solver Performance Output 46
6.7. Identifying CPU, I/O, and Memory Performance 47
7. Examples and Guidelines 51
7.1. ANSYS Examples 51
7.1.1. SMP Sparse Solver Static Analysis Example 51
7.1.2. Block Lanczos Modal Analysis Example 53
7.1.3. Summary of Lanczos Performance and Guidelines 58
7.2. Distributed ANSYS Examples 59
7.2.1. Distributed ANSYS Memory and I/O Considerations 59
7.2.2. Distributed ANSYS Sparse Solver Example 60
7.2.3. Guidelines for Iterative Solvers in Distributed ANSYS 64
A. Glossary 69
Index 75

Release 15.0 - © SAS IP, Inc. All rights reserved. - Contains proprietary and confidential information of ANSYS, Inc. and its subsidiaries and affiliates.

List of Figures

4.1. In-core vs. Out-of-core Memory Usage for Distributed Memory Sparse Solver 19

Release 15.0 - © SAS IP, Inc. All rights reserved. - Contains proprietary and confidential information of ANSYS, Inc. and its subsidiaries and affiliates.

List of Tables

3.1. Recommended Configuration for I/O Hardware ... 12
4.1. Direct Sparse Solver Memory and Disk Estimates ... 16
4.2. Iterative PCG Solver Memory and Disk Estimates ... 20
4.3. Block Lanczos Eigensolver Memory and Disk Estimates ... 22
4.4. PCG Lanczos Memory and Disk Estimates ... 23
6.1. Obtaining Performance Statistics from ANSYS Solvers ... 48
7.1. Summary of Block Lanczos Memory Guidelines ... 59

Release 15.0 - © SAS IP, Inc. All rights reserved. - Contains proprietary and confidential information of ANSYS, Inc. and its subsidiaries and affiliates.

Release 15.0 - © SAS IP, Inc. All rights reserved. - Contains proprietary and confidential information
of ANSYS, Inc. and its subsidiaries and affiliates.

Chapter 1: Introduction

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:

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

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 Linux operating system issues are covered throughout the guide.

1.1. A Guide to Using this Document

You may choose to read this document from front to back in order to learn more about maximizing ANSYS performance. However, if you are an experienced ANSYS user, you may just need to focus on particular topics that will help you gain insight into performance issues that apply to your particular analysis. The following list of chapter topics may help you to narrow the search for specific information:

- Hardware Considerations (p. 3) gives a quick introduction to hardware terms and definitions used throughout the guide.
- Understanding ANSYS Computing Demands (p. 7) describes ANSYS computing demands for memory, parallel processing, and I/O.
- ANSYS Memory Usage and Performance (p. 15) is a more detailed discussion of memory usage for various ANSYS solver options. Subsections of this chapter allow users to focus on the memory usage details for particular solver choices.
- Parallel Processing Performance (p. 27) describes how you can measure and improve scalability when running Distributed ANSYS.
- Measuring ANSYS Performance (p. 33) describes how to use ANSYS output to measure performance for each of the commonly used solver choices in ANSYS.
- Examples and Guidelines (p. 51) contains detailed examples of performance information obtained from various example runs.
- A glossary at the end of the document defines terms used throughout the guide.

Release 15.0 - © SAS IP, Inc. All rights reserved. - Contains proprietary and confidential information of ANSYS, Inc. and its subsidiaries and affiliates.

Release 15.0 - © SAS IP, Inc. All rights reserved. - Contains proprietary and confidential information
of ANSYS, Inc. and its subsidiaries and affiliates.

Chapter 2: Hardware Considerations

This chapter provides a brief discussion of hardware terms and definitions used throughout this guide. The following topics are covered:

2.1. Hardware Terms and Definitions
2.2. CPU, Memory, and I/O Balance

2.1. Hardware Terms and Definitions

This section discusses terms and definitions that are commonly used to describe current hardware capabilities.

CPUs and Cores

The advent of multicore processors has introduced some ambiguity about the definition of a CPU. Historically, the CPU was the central processing unit of a computer. However, with multicore CPUs each core is really an independently functioning processor. Each multicore CPU contains 2 or more cores and is, therefore, a parallel computer competing with the resources for memory and I/O on a single motherboard.

The ambiguity of CPUs and cores often occurs when describing parallel algorithms or parallel runs. In the context of an algorithm, CPU almost always refers to a single task on a single processor. In this document we will use core rather than CPU to identify independent processes that run on a single CPU core. CPU will be reserved for describing the socket configuration. For example, a typical configuration today contains two CPU sockets on a single motherboard with 4 or 8 cores per socket. Such a configuration could support a parallel Distributed ANSYS run of up to 16 cores. We will describe this as a *16-core* run, not a 16-processor run.

GPUs

While graphics processing units (GPUs) have been around for many years, only recently have they begun to be used to perform general purpose computations. GPUs offer a highly parallel design which often includes hundreds of compute units and have their own dedicated physical memory. Certain high-end graphics cards, the ones with the most amount of compute units and memory, can be used to accelerate the computations performed during an ANSYS simulation. In this document, we will use the term GPU to refer to these high-end cards that can be used as *accelerators* to speedup certain portions of an ANSYS simulation.

Threads and MPI Processes

Two modes of parallel processing are supported and used throughout ANSYS simulations. Details of parallel processing in ANSYS are described in a later chapter, but the two modes introduce another ambiguity in describing parallel processing. For the shared memory implementation of ANSYS, one instance of the ANSYS executable (i.e., one ANSYS process) spawns multiple threads for parallel regions. However, for the distributed memory implementation of ANSYS, multiple instances of the ANSYS executable run as separate MPI tasks or processes.

Release 15.0 - © SAS IP, Inc. All rights reserved. - Contains proprietary and confidential information of ANSYS, Inc. and its subsidiaries and affiliates.

In this document, “threads” refers to the shared memory tasks that ANSYS uses when running in parallel under a single ANSYS process. “MPI processes” refers to the distributed memory tasks that Distributed ANSYS uses when running in parallel. MPI processes serve the same function as threads but are multiple ANSYS processes running simultaneously that can communicate through MPI software.

Memory Vocabulary

Two common terms used to describe computer memory are physical and virtual memory. Physical memory is essentially the total amount of RAM (Random Access Memory) available. Virtual memory is an extension of physical memory that is actually reserved on disk storage. It allows applications to extend the amount of memory address space available at the cost of speed since addressing physical memory is much faster than accessing virtual (or disk) memory. The appropriate use of virtual memory is described in later chapters.

I/O Vocabulary

I/O performance is an important component of computer systems for ANSYS users. Advances in desktop systems have made high performance I/O available and affordable to all users. A key term used to describe multidisc, high performance systems is RAID (Redundant Array of Independent Disks). RAID arrays are common in computing environments, but have many different uses and can be the source of yet another ambiguity.

For many systems, RAID configurations are used to provide duplicate files systems that maintain a mirror image of every file (hence, the word redundant). This configuration, normally called RAID1, does not increase I/O performance, but often increases the time to complete I/O requests. An alternate configuration, called RAID0, uses multiple physical disks in a single striped file system to increase read and write performance by splitting I/O requests simultaneously across the multiple drives. This is the RAID configuration recommended for optimal ANSYS I/O performance. Other RAID setups use a parity disk to achieve redundant storage (RAID5) or to add redundant storage as well as file striping (RAID10). RAID5 and RAID10 are often used on much larger I/O systems.

Interconnects

Distributed memory parallel processing relies on message passing hardware and software to communicate between MPI processes. The hardware components on a shared memory system are minimal, requiring only a software layer to implement message passing to and from shared memory. For multi-machine or multi-node clusters with separate physical memory, several hardware and software components are required. Usually, each compute node of a cluster contains an adapter card that supports one of several standard interconnects (for example, GigE, Myrinet, Infiniband). The cards are connected to high speed switches using cables. Each interconnect system requires a supporting software library, often referred to as the fabric layer. The importance of interconnect hardware in clusters is described in later chapters. It is a key component of cluster performance and cost, particularly on large systems.

Within the major categories of GigE, Myrinet, and Infiniband, new advances can create incompatibilities with application codes. It is important to make sure that a system that uses a given interconnect with a given software fabric is compatible with ANSYS. Details of the ANSYS requirements for hardware interconnects are found in the *Parallel Processing Guide*.

The performance terms used to discuss interconnect speed are latency and bandwidth. Latency is the measured time to send a message of zero length from one MPI process to another (i.e., overhead). It is generally expressed as a time, usually in micro seconds. Bandwidth is the rate (MB/sec) at which larger messages can be passed from one MPI process to another (i.e., throughput). Both latency and bandwidth are important considerations in the performance of Distributed ANSYS.

Release 15.0 - © SAS IP, Inc. All rights reserved. - Contains proprietary and confidential information of ANSYS, Inc. and its subsidiaries and affiliates.

Many switch and interconnect vendors describe bandwidth using Gb or Mb units. Gb stands for Gigabits, and Mb stands for Megabits. Do not confuse these terms with GB (GigaBytes) and MB (MegaBytes). Since a byte is 8 bits, it is important to keep the units straight when making comparisons. Throughout this guide we consistently use GB and MB units for both I/O and communication rates.

2.2. CPU, Memory, and I/O Balance

In order to achieve good overall performance, it is imperative to have the correct balance of processors, memory, and disk I/O. The CPU speed is an obvious factor of performance. However, the other two factors—memory and disk I/O—are not always so obvious. This section discusses the importance of each major hardware component for achieving optimal performance with ANSYS.

Processors

Virtually all processors now have multiple cores and operate at several GHz (Giga (10^9) Hertz) frequencies. Processors are now capable of sustaining compute rates of 5 to 20 Gflops (Giga (10^9) floating point operations per second) per core in ANSYS equation solvers. As processors have increased their computational performance, the emphasis on memory and I/O capacity and speed has become even more important in order to achieve peak performance.

Memory

Large amounts of memory can not only extend the model size that can be simulated on a given machine, but also plays a much more important role in achieving high performance. In most cases, a system with larger amounts of memory will outperform a smaller memory system, even when the smaller memory system uses faster processors. This is because larger amounts of memory can be used to avoid doing costly I/O operations. First, some equation solvers will use the additional memory to avoid doing I/O to disk. Second, both Linux and Windows systems now have automatic, effective system buffer caching of file I/O. The operating systems automatically cache files in memory when enough physical memory is available to do so. In other words, whenever the physical memory available on a system exceeds the size of the files being read and written, I/O rates are determined by memory copy speed rather than the far slower disk I/O rates. Memory buffered I/O is automatic on most systems and can reduce I/O time by more than 10X.

Faster memory access, or memory bandwidth, also contributes to achieving optimal performance. Faster memory bandwidths allow processors to achieve closer-to-peak compute performance by feeding data to the processor at faster speeds. Memory bandwidth has become more important as systems with more processor cores are produced. Each processor core will require additional memory bandwidth. So both the speed and the total memory bandwidth available on a system are important factors for achieving optimal performance.

I/O

I/O to the hard drive is the third component of a balanced system. Well balanced systems can extend the size of models that can be solved if they use properly configured I/O components. If ANSYS simulations are able to run with all file I/O cached by a large amount of physical memory, then disk resources can be concentrated on storage more than performance.

A good rule of thumb is to have 10 times more disk space than physical memory available for your simulation. With today's large memory systems, this can easily mean disk storage requirements of 500 GB to 1 TB (TeraByte). However, if you use physical disk storage routinely to solve large models, a high performance file system can make a huge difference in simulation time.

Release 15.0 - © SAS IP, Inc. All rights reserved. - Contains proprietary and confidential information of ANSYS, Inc. and its subsidiaries and affiliates.

A high performance file system could consist of solid state drives (SSDs) or conventional spinning hard disk drives (HDDs). SSDs typically offer superior performance over HDDs, but have other factors to consider, such as cost and mean-time-to-failure. Like HDDs, multiple SSDs can be combined together in a RAID0 array. Maximum performance with a RAID array is obtained when the ANSYS simulation is run on a RAID0 array of 4 or more disks that is a separate disk partition from other system file activity. For example, on a Windows desktop the drive containing the operating system files should not be in the RAID0 configuration. This is the optimal recommended configuration. Many hardware companies do not currently configure separate RAID0 arrays in their advertised configurations. Even so, a standard RAID0 configuration is still faster for ANSYS simulations than a single drive.

Release 15.0 - © SAS IP, Inc. All rights reserved. - Contains proprietary and confidential information of ANSYS, Inc. and its subsidiaries and affiliates.

Chapter 3: Understanding ANSYS Computing Demands

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:

3.1. Computational Requirements
3.2. Memory Requirements
3.3. I/O Requirements

3.1. Computational Requirements

ANSYS uses the latest compilers and math libraries in order to achieve maximum per-core performance on virtually all processors that it supports. However, many simulations are still compute-bottlenecked. Therefore, the best way to speed up a simulation is often to utilize more processor cores. This is done with parallel processing. Another method to speed up simulations is to use a GPU card to accelerate some computations during the simulation.

Detailed information on parallel processing can be found in the *Parallel Processing Guide*. For the purpose of this discussion, some basic details are provided in the following sections.

3.1.1. Shared Memory Parallel vs. Distributed Memory Parallel

Shared memory parallel (SMP) is distinguished from *distributed memory parallel* (DMP) by a different memory model. SMP and DMP can refer to both hardware and software offerings. In terms of hardware, SMP systems share a single global memory image that is addressable by multiple processors. DMP systems, often referred to as clusters, involve multiple machines (i.e., compute nodes) connected together on a network, with each machine having its own memory address space. Communication between machines is handled by interconnects (e.g., Gigabit Ethernet, Myrinet, Infiniband).

In terms of software, the shared memory parallel version of ANSYS refers to running ANSYS across multiple cores on an SMP system. The distributed memory parallel version of ANSYS (Distributed ANSYS) refers to running ANSYS across multiple processors on SMP systems or DMP systems.

Distributed memory parallel processing assumes that the physical memory for each process is separate from all other processes. This type of parallel processing requires some form of message passing software to exchange data between the cores. The prevalent software used for this communication is called MPI (Message Passing Interface). MPI software uses a standard set of routines to send and receive messages and synchronize processes. A major attraction of the DMP model is that very large parallel systems can be built using commodity-priced components. The disadvantage of the DMP model is that it requires users to deal with the additional complications for both software and system setup. However, once operational the DMP model often obtains better parallel efficiency than the SMP model.

Release 15.0 - © SAS IP, Inc. All rights reserved. - Contains proprietary and confidential information
of ANSYS, Inc. and its subsidiaries and affiliates.

3.1.2. Parallel Processing

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 cores 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.

Distributed ANSYS parallelizes the entire ANSYS solution phase, including the three steps listed above. However, the maximum speedup obtained by Distributed ANSYS is limited by similar issues as the SMP version of ANSYS (I/O, memory bandwidth), as well as how well the computations are balanced among the processes, the speed of messages being passed, and the amount of work that cannot be done in a parallel manner.

It is important to note that the SMP version of ANSYS can only run on configurations that share a common address space; it cannot run across separate machines or even across nodes within a cluster. However, Distributed ANSYS can run using multiple cores on a single machine (SMP hardware), and it can be run across multiple machines (i.e., a cluster) using one or more cores on each of those machines (DMP hardware).

You may choose SMP or DMP processing using 2 cores with the standard ANSYS license. To achieve additional benefit from parallel processing, you must acquire additional ANSYS Mechanical HPC licenses.

3.1.3. Recommended Number of Cores

For SMP systems, ANSYS can effectively use up to 4 cores in most cases. For very large jobs, you may see reduced wall clock time when using up to 8 cores. In most cases, however, no more than 8 cores should be used for a single ANSYS job using SMP parallel.

Distributed ANSYS performance typically exceeds SMP ANSYS. The speedup achieved with Distributed ANSYS, compared to that achieved with SMP ANSYS, improves as more cores are used. Distributed ANSYS has been run using as many as 1024 cores, but is usually most efficient with up to 32 cores.

In summary, nearly all ANSYS analyses will see reduced time to solution using parallel processing. While speedups vary due to many factors, you should expect to see the best time to solution results when using Distributed ANSYS on properly configured systems having 8-32 cores.

3.1.4. GPU Accelerator Capability

GPU hardware can be used to help reduce the overall time to solution in ANSYS by off-loading some of the major computations (required by certain equation solvers) from the CPU(s) to the GPU. These computations are often executed much faster using the highly parallel architecture found with GPUs.

The use of GPU hardware is meant to be in addition to the existing CPU core(s), not a replacement for CPUs. The CPU core(s) will continue to be used for all other computations in and around the equation

Release 15.0 - © SAS IP, Inc. All rights reserved. - Contains proprietary and confidential information of ANSYS, Inc. and its subsidiaries and affiliates.

solvers. This includes the use of any shared memory parallel processing or distributed memory parallel processing by means of multiple CPU cores. The main goal of the GPU accelerator capability is to take advantage of the GPU hardware to accelerate the speed of the solver computations and, therefore, reduce the time required to complete a simulation in ANSYS.

GPUs have varying amounts of physical memory available for the ANSYS simulation to use. The amount of available memory can limit the speedups achieved. When the memory required to perform the solver computations exceeds the available memory on the GPU, the use of the GPU is temporarily de-activated for those computations and the CPU core(s) are used instead.

The speedups achieved when using GPUs will vary widely depending on the specific CPU and GPU hardware being used, as well as the simulation characteristics. When older (and therefore typically slower) CPU cores are used, the GPU speedups will be greater. Conversely, when newer (and therefore typically faster) CPUs are used, the performance of the newer CPUs will make the GPU speedups less. Also, the speedups achieved when using GPUs will depend mainly on the analysis type, element types, equation solver, and model size (number of DOFs). However, it all relates to how much time is spent performing computations on the GPU vs. on the CPU. The more computations performed on the GPU, the more opportunity for greater speedups. When using the sparse direct solver, the use of bulkier 3D models and/or higher-order elements generally results in more solver computations off-loaded to the GPU. In the PCG iterative solver, the use of lower Lev_Diff values (see the **PCGOPT** command) results in more solver computations off-loaded to the GPU.

3.2. Memory Requirements

Memory requirements within ANSYS are driven primarily by the requirement of solving large systems of equations. Details of solver memory requirements are discussed in a later section. Additional memory demands can come from meshing large components and from other pre- and postprocessing 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 when large amounts of physical memory are available on a system. This hidden benefit is the ability of operating systems 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 equation solver while also caching the large files generated during the simulation.

3.2.1. Specifying Memory Allocation

ANSYS memory is divided into two blocks: the *database* space that holds 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) can be an important component of achieving optimal performance with some of the solver options.

In general, specifying a total workspace (-m) or database memory (-db) setting at startup is no longer necessary. Both the scratch space and database space (64-bit systems only) grow dynamically in ANSYS, provided the memory is available when the ANSYS memory manager tries to allocate additional memory from the operating system. If the database space is unavailable (or forced not to grow dynamically via a negative -db value), ANSYS automatically uses a disk file (`.PAGE`) to spill the database to disk. See Memory Management and Configuration in the *Basic Analysis Guide* for additional details on ANSYS memory management.

Release 15.0 - © SAS IP, Inc. All rights reserved. - Contains proprietary and confidential information of ANSYS, Inc. and its subsidiaries and affiliates.

3.2.2. 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-bit.

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:

```
ansys150 -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. Users who experience memory limitations on a Windows 32-bit system should seriously consider upgrading to Windows 64-bit. However, for users who primarily solve smaller models that run easily within the 1 to 1.5 GB of available memory, Windows 32-bit systems can deliver HPC performance that is on par with the largest systems available today.

3.2.3. 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.

Distributed ANSYS memory usage requires more explanation. When running Distributed ANSYS using *n* cores, *n* Distributed ANSYS processes are started. The first of these processes is often referred to as the master, host, or rank-0 process, while the remaining *n-1* processes are often referred to as the slave processes.

In Distributed ANSYS, the master MPI process always does more work than the remaining processes, and therefore always requires more memory than the slave processes. The master process reads the entire ANSYS input file and does all of the pre- and postprocessing, as well as the initial model decomposition required for Distributed ANSYS. 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.

When running on a cluster, the memory available on the compute node containing the master process will typically determine the maximum size of problem that Distributed ANSYS can solve. Generally, the compute node that contains the master process should have twice as much memory as all other machines used for the run. As an example, a cluster of 8 compute 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

Release 15.0 - © SAS IP, Inc. All rights reserved. - Contains proprietary and confidential information of ANSYS, Inc. and its subsidiaries and affiliates.

is the same. Upgrading the master compute node to 8 GB, however, should allow the cluster to solve a similar-sized problem as the 32 GB SMP system.

3.3. I/O Requirements

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 sparse direct equation solver 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 or more from disk files that are tens of GB in size or larger. At a typical disk I/O rate of 50-100 MB/sec on many desktop systems, this I/O demand can add hours of elapsed time to a simulation. Another expensive I/O demand in ANSYS is saving results for multiple step (time step or load 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 large model, or for a nonlinear or transient analysis with many solutions.

This section will discuss ways to minimize the I/O time in ANSYS. Important breakthroughs in desktop I/O performance that have been added to ANSYS will be described later in this document. To understand recent improvements in ANSYS and Distributed ANSYS I/O, a discussion of I/O hardware follows.

3.3.1. I/O Hardware

I/O capacity and speed are important parts of a well balanced 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 always able to hide the I/O cost.

Many desktop systems today have very large capacity hard drives 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. ANSYS I/O requires a sustained I/O stream to write or read files that can be many GBytes in length.

Solid state drives (SSDs) are becoming more popular as the technology improves and costs decrease. SSDs offer significantly reduced seek times while maintaining good transfer rates when compared to the latest hard disk drives. This can lead to dramatic performance improvements when lots of I/O is performed, such as when running the sparse direct solver in an out-of-core memory mode. Factors such as cost and mean-time-failure must also be considered when choosing between SSDs and conventional hard disk drives.

The key to obtaining outstanding performance 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. 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, using conventional hard drives and over 500 MB/sec using SSDs.

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

Release 15.0 - © SAS IP, Inc. All rights reserved. - Contains proprietary and confidential information of ANSYS, Inc. and its subsidiaries and affiliates.

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.

Another key bottleneck for I/O on many systems comes from using centralized I/O resources that share a relatively slow 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 and large capacity to multiple compute servers, but such a configuration requires expensive high-speed interconnects to supply each compute server independently for simultaneously running jobs. Another common pitfall with centralized I/O resources comes when the central I/O system is configured for redundancy and data integrity. While this approach is desirable for transaction type processing, it will severely degrade high performance I/O in most cases. If central I/O resources are to be used for ANSYS simulations, a high performance configuration is essential.

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 64-bit operating 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 (many 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, often with no warning.

The bottom line for minimizing I/O times in ANSYS and Distributed ANSYS is to use as much memory as possible to minimize the actual I/O required and to use multiple disk RAID arrays in a separate work directory for the ANSYS working directory. Fast I/O is no longer a high cost addition if properly configured and understood. The following is a summary of I/O configuration recommendations for ANSYS users.

Table 3.1: Recommended Configuration for I/O Hardware

Recommended Configuration for I/O Hardware
• 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 RAID0 across the physical drives.
• Consider the use of solid state drive(s) for maximum performance.
• Size of ANSYS working directory should preferably be <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. Swap space need not equal memory size on very large memory systems (that is, swap space should be less than 32 GB). Increasing swap space is effective when there is a short-time, high-memory requirement such as meshing a very large component.

Release 15.0 - © SAS IP, Inc. All rights reserved. - Contains proprietary and confidential information of ANSYS, Inc. and its subsidiaries and affiliates.

3.3.2. I/O Considerations for Distributed ANSYS

ANSYS I/O in the sparse direct solver has been optimized on Windows systems to take full advantage of multiple drive RAID0 arrays. An inexpensive investment for a RAID0 array on a desktop system can yield significant gains in ANSYS performance. However, for desktop systems (and many cluster configurations) it is important to understand that many cores share the same disk resources. Therefore, obtaining fast I/O performance in applications such as ANSYS is often not as simple as adding a fast RAID0 configuration.

For SMP ANSYS runs, there is only one set of ANSYS files active for a given simulation. However, for a Distributed ANSYS simulation, each core maintains its own set of ANSYS files. This places an ever greater demand on the I/O resources for a system as the number of cores used by ANSYS is increased. For this reason, Distributed ANSYS performs best when solver I/O can be eliminated altogether or when multiple nodes are used for parallel runs, each with a separate local I/O resource. If Distributed ANSYS is run on a single machine and lots of I/O must be done due to a lack of physical memory, then solid state drives (SSDs) may be very beneficial for achieving optimal performance. The significantly reduced seek time of SSDs can help to reduce the cost of having multiple processors each writing/reading their own set of files. Conventional hard drives will have a huge sequential bottleneck as the disk head moves to the location of each processor's file(s). This bottleneck is virtually eliminated using SSDs, thus making optimal performance possible.

Release 15.0 - © SAS IP, Inc. All rights reserved. - Contains proprietary and confidential information of ANSYS, Inc. and its subsidiaries and affiliates.

Release 15.0 - © SAS IP, Inc. All rights reserved. - Contains proprietary and confidential information
of ANSYS, Inc. and its subsidiaries and affiliates.

Chapter 4: ANSYS Memory Usage and Performance

This chapter explains memory usage in ANSYS and gives recommendations on how to manage memory in order to maximize ANSYS 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 shared-memory and distributed-memory solver implementations, are described and summarized in Table 4.1: Direct Sparse Solver Memory and Disk Estimates (p. 16) and Table 4.2: Iterative PCG Solver Memory and Disk Estimates (p. 20). Information on commonly used eigensolvers is also provided in this chapter.

4.1. 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.

All of the solvers covered in this chapter have heuristics which automatically select certain defaults in an attempt to optimize performance for a given set of hardware and model conditions. For the majority of analyses, the best options are chosen. However, in some cases performance can be improved by understanding how the solvers work, the resource requirements for your particular analysis, and the hardware resources that are available to the ANSYS program. Each of the equation solvers discussed have an option command(s) which can be used to control the behavior, and ultimately the performance, of the solver.

The following topics are covered:

4.1.1. Direct (Sparse) Solver Memory Usage
4.1.2. Iterative (PCG) Solver Memory Usage
4.1.3. Modal (Eigensolvers) Solver Memory Usage

4.1.1. Direct (Sparse) Solver Memory Usage

The sparse solver in ANSYS is the default solver for virtually all analyses. It is the most robust solver in ANSYS, but it is also compute- 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. It is important to understand that the solver's mode of operation 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, 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 matrices created in ANSYS 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 calculation of this factor is computationally intensive. In contrast, the solution of the triangular systems is I/O or memory access-dominated with few computations required.

Release 15.0 - © SAS IP, Inc. All rights reserved. - Contains proprietary and confidential information of ANSYS, Inc. and its subsidiaries and affiliates.

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 shared memory version of the sparse solver in ANSYS is not the same as the distributed memory version of the sparse solver in Distributed ANSYS. While the fundamental steps of these solvers are the same, they are actually two independent solvers, and there are subtle differences in their modes of operation. These differences will be explained in the following sections. Table 4.1: Direct Sparse Solver Memory and Disk Estimates (p. 16) summarizes the direct solver memory requirements.

Table 4.1: Direct Sparse Solver Memory and Disk Estimates

Memory Mode	Memory Usage Estimate	I/0 Files Size Estimate
Sparse Direct Solver (Shared Memory)		
Out-of-core	1 GB/MDOFs	10 GB/MDOFs
In-core	10 GB/MDOFs	1 GB/MDOFs 10 GB/MDOFs if workspace is saved to `Jobname.LN22`
Sparse Direct Solver (Distributed Memory, Using p Cores)		
Out-of-core	1 GB/MDOFs on master node 0.7 GB/MDOFs on all other nodes	10 GB/MDOFs * 1/p Matrix factor is stored on disk evenly on p cores
In-core	10 GB/MDOFs * 1/p Matrix factor is stored in memory evenly in-core on p cores. Additional 1.5 GB/MODFs required on master node to store input matrix.	1 GB/MDOFs * 1/p 10 GB/MDOFs * 1/p if workspace is saved to `Jobname.DSPsymb`
Comments: • By default, smaller jobs typically run in the in-core memory mode, while larger jobs run in the out-of-core memory mode. • Double memory estimates for non-symmetric systems. • Double memory estimates for complex valued systems. • Add 30 percent to out-of-core memory for 3-D models with higher-order elements.		

Release 15.0 - © SAS IP, Inc. All rights reserved. - Contains proprietary and confidential information of ANSYS, Inc. and its subsidiaries and affiliates.

- Subtract 40 percent to out-of-core memory for 2-D or beam/shell element dominated models.
- Add 50 -100 percent to file and memory size for in-core memory for 3-D models with higher-order elements.
- Subtract 50 percent to file and memory size for in-core memory for 2-D or beam/shell element dominated models.

4.1.1.1. 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 shared memory version of the sparse solver in ANSYS. The first of these modes is a minimum memory mode that is highly dominated by I/O. However, this mode is rarely used and should be avoided whenever possible. The minimum memory mode occurs whenever the amount of memory allocated for the sparse solver is smaller than the largest front processed by the 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 sufficient memory is not 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 is most beneficial when trying to run a large analysis on a machine that has limited memory. This is especially true with models that have very large fronts, either due to the use of constraint equations that relate 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 the minimum memory mode is often 10 times greater than the size of the entire matrix factor file. (Note that this minimum memory mode does not exist for the distributed memory sparse solver.)

Fortunately, if sufficient memory is available for the assembly process, it is almost always more than enough to run the sparse solver factorization in optimal out-of-core mode. This mode uses some additional memory to make sure that the largest of all fronts can be held completely in memory. This approach avoids the excessive I/O done in the minimum memory mode, but still writes the factored matrix to disk; thus, it 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. For larger jobs, the program will typically run the sparse solver using optimal out-of-core memory mode (by default) unless a specific memory mode is defined.

The distributed memory sparse solver can be run in the optimal out-of-core mode but not the minimum memory mode. It is important to note that when running this solver in out-of-core mode, the additional memory allocated to make sure each individual front is computed in memory is allocated on all processes. Therefore, as more distributed processes are used (that is, the solver is used on more cores) the solver's memory usage for each process is not decreasing, but rather staying roughly constant, and the total sum of memory used by all processes is actually increasing (see Figure 4.1: In-core vs. Out-of-core Memory Usage for Distributed Memory Sparse Solver (p. 19)). Keep in mind, however, that the computations do scale for this memory mode as more cores are used.

4.1.1.2. In-core Factorization

In-core 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, larger memory systems are commonplace today, and users of these systems will benefit from in-core factorization. A model with 1 million DOFs can, in many cases, be factored using 10 GB of memory—easily achieved on desktop systems

Release 15.0 - © SAS IP, Inc. All rights reserved. - Contains proprietary and confidential information of ANSYS, Inc. and its subsidiaries and affiliates.

with 16 GB of memory. Users can run in-core using several different methods. The simplest way to set up an in-core run is to use the **BCSOPTION**,,INCORE command (or **DSPOPTION**,,INCORE for the distributed memory sparse solver). This option tells the sparse solver to try allocating a block of memory sufficient to run using the in-core memory mode 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 in-core performance with the sparse solver is to start the sparse solver with enough memory to run in-core. Users can start ANSYS with an initial large -m allocation (see Specifying Memory Allocation (p. 9)) such that the largest block available when the sparse solver begins is large enough to run the solver using the in-core memory mode. This method will typically obtain enough memory to run the solver factorization step with a lower peak memory usage than the simpler method described above, but it requires prior knowledge of how much memory to allocate in order to run the sparse solver using the in-core memory mode.

The in-core factorization should be used only when the computer system has enough memory to easily factor the matrix in-core. *Users should avoid using all of the available system memory or extending into virtual memory to obtain an in-core factorization*. However, users who have long-running simulations should understand how to use the in-core factorization to improve elapsed time performance.

The **BCSOPTION** command controls the shared-memory 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 allocated for solver =              1536.42 MB
Memory required for in-core =             10391.28 MB
Optimal memory required for out-of-core =  1191.67 MB
Minimum memory required for out-of-core =   204.12 MB
```

This sparse solver run required 10391 MB to run in-core, 1192 MB to run in optimal out-of-core mode, and just 204 MB to run in minimum out-of-core mode. "Memory allocated for solver" indicates that the amount of memory used for this run was just above the optimal out-of-core memory requirement.

The **DSPOPTION** command controls the distributed memory sparse solver memory modes and also enables performance debug summaries. Similar memory usage statistics for this solver are also printed in the output file for each Distributed ANSYS process. When running the distributed sparse solver using multiple cores on a single node or when running on a cluster with a slow I/O configuration, using the in-core mode can significantly improve the overall solver performance as the costly I/O time is avoided.

The memory required per core to run in optimal out-of-core mode approaches a constant value as the number of cores increases because each core in the distributed sparse solver has to store a minimum amount of information to carry out factorization in the optimal manner. The more cores that are used, the more total memory that is needed (it increases slightly at 32 or more cores) for optimal out-of-core performance.

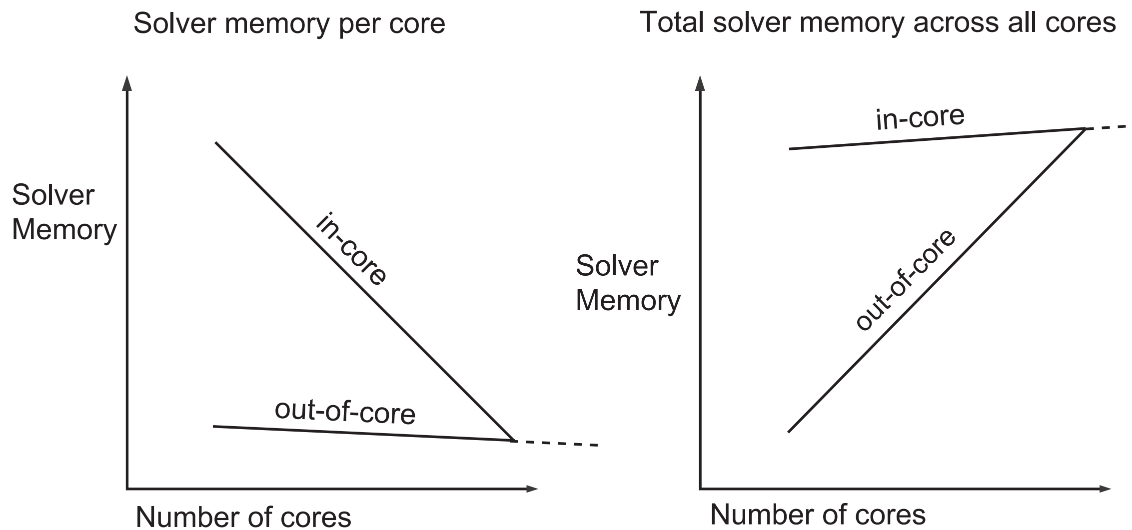
In contrast to the optimal out-of-core mode, the memory required per core to run in the in-core mode decreases as more processes are used with the distributed memory sparse solver (see the left-hand side of Figure 4.1: In-core vs. Out-of-core Memory Usage for Distributed Memory Sparse Solver (p. 19)). This is because the portion of total matrices stored and factorized in one core is getting smaller and smaller. The total memory needed will increase slightly as the number of cores increases.

At some point, as the number of processes increases (usually between 8 and 32), the total memory usage for these two modes approaches the same value (see the right-hand side of Figure 4.1: In-core vs. Out-

Release 15.0 - © SAS IP, Inc. All rights reserved. - Contains proprietary and confidential information of ANSYS, Inc. and its subsidiaries and affiliates.

of-core Memory Usage for Distributed Memory Sparse Solver (p. 19)). When the out-of-core mode memory requirement matches the in-core requirement, the solver automatically runs in-core. This is an important effect; it shows that when a job is spread out across enough machines, the distributed memory sparse solver can effectively use the memory of the cluster to automatically run a very large job in-core.

Figure 4.1: In-core vs. Out-of-core Memory Usage for Distributed Memory Sparse Solver



4.1.1.3. Partial Pivoting

Sparse solver partial pivoting is an important detail that may inhibit in-core factorization. Pivoting in direct 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, pure 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 in-core memory requirement cannot be accurately computed. As a result, it is generally recommended that pivoting-enabled factorizations in ANSYS be out-of-core.

4.1.2. 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 requires a complete factorization of the as-

Release 15.0 - © SAS IP, Inc. All rights reserved. - Contains proprietary and confidential information of ANSYS, Inc. and its subsidiaries and affiliates.

sembled global matrix. This last option (which is discussed in PCG Lanczos Solver (p. 22)) 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 ANSYS automatically selects a high level of difficulty and the user is running on a system with limited memory, it may be necessary to reduce memory requirements by manually specifying a lower level of difficulty (via the **PCGOPT** command). This is because peak memory usage for the PCG solvers often occurs during preconditioner construction.

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 SOLID186 and/or SOLID187 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 SOLID186 elements using reduced integration, or by SOLID187 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 DMP version of the iterative solver is higher than the corresponding SMP version due to some duplicated data structures required on each process for the DMP version. However, the total memory requirement scales across the processes so that memory use per process reduces as the number of processes increases. The preconditioner requires an additional data structure that is only stored and used by the master process, so the memory required for the master process is larger than all other processes. Table 4.2: Iterative PCG Solver Memory and Disk Estimates (p. 20) summarizes the memory requirements for iterative solvers.

The table shows that for Distributed ANSYS running very large models, the most significant term becomes the 300 MB/MDOFs requirement for the master process. This term does not scale (reduce) as more cores are used. A 10 MDOFs model using the iterative solver would require 3 GB of memory for this part of PCG solver memory, in addition to 12 GB distributed evenly across the nodes in the cluster. A 100 MDOFs model would require 30 GB of memory in addition to 120 GB of memory divided evenly among the nodes of the cluster.

Table 4.2: Iterative PCG Solver Memory and Disk Estimates

PCG Solver Memory and Disk Estimates (Shared Memory)
• Basic Memory requirement is 1 GB/MDOFs • Basic I/O Requirement is 1.5 GB/MDOFs
PCG Solver Memory and Disk Estimates (Distributed Memory, Using p Cores)

• Basic Memory requirement is 1.5 GB/MDOFs
  – Each process uses 1.2 GB/MDOFs * 1/p
  – Add ~300 MB/MDOFs for master process
• Basic I/O Requirement is 2.0 GB/MDOFs

Release 15.0 - © SAS IP, Inc. All rights reserved. - Contains proprietary and confidential information of ANSYS, Inc. and its subsidiaries and affiliates.

- Each process consumes 2 GB/MDOFs * 1/p file space
- File sizes are nearly evenly divided on cores

Comments:

- Add 30 percent to memory requirement for higher-order solid elements
- Add 10-50 percent to memory requirement for higher level of difficulty preconditioners (PCGOPT 2-4)
- Save up to 70 percent for memory requirement from **MSAVE** option by default, when applicable
  - SOLID186 / SOLID187 elements
  - Static analyses with small deflections (**NLGEOM**,OFF)
- Save up to 50 percent for memory requirement forcing **MSAVE**,ON
  - SOLID185 elements (at the expense of possibly longer runtime)
  - **NLGEOM**,ON for SOLID185 / SOLID186 / SOLID187 elements (at the expense of possibly longer runtime)

4.1.3. 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 three eigensolvers for modal analyses of undamped systems: the sparse solver-based Block Lanczos solver, the PCG Lanczos solver, and the Supernode solver.

The memory requirements for the two Lanczos-based eigensolvers 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.

4.1.3.1. Block Lanczos Solver

The Block Lanczos solver (**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 in-core memory mode is used) and one in-memory block of vectors. The larger the *BlockSize* (input on the **MODOPT** command), the fewer block solves are required, reducing the I/O cost for the solves.

If the memory allocated for the solver is below recommended memory in a Block Lanczos run, the block size used internally for the Lanczos iterations will be automatically reduced. Smaller block sizes will require more block solves, the most expensive part of the Lanczos algorithm for I/O performance. Typically, the default block size of 8 is optimal. On machines with limited physical memory where the I/O cost in

Release 15.0 - © SAS IP, Inc. All rights reserved. - Contains proprietary and confidential information of ANSYS, Inc. and its subsidiaries and affiliates.

Block Lanczos is very high (for example, machines without enough physical memory to run the Block Lanczos eigensolver using the in-core memory mode), forcing a larger *BlockSize* (such as 12 or 16) on the **MODOPT** command can reduce the amount of I/O and, thus, improve overall performance.

Finally, multiple matrix factorizations may be required for a Block Lanczos run. (See the table below for Block Lanczos memory requirements.) 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 the rate of convergence for the requested modes and the cost of each Lanczos iteration. This means that performance characteristics can change when hardware is changed, when the memory mode is changed from out-of-core to in-core (or vice versa), or when shared memory parallelism is used.

Table 4.3: Block Lanczos Eigensolver Memory and Disk Estimates

Memory Mode	Memory Usage Estimate	I/0 Files Size Estimate
Out-of-core	1.5 GB/MDOFs	15-20 GB/MDOFs
In-core	15-20 GB/MDOFs	~1.5 GB/MDOFs
Comments: • Add 30 percent to out-of-core memory for 3-D models with higher-order elements • Subtract 40 percent to out-of-core memory for 2-D or beam/shell element dominated models • Add 50 -100 percent to file size for in-core memory for 3-D models with higher-order elements • Subtract 50 percent to file size for in-core memory for 2-D or beam/shell element dominated models		

4.1.3.2. PCG Lanczos Solver

The 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.

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 the *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 (for example, 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 *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

Release 15.0 - © SAS IP, Inc. All rights reserved. - Contains proprietary and confidential information of ANSYS, Inc. and its subsidiaries and affiliates.

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 in-core memory mode or in an out-of-core memory mode. 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. By default, only a single matrix factorization is done by this solver unless the Sturm check option on the **PCGOPT** command is enabled, which results in one additional matrix factorization.

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 over *Lev_Diff* values 1 through 4 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. *Lev_Diff* = 5 is more efficient than other *Lev_Diff* values when more modes are requested, so larger numbers of modes may increase the size of problem for which a value of 5 should be used. The *Lev_Diff* value of 5 requires a costly factorization step which can be computed using an in-core memory mode or an out-of-core memory mode. Thus, when this option runs in the out-of-core memory mode on a machine with slow I/O performance, it decreases the size of problem for which a value of 5 should be used.

Using Lev_Diff = 5 with PCG Lanczos in Distributed ANSYS

The *Lev_Diff* value of 5 is supported in Distributed ANSYS. When used with the PCG Lanczos eigensolver, *Lev_Diff* = 5 causes this eigensolver to run in a completely distributed fashion. This is in contrast to the Block Lanczos method which can only operate in a shared memory fashion, even when used in Distributed ANSYS. Thus, the *Lev_Diff* = 5 setting provides a way to use a distributed eigensolver for the class of problems where the PCG Lanczos eigensolver's iterative method is not necessarily an efficient eigensolver choice (in other words, for problems with ill-conditioned matrices that are slow to converge when using *Lev_Diff* values 1 through 4).

The *Lev_Diff* = 5 setting can require a large amount of memory or disk I/O compared to *Lev_Diff* values of 1 through 4 because this setting uses a direct solver approach (i.e., a matrix factorization) within the Lanczos algorithm. However, by running in a distributed fashion it can spread out these resource requirements over multiples machines, thereby helping to achieve significant speedup and extending the class of problems for which the PCG Lanczos eigensolver is a good candidate. If *Lev_Diff* = 5 is specified, choosing the option to perform a Sturm check (via the **PCGOPT** command) does not require additional resources (e.g., additional memory usage or disk space). A Sturm check does require one additional factorization for the run to guarantee that no modes were skipped in the specified frequency range, and so it does require more computations to perform this extra factorization. However, since the *Lev_Diff* = 5 setting already does a matrix factorization for the Lanczos procedure, no extra memory or disk space is required.

Table 4.4: PCG Lanczos Memory and Disk Estimates

PCG Lanczos Solver (Shared Memory)
• Basic Memory requirement is 1.5 GB/MDOFs • Basic I/O Requirement is 2.0 GB/MDOFs
PCG Lanczos Solver (Distributed Memory, Using p Cores)
• Basic Memory requirement is 2.2 GB/ MDOFs – Each process 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 cores

Comments:

- Add 30 percent to memory requirement for higher-order solid elements
- Add 10-50 percent to memory requirement for higher level of difficulty preconditioners (PCGOPT 2-4)
- Save up to 70 percent for memory requirement from **MSAVE** option, when applicable
  - – SOLID186 / SOLID187 elements (at the expense of possibly longer runtime)
  - – SOLID185 elements (at the expense of possibly much longer runtime)

4.1.3.3. Supernode Solver

The Supernode eigensolver (**MODOPT**,SNODE) is designed to efficiently solve modal analyses in which a high number of modes is requested. For this class of problems, this solver often does less computation and uses considerably less computer resources than the Block Lanczos eigensolver. By utilizing fewer resources than Block Lanczos, the Supernode eigensolver becomes an ideal choice when solving this sort of analysis on the typical desktop machine, which can often have limited memory and slow I/O performance.

The **MODOPT** command allows you to specify how many frequencies are desired and what range those frequencies lie within. With other eigensolvers, the number of modes requested affects the performance of the solver, and the frequency range is essentially optional; asking for more modes increases the solution time, while the frequency range generally decides which computed frequencies are output. On the other hand, the Supernode eigensolver behaves completely opposite to the other solvers with regard to the **MODOPT** command input. This eigensolver will compute all of the frequencies within the requested range regardless of the number of modes the user requests. For maximum efficiency, it is highly recommended that you input a range that only covers the spectrum of frequencies between the first and last mode of interest. The number of modes requested on the **MODOPT** command then determines how many of the computed frequency modes are output.

The Supernode eigensolver benefits from shared-memory parallelism. Also, for users who want full control of this modal solver, the **SNOPTION** command gives you control over several important parameters that affect the accuracy and efficiency of the Supernode eigensolver.

Controlling Supernode Parameters

The Supernode eigensolver computes approximate eigenvalues. Typically, this should not be an issue as the lowest modes in the system (which are often used to compute the resonant frequencies) are computed very accurately (<< 1% difference compared to the same analysis performed with the Block Lanczos eigensolver). However, the accuracy drifts somewhat with the higher modes. For the highest requested modes in the system, the difference (compared to Block Lanczos) is often a few percent, and so it may be desirable in certain cases to tighten the accuracy of the solver. This can be done using the

Release 15.0 - © SAS IP, Inc. All rights reserved. - Contains proprietary and confidential information of ANSYS, Inc. and its subsidiaries and affiliates.

range factor (*RangeFact*) field on the **SNOPTION** command. Higher values of *RangeFact* lead to more accurate solutions at the cost of extra memory and computations.

When computing the final mode shapes, the Supernode eigensolver often does the bulk of its I/O transfer to and from disk. While the amount of I/O transfer is often significantly less than that done in a similar run using Block Lanczos, it can be desirable to further minimize this I/O, thereby maximizing the Supernode solver efficiency. You can do this by using the block size (*BlockSize*) field on the **SNOPTION** command. Larger values of *BlockSize* will reduce the amount of I/O transfer done by holding more data in memory, which generally speeds up the overall solution time. However, this is only recommended when there is enough physical memory to do so.

Release 15.0 - © SAS IP, Inc. All rights reserved. - Contains proprietary and confidential information of ANSYS, Inc. and its subsidiaries and affiliates.

Release 15.0 - © SAS IP, Inc. All rights reserved. - Contains proprietary and confidential information
of ANSYS, Inc. and its subsidiaries and affiliates.

Chapter 5: Parallel Processing Performance

When discussing parallel processing performance, the terms *speedup* and *scalability* often arise. This chapter describes how you can measure and improve speedup or scalability when running with either shared memory or distributed memory parallel activated, including the use of GPUs. The following topics are available:

5.1. What Is Scalability?
5.2. Measuring Scalability
5.3. Hardware Issues for Scalability
5.4. Software Issues for Scalability

5.1. What Is Scalability?

There are many ways to define the term scalability. For most users of the ANSYS program, scalability generally compares the total solution time of a simulation using one core with the total solution time of a simulation using some number of cores greater than one. In this sense, perfect scalability would mean a drop in solution time that directly corresponds to the increase in the number of processing cores; for example, a 4X decrease in solution time when moving from one to four cores. However, such ideal speedup is rarely seen in most software applications, including the ANSYS program.

There are a variety of reasons for this lack of perfect scalability. Some of the reasons are software- or algorithmic-related, while others are due to hardware limitations. For example, while Distributed ANSYS has been run on as many as 1024 cores, there is some point for every analysis at which parallel efficiency begins to drop off considerably. In this chapter, we will investigate the main reasons for this loss in efficiency.

5.2. Measuring Scalability

Scalability should always be measured using wall clock time or elapsed time, and not CPU time. CPU time can be either accumulated by all of the involved processors (or cores), or it can be the CPU time for any of the involved processors (or cores). CPU time may exclude time spent waiting for data to be passed through the interconnect or the time spent waiting for I/O requests to be completed. Thus, elapsed times provide the best measure of what the user actually experiences while waiting for the program to complete the analysis.

Several elapsed time values are reported near the end of every ANSYS output file. An example of this output is shown below.

```
Elapsed time spent pre-processing model (/PREP7)  :       3.7 seconds
Elapsed time spent solution - preprocessing       :       4.7 seconds
Elapsed time spent computing solution             :     284.1 seconds
Elapsed time spent solution - postprocessing      :       8.8 seconds
Elapsed time spent postprocessing model (/POST1) :       0.0 seconds
```

At the very end of the file, these time values are reported:

```
CP Time      (sec) =        300.890       Time  =  11:10:43
Elapsed Time (sec) =        302.000       Date  =  02/10/2009
```

Release 15.0 - © SAS IP, Inc. All rights reserved. - Contains proprietary and confidential information of ANSYS, Inc. and its subsidiaries and affiliates.

When using shared memory parallel, all of these elapsed times may be reduced since this form of parallelism is present in the preprocessing, solution, and postprocessing phases (**/PREP7**, **/SOLU**, and **/POST1**). Comparing each individual result when using a single core and multiple cores can give an idea of the parallel efficiency of each part of the simulation, as well as the overall speedup achieved.

When using Distributed ANSYS, the main values to review are:

- The "Elapsed time spent computing solution" which measures the time spent doing parallel work inside the **SOLVE** command (see ANSYS Program Architecture (p. 30) for more details).
- The "Elapsed Time" reported at the end of the output file.

The "Elapsed time spent computing solution" helps measure the parallel efficiency for the computations that are actually parallelized, while the "Elapsed Time" helps measure the parallel efficiency for the entire analysis of the model. Studying the changes in these values as the number of cores is increased/decreased will give a good indication of the parallel efficiency of Distributed ANSYS for your analysis.

When using a GPU to accelerate the solution, the same main values listed above should be reviewed. Since the GPU is currently only used to accelerate computations performed during solution, the elapsed time computing the solution is the only time expected to decrease. Of course, the overall time for the simulation should also decrease when using a GPU.

5.3. Hardware Issues for Scalability

This section discusses some key hardware aspects which affect the parallel performance of Distributed ANSYS.

5.3.1. Multicore Processors

Though multicore processors have extended parallel processing to virtually all computer platforms, some multicore processors have insufficient memory bandwidth to support all of the cores functioning at peak processing speed simultaneously. This can cause the performance efficiency of the ANSYS program to degrade significantly when all of the cores within a processor are used during solution. For some processors, we recommend only using a maximum of half the cores available in the processor. For example, on a cluster with two quad-core processors per node, we might recommend using at most four cores per node.

Given the wide range of CPU products available, it is difficult to determine the maximum number of cores to use for peak efficiency as this can vary widely. However, it is important to understand that having more cores or faster clock speeds does not always translate into proportionate speedups. Memory bandwidth is an important hardware issue for multicore processors and has a significant impact on the scalability of the ANSYS program. Finally, some processors run at faster clock speeds when only one core is used compared to when all cores are used. This can also have a negative impact on the scalability of the ANSYS program as more cores are used.

5.3.2. Interconnects

One of the most important factors to achieving good scalability performance using Distributed ANSYS when running across multiple machines is to have a good interconnect. Various forms of interconnects exist to connect multiple nodes on a cluster. Each type of interconnect will pass data at different speeds with regards to latency and bandwidth. See Hardware Terms and Definitions (p. 3) for more information on this terminology. A good interconnect is one that transfers data as quickly between cores on different machines as data moves between cores within a single machine.

Release 15.0 - © SAS IP, Inc. All rights reserved. - Contains proprietary and confidential information of ANSYS, Inc. and its subsidiaries and affiliates.

The interconnect is essentially the path for one machine to access memory on another machine. When a processing core needs to compute data, it has to access the data for the computations from some form of memory. With Distributed ANSYS that memory can either come from the local RAM on that machine or can come across the interconnect from another node on the cluster. When the interconnect is slow, the performance of Distributed ANSYS is degraded as the core must wait for the data. This "waiting" time causes the overall solution time to increase since the core cannot continue to do computations until the data is transferred. The more nodes used in a cluster, the more the speed of the interconnect will make a difference. For example, when using a total of eight cores with two nodes in a cluster (that is, four cores on each of two machines), the interconnect does not have as much of an impact on performance as another cluster configuration consisting of eight nodes using a single core on each node.

Typically, Distributed ANSYS achieves the best scalability performance when the communication bandwidth is above 1000 MB/s. This interconnect bandwidth value is printed out near the end of the Distributed ANSYS output file and can be used to help compare the interconnect of various hardware systems.

5.3.3. I/O Configurations

5.3.3.1. Single Machine

The I/O system used on a single machine (workstation, server, laptop, etc.) can be very important to the overall scalability of the ANSYS program. When running a job using parallel processing, the I/O system can be a sequential bottleneck that drags down the overall performance of the system.

Certain jobs perform more I/O than others. The sparse solver (including both the shared and distributed memory versions) running in the out-of-core memory mode along with the Block Lanczos eigensolver commonly perform the most amounts of I/O in the ANSYS program. Also, Distributed ANSYS can perform higher numbers of I/O requests than the shared memory version of ANSYS because each Distributed ANSYS process writes its own set of files. For jobs that perform a large amount of I/O, having a slow file system can impact the scalability of the ANSYS program because the elapsed time spent doing the I/O does not decrease as more CPU cores are utilized. If this I/O time is a significant portion of the overall runtime, then the scalability is significantly impacted.

Users should be aware of this potential bottleneck when running their jobs. It is highly recommended that a RAID0 array consisting of multiple hard drives be used for jobs that perform a large amount of I/O. One should also consider the choice of SSDs (solid state drives) when trying to minimize any I/O bottlenecks. SSDs, while more expensive than conventional hard drives, can have dramatically reduced seek times and demonstrate some impressive I/O transfer speeds when used in a RAID0 configuration.

5.3.3.2. Clusters

There are various ways to configure I/O systems on clusters. However, these configurations can generally be grouped into two categories: shared disk resources and independent disk resources. Each has its advantages and disadvantages, and each has an important effect on the scalability of Distributed ANSYS.

For clusters, the administrator(s) of the cluster will often setup a shared disk resource (SAN, NAS, etc.) where each node of a cluster can access the same disk storage location. This location may contain all of the necessary files for running a program like Distributed ANSYS across the cluster. While this is convenient for users of the cluster and for applications whose I/O configuration is setup to work in such an environment, this configuration can severely limit the scalability performance of Distributed ANSYS, particularly when using the distributed sparse solver. The reason is twofold. First, this type of I/O configuration often uses the same interconnect to transfer I/O data to the shared disk resource as Distributed

Release 15.0 - © SAS IP, Inc. All rights reserved. - Contains proprietary and confidential information of ANSYS, Inc. and its subsidiaries and affiliates.

ANSYS uses to transfer computational data between machines. This can place more demands on the interconnect, especially for a program like Distributed ANSYS which often does a lot of data transfer across the interconnect and often requires a large amount of I/O. Second, Distributed ANSYS is built to have each running process create and access its own set of files. Each process writes its own .ESAV file, .FULL file, .MODE file, results file, solver files, etc. In the case of running the distributed sparse solver in the out-of-core memory mode, this can be a huge amount of I/O and can cause a bottleneck in the interconnect used by the shared disk resource, thus hurting the scalability of Distributed ANSYS.

Alternatively, clusters might employ using local hard drives on each node of the cluster. In other words, each node has its own independent disk(s) shared by all of the cores within the node. Typically, this configuration is ideal assuming that (1) a limited number of cores are accessing the disk(s) or (2) multiple local disks are used in a RAID0 configuration. For example, if eight cores are used by Distributed ANSYS on a single node, then there are eight processes all trying to write their own set of I/O data to the same hard drive. The time to create and access this I/O data can be a big bottleneck to the scalability of Distributed ANSYS. When using local disks on each node of a cluster, or when using a single box server, you can improve performance by either limiting the number of cores used per machine or investing in an improved configuration consisting of multiple disks and a good RAID0 controller.

It is important to note that there are some very good network-attached storage solutions for clusters that employ separate high speed interconnects between processing nodes and a central disk resource. Often, the central disk resource has multiple disks that can be accessed independently by the cluster nodes. These I/O configurations can offer both the convenience of a shared disk resource visible to all nodes, as well as high speed I/O performance that scales nearly as well as independent local disks on each node. The best choice for an HPC cluster solution may be a combination of network-attached storage and local disks on each node.

5.3.4. GPUs

The GPU accelerator capability only supports the highest end GPU cards. The reason is that these high-end cards can offer some acceleration relative to the latest CPU cores. Older or less expensive graphics cards cannot typically offer this acceleration over CPU cores. When measuring the scalability of the GPU accelerator capability, it is important to consider not only the GPU being used, but also the CPU cores. These products (both CPUs and GPUs) are constantly evolving, and new products emerge on a regular basis. The number of available CPU cores along with the total peak computational rate when using those cores can impact the scalability. Older, slower CPUs often have less cores and show better GPU speedups, as the peak speed of the GPU will be greater relative to the peak speed of the older CPU cores. Likewise, as newer CPUs come to market (often with more cores), the GPU speedups may degrade since the peak speed of the GPU will be lessened relative to the peak speed of the new CPU cores.

5.4. Software Issues for Scalability

This section addresses some key software aspects which affect the parallel performance of the ANSYS program.

5.4.1. ANSYS Program Architecture

It should be expected that only the computations performed in parallel would speedup when more processing cores are used. In the ANSYS program, some computations before and after solution (for example, /PREP7 or /POST1) are setup to use some shared memory parallelism; however, the bulk of the parallel computations are performed during solution (specifically within the SOLVE command). Therefore, it would be expected that only the solution time would significantly decrease as more processing cores are used. Moreover, if a significant portion of the analysis time is spent anywhere outside

Release 15.0 - © SAS IP, Inc. All rights reserved. - Contains proprietary and confidential information of ANSYS, Inc. and its subsidiaries and affiliates.

solution, then adding more cores would not be expected to significantly decrease the solution time (that is, the efficiency of the ANSYS program would be greatly diminished for this case).

As described in Measuring Scalability (p. 27), the "Elapsed time spent computing solution" shown in the output file gives an indication of the amount of wall clock time spent actually computing the solution. If this time dominates the overall runtime, then it should be expected that parallel processing will help this model run faster as more cores are used. However, if this time is only a fraction of the overall runtime, then parallel processing should not be expected to help this model run significantly faster.

5.4.2. Distributed ANSYS

5.4.2.1. Contact Elements

Distributed ANSYS seeks to balance the number of elements, nodes, and DOFs for each process so that each process has roughly the same amount of work. However, this becomes a challenge when contact elements are present in the model. Contact elements often need to perform more computations at the element level (for example, searching for contact, penetration detection) than other types of elements. This can affect the scalability of Distributed ANSYS by causing one process to have much more work (that is, more computations to perform) than other processes, ultimately hurting the load balancing. This is especially true if very large contact pairs exist in the model (for example, when the number of contact/target elements is a big percentage of the total number of elements in the entire model). When this is the case, the best approach is to try and limit the scope of the contact to only what is necessary. Contact pairs can be trimmed using the **CNCHECK**,TRIM command.

5.4.2.2. Using the Distributed PCG Solver

One issue to consider when using the PCG solver is that higher level of difficulty values (*Lev_Diff* on the **PCGOPT** command), can hurt the scalability of Distributed ANSYS. These higher values of difficulty are often used for models that have difficulty converging within the PCG solver, and they are typically necessary to obtain optimal performance in this case when using a limited number of cores. However, when using a higher number of cores, for example more than 16 cores, it may be wise to consider lowering the level of difficulty value by 1 (if possible) in order to improve the overall solver performance. Lower level of difficulty values scale better than higher level of difficulty values; thus, the optimal *Lev_Diff* value at a few cores will not necessary be the optimal *Lev_Diff* value at a high number of cores.

5.4.2.3. Using the Distributed Sparse Solver

When using the distributed sparse solver, you should always consider which memory mode is being used. For optimal scalability, the in-core memory mode should always be used. This mode avoids writing the large matrix factor file. When running in the out-of-core memory mode, each Distributed ANSYS process must create and access its own set of solver files, which can cause a bottleneck in performance as each process tries to access the hard drive(s). Since hard drives can only seek to one file at a time, this file access within the solver becomes a big sequential block in an otherwise parallel code.

Fortunately, the memory requirement to run in-core is divided among the number of cluster nodes used for a Distributed ANSYS simulation. While some single core runs may be too large for a single compute node, a 4 or 8 node configuration may easily run the distributed sparse solver in-core. In Distributed ANSYS, the in-core mode will be selected automatically in most cases whenever available physical memory on each node is sufficient. If very large models require out-of-core factorization, even when using several compute nodes, local I/O on each node will help to scale the I/O time as more compute nodes are used.

Release 15.0 - © SAS IP, Inc. All rights reserved. - Contains proprietary and confidential information of ANSYS, Inc. and its subsidiaries and affiliates.

5.4.2.4. Combining Files

After a parallel solution successfully completes, Distributed ANSYS automatically combines some of the local files written by each processor into a single, global file. These include the `.RST` (or `.RTH`), `.ESAV`, `.EMAT`, `.MODE`, `.IST`, `.MLV`, and `.SELD` files. This step can be costly due to the large amount of I/O and MPI communication involved. In some cases, this step can be a bottleneck for performance as it involves serial operations.

Automatic file combination is performed when the **FINISH** command is executed upon leaving the solution processor. If any of these global files are not needed to perform downstream operations, you can often reduce the overall solution time by suppressing the file combination for each individual file type that is not needed (see the **DMPOPTION** command for more details). In addition, reducing the amount of data written to the results file (see **OUTRES** command) can also help improve the performance of this step by reducing the amount of I/O and MPI communication required to combine the local results files into a single, global results file.

5.4.3. GPU Accelerator Capability

Similar to the expectations described in ANSYS Program Architecture (p. 30), the GPU accelerator capability will typically accelerate the computations only during solution. Thus, if the solution time is only a fraction of the overall runtime, then the GPU accelerator capability is not expected to help the model run significantly faster.

Also, different amounts of speedup are expected depending on the equation solver used as well as various model features (e.g., geometry, element types, analysis options, etc.). All of these factors affect how many computations are off-loaded onto the GPU for acceleration; the more opportunity for the GPU to accelerate the solver computations, the more opportunity for improved speedups.

Release 15.0 - © SAS IP, Inc. All rights reserved. - Contains proprietary and confidential information of ANSYS, Inc. and its subsidiaries and affiliates.

Chapter 6: Measuring ANSYS Performance

A key step in maximizing ANSYS performance on any hardware system is to properly interpret the ANSYS output. This chapter describes 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:

6.1. Sparse Solver Performance Output
6.2. Distributed Sparse Solver Performance Output
6.3. Block Lanczos Solver Performance Output
6.4. PCG Solver Performance Output
6.5. PCG Lanczos Solver Performance Output
6.6. Supernode Solver Performance Output
6.7. Identifying CPU, I/O, and Memory Performance

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.

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 (p. 35) 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 core, the CPU time is a subset of the wall time. However, when using multiple cores, some systems accumulate the CPU times from all cores, 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 core, 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 (p. 35)).

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 for the bulk of 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 (p. 35)) 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 measured 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 5000 Mflops (5 Gflops) to over 15000 Mflops on a single core. For parallel factorization, compute rates can now approach 100 Gflops using the fastest multicore processors, although rates of 20 to 80 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 in-core, the effective I/O rate is really a measure of memory bandwidth, and rates of 3000 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 1000+ 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 50 to 100 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 Linux servers). With experience, a glance at the effective I/O rate will reveal whether a sparse solver analysis ran in-core, 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 (p. 35)). In the example, the LN09 file is 18904 MB and is written once and read twice for a total of 56712 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 in-core memory only when the memory required fits comfortably within the available physical memory.

This example shows a well-balanced system (high factor mflops and adequate I/O rate from multiple disks in a RAID0 configuration). Additional performance gains could be achieved by using the in-core memory mode on a machine with more physical memory and by using more than one core.

Release 15.0 - © SAS IP, Inc. All rights reserved. - Contains proprietary and confidential information of ANSYS, Inc. and its subsidiaries and affiliates.

Example 6.1: Sparse Solver Performance Summary

```
number of equations                      =         1058610
no. of nonzeroes in lower triangle of a  =        25884795
number of compressed nodes               =          352870
no. of compressed nonzeroes in l. tri.   =         4453749
amount of workspace currently in use     =        14399161
max. amt. of workspace used              =       308941051
no. of nonzeroes in the factor l         =      2477769015.
number of super nodes                    =           12936
number of compressed subscripts          =         6901926
size of stack storage                    =       485991486
maximum order of a front matrix          =           22893
maximum size of a front matrix           =       262056171
maximum size of a front trapezoid        =         1463072
no. of floating point ops for factor     =      2.2771D+13
no. of floating point ops for solve      =      8.0370D+09
actual no. of nonzeroes in the factor l  =      2477769015.
actual number of compressed subscripts   =         6901926
actual size of stack storage used        =       285479868
negative pivot monitoring activated
number of negative pivots encountered    =              0.
factorization panel size                 =              64
factorization update panel size          =              32
solution block size                      =               2
number of cores used                     =               1
time (cpu & wall) for structure input    =        3.030000        3.025097
time (cpu & wall) for ordering           =       22.860000       22.786610
time (cpu & wall) for symbolic factor    =        0.400000        0.395507
time (cpu & wall) for value input        =        3.310000        3.304702
time (cpu & wall) for numeric factor     =     2051.500000     2060.382486  <---A (Factor)
computational rate (mflops) for factor   =    11099.910696    11052.058028  <---A (Factor)
condition number estimate                =      0.0000D+00
time (cpu & wall) for numeric solve      =       14.050000      109.646978  <---B (Solve)
computational rate (mflops) for solve    =      572.028766       73.298912  <---B (Solve)
effective I/O rate (MB/sec) for solve    =     2179.429565      279.268850  <---C (I/O)

i/o stats:    unit           file length              amount transferred
                           words      mbytes              words      mbytes
             ----          -----      ------              -----      ------
               20      61735699.       471. MB     133437507.      1018. MB
               25      13803852.       105. MB      34509630.       263. MB
                9    2477769015.     18904. MB    7433307045.     56712. MB  <---D (File)
               11     375004575.      2861. MB    1566026058.     11948. MB

          -------     ----------     --------     ----------     --------
          Totals:    2928313141.     22341. MB    9167280240.     69941. MB


  Sparse Matrix Solver      CPU Time (sec) =        2097.670
  Sparse Matrix Solver  ELAPSED Time (sec) =        2204.057
  Sparse Matrix Solver   Memory Used ( MB) =        2357.033
```

6.2. Distributed Sparse Solver Performance Output

Similar to the shared memory sparse solver, performance information for the distributed memory version of the sparse solver is printed by default to the Distributed ANSYS file `Jobname.DSP`. Also, the command **DSPOPTION**,,,,,,PERFORMANCE can be used to print this same information, along with additional solver information, to the standard Distributed ANSYS output file.

Example 6.2: Distributed Sparse Solver Performance Summary for 4 Processes and 1 GPU (p. 36) shows an example of the performance summary output from the distributed memory sparse solver. Most of this performance information is identical in format and content to what is described in Sparse Solver Performance Output (p. 33). However, a few items are unique or are presented differently when using the distributed memory sparse solver, which we will discuss next.

Release 15.0 - © SAS IP, Inc. All rights reserved. - Contains proprietary and confidential information of ANSYS, Inc. and its subsidiaries and affiliates.

Example 6.2: Distributed Sparse Solver Performance Summary for 4 Processes and 1 GPU

```
number of equations                      =          774144
no. of nonzeroes in lower triangle of a =        30765222
no. of nonzeroes in the factor l         =      1594957656
ratio of nonzeroes in factor (min/max)   =          0.4749 <---A
number of super nodes                    =           29048
maximum order of a front matrix          =           18240
maximum size of a front matrix           =       166357920
maximum size of a front trapezoid        =       162874176
no. of floating point ops for factor     =      1.1858D+13
no. of floating point ops for solve      =      6.2736D+09
ratio of flops for factor (min/max)      =          0.2607 <---A
negative pivot monitoring activated
number of negative pivots encountered    =               0
factorization panel size                 =             128
number of cores used                     =               4
GPU acceleration activated                                 <---D
percentage of GPU accelerated flops      =         99.6603 <---D
time (cpu & wall) for structure input    =        0.250000         0.252863
time (cpu & wall) for ordering           =        4.230000         4.608260
time (cpu & wall) for value input        =        0.280000         0.278143
time (cpu & wall) for matrix distrib.    =        1.060000         1.063457
time (cpu & wall) for numeric factor     =      188.690000       405.961216
computational rate (mflops) for factor   =    62841.718360     29208.711028
time (cpu & wall) for numeric solve      =       90.560000       318.875234
computational rate (mflops) for solve    =       69.275294        19.674060
effective I/O rate (MB/sec) for solve    =      263.938866        74.958169

i/o stats: unit-Core          file length              amount transferred
                           words        mbytes            words        mbytes
              ----      ----------     --------      ----------      --------
           90-   0      332627968.     2538. MB      868420752.      6626. MB <---B
           90-   1      287965184.     2197. MB      778928589.      5943. MB <---B
           90-   2      397901824.     3036. MB     1091969277.      8331. MB <---B
           90-   3      594935808.     4539. MB     1617254575.     12339. MB <---B
           93-   0       58916864.      450. MB      274514418.      2094. MB
           93-   1       66748416.      509. MB      453739218.      3462. MB
           93-   2       64585728.      493. MB      910612446.      6947. MB
           93-   3      219578368.     1675. MB      968105082.      7386. MB
           94-   0       10027008.       76. MB       20018352.       153. MB
           94-   1       10256384.       78. MB       20455440.       156. MB
           94-   2       10584064.       81. MB       21149352.       161. MB
           94-   3       11272192.       86. MB       22481832.       172. MB

           -------      ----------     --------      ----------      --------
           Totals:     2065399808.    15758. MB     7047649333.     53769. MB


  Memory allocated on core    0        =    1116.498 MB <---C
  Memory allocated on core    1        =     800.319 MB <---C
  Memory allocated on core    2        =    1119.103 MB <---C
  Memory allocated on core    3        =    1520.533 MB <---C
  Total Memory allocated by all cores  =    4556.453 MB


  DSP Matrix Solver         CPU Time (sec) =        285.240
  DSP Matrix Solver     ELAPSED Time (sec) =        739.525
  DSP Matrix Solver      Memory Used ( MB) =       1116.498
```

While factorization speed and I/O performance remain important factors in the overall performance of the distributed sparse solver, the balance of work across the processing cores is also important. Items marked (A) in the above example give some indication of how evenly the computations and storage requirements are balanced across the cores. Typically, the more evenly distributed the work (that is, the closer these values are to 1.0), the better the performance obtained by the solver. Often, one can improve the balance (if necessary) by changing the number of cores used; in other words, by splitting the matrix factorization across more or less cores (for example, 3 or 5 cores instead of 4).

Release 15.0 - © SAS IP, Inc. All rights reserved. - Contains proprietary and confidential information of ANSYS, Inc. and its subsidiaries and affiliates.

The I/O statistics for the distributed memory sparse solver are presented a little differently than they are for the shared memory sparse solver. The first column shows the file unit followed by the core to which that file belongs ("unit-Core" heading). With this solver, the matrix factor file is `JobnameN.DSPtri`, or unit 90. Items marked (B) in the above example show the range of sizes for this important file. This also gives some indication of the computational load balance within the solver. The memory statistics printed at the bottom of this output (items marked (C)) help give some indication of how much memory was used by each utilized core to run the distributed memory sparse solver. If multiple cores are used on any node of a cluster (or on a workstation), the sum of the memory usage and disk usage for all cores on that node/workstation should be used when comparing the solver requirements to the physical memory and hard drive capacities of the node/workstation. If a single node on a cluster has slow I/O performance or cannot buffer the solver files in memory, it will drag down the performance of all cores since the solver performance is only as fast as the slowest core.

Items marked (D) show that GPU acceleration was enabled and used for this model. In this example, over 99% of the matrix factorization flops were accelerated on the GPU hardware. However, the numeric solve time is almost as great as the numeric factorization time. This is because the job is heavily I/O bound, as evidenced by the slow 75 MB/sec of I/O performance in the numeric solve computations and by the large difference in CPU and elapsed times for the numeric factorization computations. Due to this high I/O cost, the overall impact of using a GPU to accelerate the factorization computations was certainly lessened.

This example shows a very unbalanced system with high factorization speed but very poor I/O performance. Significant performance improvements could be achieved by simply running on a machine with more physical memory. Alternatively, making significant improvements to the disk configuration, possibly through the use of multiple SSD drives in a RAID0 configuration, would also be beneficial.

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 matrices that are 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 in-core or out-of-core depending on the memory available at the start of the Lanczos procedure. 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. The block size can be controlled via the *BlockSize* field on the **MODOPT** command. The requirements for each memory mode are computed using either the default block size or the user-specified block size. If these requirements happen to fall slightly short of what Lanczos requires, Lanczos will automatically reduce the block size and try to continue. This is generally a rare occurrence. However, if it does occur, forcing a specific memory value for the solver using **BCSOPTION**,,FORCE,*Memory_Size* should help increase the block size if *Memory_Size* is slightly bigger than the memory size the Block Lanczos solver originally allocated for your model.

When running the solver in out-of-core mode, a reduced 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`. Increasing the block size will slightly increase

Release 15.0 - © SAS IP, Inc. All rights reserved. - Contains proprietary and confidential information of ANSYS, Inc. and its subsidiaries and affiliates.

the memory required to run Block Lanczos. However, when running in out-of-core mode, increasing the block size from 8 to 12 or 16 may significantly reduce the amount of I/O done and, therefore, reduce the total solution time. This is most often the case if there is not enough physical memory on the system to hold the Block Lanczos files in the system cache (that is, the cost of I/O in the block solves appears to be very expensive).

Example 6.3: Block Lanczos Performance Summary (p. 38) shows an example of the output from the Block Lanczos eigensolver. This example gives details on the costs of various steps of the Lanczos algorithm. The important parts of this summary for measuring hardware performance are the times and rates for factorization (A) and solves (B) computed using CPU and wall times. Wall times are the most useful because they include all system overhead, including the cost of I/O. In this run, the factorization performance is measured at 10808 Mflops, while the solve rate was over 2500 Mflops and the matrix multiply rate was over 1300 Mflops. This suggests that the solver ran in the optimal out-of-core memory mode (hence the large `Jobname.LN07` file), but had enough physical memory that the OS was able to cache the files using physical memory and achieve good performance. Indeed, this is the case.

This example shows a well-balanced system with high computational speed and fast memory bandwidth. Additional performance gains could be achieved by using the in-core memory mode and using more than one core.

Other statistics from Example 6.3: Block Lanczos Performance Summary (p. 38) that are useful for comparing Lanczos performance are the number of factorizations (C), Lanczos steps (D), block solves (E), and Lanczos block size (F). 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 eigenvalues/modes within the specified range or miss any of the lowest-valued eigenvalues/modes. 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 typically 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 (that is, modes closest to zero).

The final part of Example 6.3: Block Lanczos Performance Summary (p. 38) shows the files used by the Block Lanczos run. The largest file is unit 7 (`Jobname.LN07`), the matrix factor file (G). 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 `Jobname.LN07` file is over 12 GB in size, and a total of 389 GB of I/O to this file was required (G).

Example 6.3: Block Lanczos Performance Summary

```
number of equations                    =          774144
no. of nonzeroes in lower triangle of a =        18882786
number of compressed nodes             =          258048
no. of compressed nonzeroes in l. tri.  =         3246326
amount of workspace currently in use    =        39099545
b has general form.
no. of nonzeroes in lower triangle of b =         9738978
max. amt. of workspace used            =       249584441
no. of nonzeroes in the factor l       =      1593901620.
number of super nodes                  =           10476
number of compressed subscripts        =         5206005
size of stack storage                  =       305256930
maximum order of a front matrix        =           18144
maximum size of a front matrix         =       164611440
maximum size of a front trapezoid      =         1159136
no. of flt. pt. ops for a single factor =    1.176060D+13
```


Release 15.0 - © SAS IP, Inc. All rights reserved. - Contains proprietary and confidential information of ANSYS, Inc. and its subsidiaries and affiliates.

```
no. of flt. pt. ops for a block solve   =    5.183227D+09
no. of flt. pt. ops for block mtx mult. =    3.178404D+08
total no. of flt. pt. ops for factor    =    2.352119D+13
total no. of flt. pt. ops for solve     =    6.219873D+11
total no. of flt. pt. ops for mtx mult. =    1.144226D+10
actual no. of nonzeroes in the factor l =      1593901620.
actual number of compressed subscripts  =         5206005
actual size of stack storage used       =       197037093
pivot tolerance used for factorization  =        0.000000
factorization panel size                =              64
factorization update panel size         =              32
solution block size                     =               8
lanczos block size                      =               8 <---F
number of cores used                    =               1
total number of factorizations          =               2 <---C
total number of lanczos runs            =               1
total number of lanczos steps           =              14 <---D
total number of block solves            =              15 <---E
time (cpu & wall) for structure input   =        2.890000        2.876984
time (cpu & wall) for ordering          =       20.480000       20.399301
time (cpu & wall) for symbolic factor   =        0.290000        0.284939
time (cpu & wall) for value input       =        3.410000        3.420249

time (cpu & wall) for numeric factor    =     2184.600000     2176.245220 <---A (Factor)
computational rate (mflops) for factor  =    10766.818317    10808.152999 <---A (Factor)
time (cpu & wall) for numeric solve     =      247.750000      247.046678 <---B (Solve)
computational rate (mflops) for solve   =     2510.544085     2517.691403 <---B (Solve)
time (cpu & wall) for matrix multiply   =        8.580000        8.549612
computational rate (mflops) for mult.   =     1333.596285     1338.336260

cost (elapsed time) for sparse eigenanalysis
-----------------------------
lanczos run start up cost               =       18.802963
lanczos run recurrence cost             =      231.153277
lanczos run reorthogonalization cost    =       22.879937
lanczos run internal eigenanalysis cost =        0.005127
lanczos eigenvector computation cost    =        4.207299
lanczos run overhead cost               =        0.438301

total lanczos run cost                  =      277.486904
total factorization cost                =     2176.245238
shift strategy and overhead  cost       =       20.179957

total sparse eigenanalysis cost         =     2473.912098

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

               20      63915155.      488. MB     191745465.     1463. MB
               21      10412010.       79. MB      31236030.      238. MB
               25      92897280.      709. MB     532611072.     4064. MB
               28      86704128.      662. MB     346816512.     2646. MB
                7    1593901620.    12161. MB   51004851840.   389136. MB <---G (File)
                9     241078854.     1839. MB    2050167804.    15642. MB
               11      15482880.      118. MB      15482880.      118. MB

           Total:    2104391927.    16055. MB   54172911603.   413307. MB

Block Lanczos       CPU Time (sec) =       2494.730
Block Lanczos  ELAPSED Time (sec) =       2504.100
Block Lanczos   Memory Used ( MB) =       1904.178
```

6.4. PCG Solver Performance Output

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

Release 15.0 - © SAS IP, Inc. All rights reserved. - Contains proprietary and confidential information of ANSYS, Inc. and its subsidiaries and affiliates.

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 6.4: Jobname.PCS Output File (p. 41) shows a typical `Jobname.PCS` file.

The memory information in Example 6.4: Jobname.PCS Output File (p. 41) shows that this 2.67 million DOF PCG solver run requires only 1.5 GB of memory (A). This model uses SOLID186 (Structural Solid) elements which, by default, use the **MSAVE**,ON feature in ANSYS for this static analysis. The **MSAVE** 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 which preconditioner was used by means of the level of difficulty (D). By default, 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 the PCG solver is probably not an efficient option, and the matrix could possibly require the use of the sparse direct solver for a faster solution time.

The key here is to find the best preconditioner option using *Lev_Diff* that balances the cost per iteration as well as the total number of iterations. Simply reducing the number of iterations with an increased *Lev_Diff* does not always achieve the expected end result: lower elapsed time to solution. The reason is that the cost per iteration may increase too greatly for this case. Another option which can add complexity to this decision is parallel processing. For both SMP and Distributed ANSYS, using more cores to help with the computations will reduce the cost per iteration, which typically shifts the optional *Lev_Diff* value slightly lower. Also, lower *Lev_Diff* values in general scale better with the preconditioner computations when parallel processing is used. Therefore, when using 16 or more cores it is recommended that you decrease by one the optimal *Lev_Diff* value found when using one core in an attempt to achieve better scalability and improve overall solver performance.

The CPU performance reported in the PCS file is divided into matrix multiplication using the stiffness matrix (E) and the various compute kernels of the preconditioner (F). It is normal that the Mflop rates reported in the PCS file are a lot lower than those reported with the sparse solver 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 for 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 PCG Lanczos eigensolver and only for smaller problems (under 1 million DOFs) where the factored matrix (matrices) usually fit in memory.

Release 15.0 - © SAS IP, Inc. All rights reserved. - Contains proprietary and confidential information of ANSYS, Inc. and its subsidiaries and affiliates.

This example shows a model that performed quite well with the PCG solver. Considering that it converged in about 1000 iterations, the Lev_Diff value of 1 is probably optimal for this model (especially at higher core counts). However, in this case it might be worthwhile to try Lev_Diff = 2 to see if it improves the solver performance. Using more than one core would also certainly help to reduce the time to solution.

Example 6.4: Jobname.PCS Output File

```
        Number of cores used: 1
        Degrees of Freedom: 2671929
        DOF Constraints: 34652
        Elements: 211280 <---B
                Assembled: 0 <---G (MSAVE,ON does not apply)
                Implicit: 211280 <---G (MSAVE,ON applies)
        Nodes: 890643
        Number of Load Cases: 1

        Nonzeros in Upper Triangular part of
                     Global Stiffness Matrix : 0
        Nonzeros in Preconditioner: 46325031
                *** Precond Reorder: MLD ***
                Nonzeros in V: 30862944
                Nonzeros in factor: 10118229
                Equations in factor: 25806
        *** Level of Difficulty: 1   (internal 0) *** <---D (Preconditioner)

        Total Operation Count: 2.07558e+12
        Total Iterations In PCG: 1042 <---C (Convergence)
        Average Iterations Per Load Case: 1042.0
        Input PCG Error Tolerance: 1e-08
        Achieved PCG Error Tolerance: 9.93822e-09


        DETAILS OF PCG SOLVER SETUP TIME(secs)        Cpu         Wall
             Gather Finite Element Data              0.30         0.29
             Element Matrix Assembly                 6.89         6.80

        DETAILS OF PCG SOLVER SOLUTION TIME(secs)     Cpu         Wall
             Preconditioner Construction             6.73         6.87
             Preconditioner Factoring                1.32         1.32
             Apply Boundary Conditions               0.24         0.25
             Preconditioned CG Iterations          636.98       636.86
                  Multiply With A                  470.76       470.64 <---E (Matrix Mult. Time)
                       Multiply With A22           470.76       470.64
                  Solve With Precond               137.10       137.01 <---F (Preconditioner Time)
                       Solve With Bd                26.63        26.42
                       Multiply With V              89.47        89.37
                       Direct Solve                 14.87        14.94
******************************************************************************
             TOTAL PCG SOLVER SOLUTION CP TIME      =     645.89 secs
             TOTAL PCG SOLVER SOLUTION ELAPSED TIME =     648.25 secs
******************************************************************************
        Total Memory Usage at CG         :    1514.76 MB <---A (Memory)
        PCG Memory Usage at CG           :     523.01 MB
        Memory Usage for MSAVE Data      :     150.90 MB
        *** Memory Saving Mode Activated : Jacobians Precomputed ***
******************************************************************************
        Multiply with A MFLOP Rate       :    3911.13 MFlops
        Solve With Precond MFLOP Rate    :    1476.93 MFlops
        Precond Factoring MFLOP Rate     :       0.00 MFlops
******************************************************************************
        Total amount of I/O read         :    1873.54 MB <---G
        Total amount of I/O written      :    1362.51 MB <---G
******************************************************************************
```

6.5. PCG Lanczos Solver Performance Output

The PCG Lanczos eigensolver uses the Lanczos algorithm to compute eigenvalues and eigenvectors (frequencies and mode shapes) for modal analyses, but replaces matrix factorization and multiple solves

Release 15.0 - © SAS IP, Inc. All rights reserved. - Contains proprietary and confidential information of ANSYS, Inc. and its subsidiaries and affiliates.

with multiple iterative solves. In other words, it replaces a direct sparse solver with the iterative PCG solver while keeping the same Lanczos algorithm. An iterative solution is faster than matrix factorization, but usually takes longer than a single block solve. The real power of the PCG Lanczos method is experienced for very large models, usually above a few million DOFs, where matrix factorization and solves become very expensive.

The PCG Lanczos method will automatically choose an appropriate default level of difficulty, but experienced users may improve solution time by manually specifying the level of difficulty via the **PCGOPT** command. 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 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 given matrix is available, and the cost of factorization is not dominant.

The performance summary for PCG Lanczos is contained in the file `Jobname.PCS`, with additional information related to the Lanczos solver. The first part of the `.PCS` file contains information specific to the modal analysis, including the computed eigenvalues and frequencies. The second half of the `.PCS` file contains similar performance data as found in a static or transient analysis. As highlighted in the next two examples, 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 number of load cases corresponds to the number of Lanczos steps required to obtain the specified number of eigenvalues. It 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 relatively small number of modes (up to 100) for large models (over a few million DOF).

The next two examples show parts of the PCS file that report performance statistics described above for a 2 million 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 6.5: PCS File for PCG Lanczos, Level of Difficulty = 3 (p. 42) uses **PCGOPT**,3. The output shows that *Lev_Diff* = 3 (C), and the total iterations required for 25 Lanczos steps (A) is 2355 (B), or an average of 94.2 iterations per step (E). Example 6.6: PCS File for PCG Lanczos, Level of Difficulty = 5 (p. 44) shows that increasing *Lev_Diff* to 5 (C) on **PCGOPT** reduces the iterations required per Lanczos step to just one (E).

Though the solution time difference in these examples shows that a *Lev_Diff* value of 3 is faster in this case (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).

This example shows a model that performed quite well with the PCG Lanczos eigensolver. Considering that it converged in under 100 iterations per load case, the Lev_Diff value of 3 is probably too high for this model (especially at higher core counts). In this case, it might be worthwhile to try Lev_Diff = 1 or 2 to see if it improves the solver performance. Using more than one core would also certainly help to reduce the time to solution.

Example 6.5: PCS File for PCG Lanczos, Level of Difficulty = 3

```
Lanczos Solver Parameters
------------------------
        Lanczos Block Size: 1
        Eigenpairs computed: 10 lowest
```


Release 15.0 - © SAS IP, Inc. All rights reserved. - Contains proprietary and confidential information of ANSYS, Inc. and its subsidiaries and affiliates.

```
        Extra Eigenpairs: 1
        Lumped Mass Flag: 0
        In Memory Flag: 0
        Extra Krylov Dimension: 1
        Mass Matrix Singular Flag: 1
        PCCG Stopping Criteria Selector: 4
        PCCG Stopping Threshold: 1.000000e-04
        Extreme Preconditioner Flag: 0
        Reortho Type: 1
        Number of Reorthogonalizations: 7
        nExtraWorkVecs for computing eigenvectors: 4
        Rel. Eigenvalue Tolerance: 1.000000e-08
        Rel. Eigenvalue Residual Tolerance: 1.000000e-11
        Restart Condition Number Threshold: 1.000000e+15
        Sturm Check Flag: 0

        Shifts Applied:   1.017608e-01

Eigenpairs

Number of Eigenpairs 10

---------------------------------------

No.     Eigenvalue      Frequency(Hz)
---     ----------      -------------
1       1.643988e+03    6.453115e+00
2       3.715504e+04    3.067814e+01
3       5.995562e+04    3.897042e+01
4       9.327626e+04    4.860777e+01
5       4.256303e+05    1.038332e+02
6       7.906460e+05    1.415178e+02
7       9.851501e+05    1.579688e+02
8       1.346627e+06    1.846902e+02
9       1.656628e+06    2.048484e+02
10      2.050199e+06    2.278863e+02



        Number of cores used: 1
        Degrees of Freedom: 2067051
        DOF Constraints: 6171
        Elements: 156736
                Assembled: 156736
                Implicit: 0
        Nodes: 689017
        Number of Load Cases: 25 <---A (Lanczos Steps)

        Nonzeros in Upper Triangular part of
                     Global Stiffness Matrix : 170083104
        Nonzeros in Preconditioner: 201288750
                *** Precond Reorder: MLD ***
                Nonzeros in V: 12401085
                Nonzeros in factor: 184753563
                Equations in factor: 173336
        *** Level of Difficulty: 3   (internal 2) *** <---C (Preconditioner)

        Total Operation Count: 3.56161e+12
        Total Iterations In PCG: 2355 <---B (Convergence)
        Average Iterations Per Load Case:   94.2 <---E (Iterations per Step)
        Input PCG Error Tolerance: 0.0001
        Achieved PCG Error Tolerance: 9.98389e-05


        DETAILS OF PCG SOLVER SETUP TIME(secs)        Cpu        Wall
             Gather Finite Element Data               0.40        0.40
             Element Matrix Assembly                 96.24       96.52

        DETAILS OF PCG SOLVER SOLUTION TIME(secs)     Cpu        Wall
             Preconditioner Construction              1.74        1.74
             Preconditioner Factoring                51.69       51.73
             Apply Boundary Conditions                5.03        5.03
```

Release 15.0 - © SAS IP, Inc. All rights reserved. - Contains proprietary and confidential information of ANSYS, Inc. and its subsidiaries and affiliates.

```
             Eigen Solve                          3379.36     3377.52
                  Eigen Solve Overhead             172.49      172.39
                       Compute MQ                  154.00      154.11
                       Reorthogonalization         123.71      123.67
                            Computation            120.66      120.61
                            I/O                      3.05        3.06
                       Block Tridiag Eigen           0.00        0.00
                       Compute Eigenpairs            1.63        1.63
                       Output Eigenpairs             0.64        0.64
                  Multiply With A                 1912.52     1911.41
                       Multiply With A22          1912.52     1911.41
                  Solve With Precond              1185.62     1184.85
                       Solve With Bd                89.84       89.89
                       Multiply With V             192.81      192.53
                       Direct Solve                880.71      880.38
******************************************************************************
             TOTAL PCG SOLVER SOLUTION CP TIME      =    3449.01 secs
             TOTAL PCG SOLVER SOLUTION ELAPSED TIME =    3447.20 secs <---D (Total Time)
******************************************************************************
        Total Memory Usage at Lanczos    :    3719.16 MB
        PCG Memory Usage at Lanczos      :    2557.95 MB
        Memory Usage for Matrix          :       0.00 MB
******************************************************************************
        Multiply with A Memory Bandwidth :      15.52 GB/s
        Multiply with A MFLOP Rate       :     833.13 MFlops
        Solve With Precond MFLOP Rate    :    1630.67 MFlops
        Precond Factoring MFLOP Rate     :       0.00 MFlops
******************************************************************************
        Total amount of I/O read         :    6917.76 MB
        Total amount of I/O written      :    6732.46 MB
******************************************************************************
```

Example 6.6: PCS File for PCG Lanczos, Level of Difficulty = 5

```
Lanczos Solver Parameters
-------------------------
        Lanczos Block Size: 1
        Eigenpairs computed: 10 lowest
        Extra Eigenpairs: 1
        Lumped Mass Flag: 0
        In Memory Flag: 0
        Extra Krylov Dimension: 1
        Mass Matrix Singular Flag: 1
        PCCG Stopping Criteria Selector: 4
        PCCG Stopping Threshold: 1.000000e-04
        Extreme Preconditioner Flag: 1
        Reortho Type: 1
        Number of Reorthogonalizations: 7
        nExtraWorkVecs for computing eigenvectors: 4
        Rel. Eigenvalue Tolerance: 1.000000e-08
        Rel. Eigenvalue Residual Tolerance: 1.000000e-11
        Restart Condition Number Threshold: 1.000000e+15
        Sturm Check Flag: 0

        Shifts Applied:  -1.017608e-01

Eigenpairs

Number of Eigenpairs 10

---------------------------------------

No.     Eigenvalue      Frequency(Hz)
---     ----------      -------------
1       1.643988e+03    6.453116e+00
2       3.715494e+04    3.067810e+01
3       5.995560e+04    3.897041e+01
4       9.327476e+04    4.860738e+01
5       4.256265e+05    1.038328e+02
6       7.906554e+05    1.415187e+02
7       9.851531e+05    1.579690e+02
8       1.346626e+06    1.846901e+02
```


Release 15.0 - © SAS IP, Inc. All rights reserved. - Contains proprietary and confidential information of ANSYS, Inc. and its subsidiaries and affiliates.

```
9       1.656620e+06    2.048479e+02
10      2.050184e+06    2.278854e+02




        Number of cores used: 1
        Degrees of Freedom: 2067051
        DOF Constraints: 6171
        Elements: 156736
                Assembled: 156736
                Implicit: 0
        Nodes: 689017
        Number of Load Cases: 25 <---A (Lanczos Steps)

        Nonzeros in Upper Triangular part of
                     Global Stiffness Matrix : 170083104
        Nonzeros in Preconditioner: 4168012731
                *** Precond Reorder: MLD ***
                Nonzeros in V: 0
                Nonzeros in factor: 4168012731
                Equations in factor: 2067051
        *** Level of Difficulty: 5   (internal 0) *** <---C (Preconditioner)

        Total Operation Count: 4.34378e+11
        Total Iterations In PCG: 25 <---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


        DETAILS OF PCG SOLVER SETUP TIME(secs)        Cpu        Wall
             Gather Finite Element Data              0.42        0.43
             Element Matrix Assembly               110.99      111.11

        DETAILS OF PCG SOLVER SOLUTION TIME(secs)     Cpu        Wall
             Preconditioner Construction            26.16       26.16
             Preconditioner Factoring             3245.98     3246.01
             Apply Boundary Conditions               5.08        5.08
             Eigen Solve                          1106.75     1106.83
                  Eigen Solve Overhead             198.14      198.15
                       Compute MQ                  161.40      161.28
                       Reorthogonalization         130.51      130.51
                            Computation            127.45      127.44
                            I/O                      3.06        3.07
                       Block Tridiag Eigen           0.00        0.00
                       Compute Eigenpairs            1.49        1.49
                       Output Eigenpairs             0.62        0.62
                  Multiply With A                    7.89        7.88
                       Multiply With A22             7.89        7.88
                  Solve With Precond                 0.00        0.00
                       Solve With Bd                 0.00        0.00
                       Multiply With V               0.00        0.00
                       Direct Solve                908.61      908.68
******************************************************************************
             TOTAL PCG SOLVER SOLUTION CP TIME      =    4395.84 secs
             TOTAL PCG SOLVER SOLUTION ELAPSED TIME =    4395.96 secs <---D (Total Time)
******************************************************************************
        Total Memory Usage at Lanczos    :    3622.87 MB
        PCG Memory Usage at Lanczos      :    2476.45 MB
        Memory Usage for Matrix          :       0.00 MB
******************************************************************************
        Multiply with A Memory Bandwidth :      39.94 GB/s
        Solve With Precond MFLOP Rate    :     458.69 MFlops
        Precond Factoring MFLOP Rate     :       0.00 MFlops
******************************************************************************
        Total amount of I/O read         :   11853.51 MB
        Total amount of I/O written      :    7812.11 MB
******************************************************************************
```

6.6. Supernode Solver Performance Output

The Supernode eigensolver works by taking the matrix structure of the stiffness and mass matrix from the original FEA model and internally breaking it into pieces (supernodes). These supernodes are then used to reduce the original FEA matrix to a much smaller system of equations. The solver then computes all of the modes and mode shapes within the requested frequency range on this smaller system of equations. Then, the supernodes are used again to transform (or expand) the smaller mode shapes back to the larger problem size from the original FEA model.

The power of this eigensolver is best experienced when solving for a high number of modes, usually more than 200 modes. Another benefit is that this eigensolver typically performs much less I/O than the Block Lanczos eigensolver and, therefore, is especially useful on typical desktop machines that often have limited disk space and/or slow I/O transfer speeds.

Performance information for the Supernode Eigensolver is printed by default to the file `Jobname.DSP`. Use the command **SNOPTION**,,,,,,PERFORMANCE to print this same information, along with additional solver information, to the standard ANSYS output file.

When studying performance of this eigensolver, each of the three key steps described above (reduction, solution, expansion) should be examined. The first step of forming the supernodes and performing the reduction increases in time as the original problem size increases; however, it typically takes about the same amount of computational time whether 10 modes or 1000 modes are requested. In general, the larger the original problem size is relative to the number of modes requested, the larger the percentage of solution time spent in this reduction process.

The next step, solving the reduced eigenvalue problem, increases in time as the number of modes in the specified frequency range increases. This step is typically a much smaller portion of the overall solver time and, thus, does not often have a big effect on the total time to solution. However, this depends on the size of the original problem relative to the number of requested modes. The larger the original problem, or the fewer the requested modes within the specified frequency range, the smaller will be the percentage of solution time spent in this step. Choosing a frequency range that covers only the range of frequencies of interest will help this step to be as efficient as possible.

The final step of expanding the mode shapes can be an I/O intensive step and, therefore, typically warrants the most attention when studying the performance of the Supernode eigensolver. The solver expands the final modes using a block of vectors at a time. This block size is a controllable parameter and can help reduce the amount of I/O done by the solver, but at the cost of more memory usage.

Example 6.7: DSP File for Supernode (SNODE) Solver (p. 46) shows a part of the `Jobname.DSP` file that reports the performance statistics described above for a 1.5 million DOF modal analysis that computes 1000 modes and mode shapes. The output shows the cost required to perform the reduction steps (A), the cost to solve the reduced eigenvalue problem (B), and the cost to expand and output the final mode shapes to the `Jobname.RST` and `Jobname.MODE` files (C). The block size for the expansion step is also shown (D). At the bottom of this file, the total size of the files written by this solver is printed with the total amount of I/O transferred.

In this example, the reduction time is by far the most expensive piece. The expansion computations are running at over 3000 Mflops, and this step is not relatively time consuming. In this case, using more than one core would certainly help to significantly reduce the time to solution.

Example 6.7: DSP File for Supernode (SNODE) Solver

```
number of equations                     =         1495308
no. of nonzeroes in lower triangle of a =        41082846
no. of nonzeroes in the factor l        =       873894946
```


Release 15.0 - © SAS IP, Inc. All rights reserved. - Contains proprietary and confidential information of ANSYS, Inc. and its subsidiaries and affiliates.

```
ratio of nonzeroes in factor (min/max)  =         1.0000
number of super nodes                   =           6554
maximum order of a front matrix         =           6324
maximum size of a front matrix          =       19999650
maximum size of a front trapezoid       =       14547051
no. of floating point ops for eigen sol =     8.0502D+12
no. of floating point ops for eigen out =     1.2235D+12
no. of equations in global eigenproblem =           5983
factorization panel size                =            256
supernode eigensolver block size        =             40 <---D
number of cores used                    =              1
time (cpu & wall) for structure input   =       1.520000        1.531496
time (cpu & wall) for ordering          =       4.900000        4.884938
time (cpu & wall) for value input       =       1.050000        1.045083
time (cpu & wall) for matrix distrib.   =       3.610000        3.604622
time (cpu & wall) for eigen solution    =    1888.840000     1949.006305
computational rate (mflops) for eig sol =    4261.983276     4130.414801
effective I/O rate (MB/sec) for eig sol =                     911.661409
time (cpu & wall) for eigen output      =     377.760000      377.254216
computational rate (mflops) for eig out =    3238.822665     3243.164948
effective I/O rate (MB/sec) for eig out =                    1175.684097

cost (elapsed time) for SNODE eigenanalysis
-----------------------------
Substructure eigenvalue cost            =     432.940317 <---A (Reduction)
Constraint mode & Schur complement cost =     248.458204 <---A (Reduction)
Guyan reduction cost                    =     376.112464 <---A (Reduction)
Mass update cost                        =     715.075189 <---A (Reduction)
Global eigenvalue cost                  =     146.099520 <---B (Reduced Problem)
Output eigenvalue cost                  =     377.254077 <---C (Expansion)

i/o stats: unit-Core          file length             amount transferred
                            words        mbytes            words        mbytes
              ----      ----------     --------       ----------      --------
           17-   0      628363831.     4794. MB     16012767174.    122168. MB
           45-   0      123944224.      946. MB      1394788314.     10641. MB
           46-   0       35137755.      268. MB        70275407.       536. MB
           93-   0       22315008.      170. MB       467938080.      3570. MB
           94-   0       52297728.      399. MB       104582604.       798. MB
           98-   0       22315008.      170. MB       467938080.      3570. MB
           99-   0       52297728.      399. MB       104582604.       798. MB

           -------      ----------     --------       ----------      --------
           Totals:      936671282.     7146. MB     18622872263.    142081. MB


Total Memory allocated =      561.714 MB
```

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

Table 6.1: Obtaining Performance Statistics from ANSYS Solvers (p. 48) 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 and file size information from the sparse solver are important because they set boundaries indicating which problems can run efficiently using the in-core memory mode and which problems should use optimal out-of-core memory mode.

The expected results summarized in the table below are for current systems. Continual improvements in processor performance are expected, although processor clock speeds have 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 and as SSD technology improves. The sparse solver effective I/O rate statistic can determine whether a given system is getting in-core 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 provide a measure of memory bandwidth. The computations in the iterative solver place a high demand on memory

Release 15.0 - © SAS IP, Inc. All rights reserved. - Contains proprietary and confidential information of ANSYS, Inc. and its subsidiaries and affiliates.

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.

Table 6.1: Obtaining Performance Statistics from ANSYS Solvers

Solver	Performance Stats Source	Expected Results on a Balanced System
Sparse	**BCSOPTION**,,,,,,PERFORMANCE command or `Jobname.BCS` file	5000-15000 Mflops factor rate for 1 core 50-100 MB/sec effective I/O rate for single conventional drive 150-300 MB/sec effective I/O rate for Windows 64-bit RAID0, 4 conventional drives, or striped Linux configuration 200-1000 MB/sec using high-end SSDs in a RAID0 configuration 1000-3000 MB/sec effective I/O rate for in-core or when system cache is larger than file size
Distributed sparse	**DSPOPTION**,,,,,,PERFORMANCE command or `Jobname.DSP` file	Similar to sparse solver above when using a single core Note: I/O performance can significantly degrade if many MPI processes are writing to the same disk resource. In-core memory mode or using SSDs is recommended.
LANB - Block Lanczos	**BCSOPTION**,,,,,,PERFORMANCE command or `Jobname.BCS` file	Same as sparse solver 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 1500-3000 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.

Release 15.0 - © SAS IP, Inc. All rights reserved. - Contains proprietary and confidential information of ANSYS, Inc. and its subsidiaries and affiliates.

Solver	Performance Stats Source	Expected Results on a Balanced System
		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` - always written by PCG Lanczos eigensolver	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 for smaller jobs when system can handle factorization cost well.

Release 15.0 - © SAS IP, Inc. All rights reserved. - Contains proprietary and confidential information
of ANSYS, Inc. and its subsidiaries and affiliates.

Release 15.0 - © SAS IP, Inc. All rights reserved. - Contains proprietary and confidential information
of ANSYS, Inc. and its subsidiaries and affiliates.

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 following topics are available:

7.1. ANSYS Examples
7.2. Distributed ANSYS Examples

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 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 SOLID186 elements. Two model sizes are used to demonstrate expected performance for in-core and out-of-core HPC systems.

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. The Windows 32-bit system has previous generation Xeon processors, so the CPU performance is not directly comparable to the 64-bit system. However, the 32-bit system is representative of many 32-bit desktop workstation configurations, and comparison with larger memory 64-bit workstations shows the savings from reducing I/O costs. In addition, the 32-bit system has a fast RAID0 drive I/O configuration as well as a standard single drive partition; the two I/O configurations are compared in the second example.

The model size for the first solver example is 250K DOFs. Example 7.1: SMP Sparse Solver Statistics Comparing Windows 32-bit and Windows 64-bit (p. 52) 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 size is 1746 MB (A), and the total file storage for the sparse solver in this run is 2270 MB. This problem does not run in-core on a Windows 32-bit system, but easily runs in-core on the 8 GB Windows 64-bit HPC system. Even when using a single core on each machine, the CPU performance is doubled on the Windows 64-bit system, (from 1918 Mflops (B) to 4416 Mflops (C)) reflecting the performance of the current Intel Xeon core micro architecture processors. These processors can achieve 4 flops per core per clock cycle, compared to the previous generation Xeon processors that achieve only 2 flops per core 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 in-core solves—a factor of over 50X speedup compared to 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). If the same Intel processor used in the 64-bit system were used on a desktop Windows 32-bit system, the performance would be closer to the 64-bit system performance, but the limitation of memory would still increase I/O costs significantly.

Release 15.0 - © SAS IP, Inc. All rights reserved. - Contains proprietary and confidential information of ANSYS, Inc. and its subsidiaries and affiliates.

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, 1 core, optimal out-of-core mode, 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, 1 core, in-core mode, 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 In-core vs. Out-of-Core (p. 53) shows an in-core 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 in-core 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 in-core run solve time (C) is only 1.26 seconds, compared to a factorization time of almost 60 seconds. The larger model, which cannot run in-core 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 50 to 100 MB/sec for the forward/backward 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

Release 15.0 - © SAS IP, Inc. All rights reserved. - Contains proprietary and confidential information of ANSYS, Inc. and its subsidiaries and affiliates.

time because the preprocessing times and the I/O and solves are not parallel; only the computations during the matrix factorization are 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 higher-order solid elements) the factorization time can be hours, and parallel speedup for these models is significant. For smaller models, running in-core 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 in-core run. Only models that run “comfortably” within the available physical memory should run in-core.

Example 7.2: SMP Sparse Solver Statistics Comparing In-core vs. Out-of-Core

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

```
 Windows 64-bit system, 250k DOFs, 2 cores, in-core mode

      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, 2 cores, optimal out-of-core mode

      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 re-

Release 15.0 - © SAS IP, Inc. All rights reserved. - Contains proprietary and confidential information
of ANSYS, Inc. and its subsidiaries and affiliates.

quirement 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 in-core, *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 Minimum Memory Mode (p. 54) 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. The **BCSOPTION** command was also used to force the minimum memory mode. This memory mode 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 Block Lanczos uses a memory allocation of 338 MB (A) to run. This amount is just above the minimum allowed for the out-of-core solution in Block Lanczos (B). This forced minimum memory mode is not recommended, but is used in this example to illustrate how the reduction of block size when memory is insufficient can greatly increase I/O time.

The performance summary in Example 7.3: Windows 32-bit System Using Minimum Memory Mode (p. 54) 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, it is important to check the Lanczos block size (H). By default, the block size is 8. However, it may be reduced, as in this case, if the memory given to the Block Lanczos solver is slightly less than what is required. This occurs because the memory size formulas are heuristic, and not perfectly accurate. ANSYS usually provides the solver with sufficient memory so that the block size is rarely reduced.

When the I/O cost is so severe, as it is for this example due to the minimum memory mode, it is recommended that you increase the Lanczos block size using the **MODOPT** command (if there is available memory to do so). Usually, a block size of 12 or 16 would help improve performance for this sort of example by reducing the number of block solves, thus reducing the amount of I/O done by the solver.

When running in the minimum memory mode, ANSYS will also write the mass matrix to disk. This means that the mass matrix multiply operations are done out-of-core, as well as the factorization computations. Overall, the best way to improve performance for this example is to move to a Windows 64-bit system with 8 GB or more of memory. However, the performance on the Windows 32-bit system can be significantly improved in this case simply by avoiding the use of the minimum memory mode. This mode should only be used when the need arises to solve the biggest possible problem on a given machine and when the user can tolerate the resulting poor performance. The optimal out-of-core memory mode often provides much better performance without using much additional memory over the minimum memory mode.

Example 7.3: Windows 32-bit System Using Minimum Memory Mode

500k DOFs Block Lanczos Run Computing 40 Modes

```
Memory allocated for solver =              338.019 MB   <---A
Memory required for in-core =             5840.798 MB
Optimal memory required for out-of-core =  534.592 MB
Minimum memory required for out-of-core =  306.921 MB   <---B

     Lanczos block size                     =              5 <---H
```


Release 15.0 - © SAS IP, Inc. All rights reserved. - Contains proprietary and confidential information of ANSYS, Inc. and its subsidiaries and affiliates.

```
      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 Time)
      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 Optimal Out-of-core Memory Mode (p. 55) shows results from a run on the same Windows 32-bit system using the optimal out-of-core memory mode. With this change, the memory used for this Lanczos run is 588 MB (A). This is more than enough to run in this memory mode (B) and well shy of what is needed to run in-core.

This run uses a larger block size than the previous example 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 solver memory usage 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 by simply avoiding use of the minimum memory mode.

Example 7.4: Windows 32-bit System Using Optimal Out-of-core Memory Mode

500k DOFs Block Lanczos Run Computing 40 Modes

```
 Memory allocated for solver =               587.852 MB  <---A
 Memory required for in-core =              5840.798 MB
 Optimal memory required for out-of-core =   534.592 MB
```

```
 Minimum memory required for out-of-core =   306.921 MB  <---B

      Lanczos block size                     =              8
      total number of factorizations         =              2
      total number of lanczos runs           =              1
      total number of lanczos steps          =             16
      total number of block solves           =             17 <---C
      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
```

When running on a Windows 32-bit system, a good rule of thumb for Block Lanczos runs is to always make sure the initial ANSYS memory allocation (-m) 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 (-m) 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. A good initial memory allocation can often help avoid out-of-memory errors that can sometimes occur on Windows 32-bit systems due to the limited memory address space of this platform (see Memory Limits on 32-bit Systems (p. 10)).

Example 7.5: Windows 32-bit System with 2 Processors and RAID0 I/O (p. 57) shows that a Windows 32-bit system with a RAID0 I/O configuration and parallel processing (along with the optimal out-of-core memory mode used in Example 7.4: Windows 32-bit System Using Optimal Out-of-core Memory Mode (p. 55)) can reduce the modal analysis time even further. The initial Windows 32-bit run took over 2 hours to compute 40 modes; but when using RAID0 I/O and parallel processing with a better memory mode, this job ran in just over half an hour (A). This example shows that, with a minimal invest-

Release 15.0 - © SAS IP, Inc. All rights reserved. - Contains proprietary and confidential information of ANSYS, Inc. and its subsidiaries and affiliates.

ment 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.

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 Optimal Out-of-Core Memory Mode (p. 57) shows some of the performances statistics from a run which uses the same memory mode as one of the previous examples done using the Windows 32-bit system. However, a careful comparison with Example 7.5: Windows 32-bit System with 2 Processors and RAID0 I/O (p. 57) shows that this Windows 64-bit system is over 2 times faster (A) than the best Windows 32-bit system results, even when using the same number of cores (2) and when using a RAID0 disk array on the Windows 32-bit system. It is over 7 times faster than the initial Windows 32-bit system using a single core, minimum memory mode, and a standard single disk drive.

In Example 7.7: Windows 64-bit System Using In-core Memory Mode (p. 58), a further reduction in Lanczos total solution time (A) is achieved using the **BCSOPTION**,,INCORE option. The solve time is reduced to just 127 seconds (B). This compares with 3517.9 seconds in the original 32-bit Windows run 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 in-core 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 7.6: Windows 64-bit System Using Optimal Out-of-Core Memory Mode

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

```
      total number of lanczos steps          =           16
      total number of block solves           =           17
      time (cpu & wall) for structure input  =     2.656250        2.645897
      time (cpu & wall) for ordering         =     8.390625        9.603793
```

Release 15.0 - © SAS IP, Inc. All rights reserved. - Contains proprietary and confidential information of ANSYS, Inc. and its subsidiaries and affiliates.

```
      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 <---A (Total Time)

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

Example 7.7: Windows 64-bit System Using In-core Memory Mode

500k DOFs Block Lanczos Run Computing 40 Modes; 8 Cores, 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 <---B (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 <---A (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 7.1: Summary of Block Lanczos Memory Guidelines (p. 59) 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. 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 block size can often help when running Lanczos out-of-core on machines with limited memory and poor I/O performance.

The 500k DOF Lanczos example is a large model for a Windows 32-bit desktop system, but an easy in-core 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 in this mode than to attempt running in-core using up all or nearly all of the physical memory on the system.

One way to force an in-core Lanczos run on a large memory machine is to start ANSYS with a 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 in-core. Once the in-core memory is known for a given problem, it is possible to start ANSYS with enough memory initially so

Release 15.0 - © SAS IP, Inc. All rights reserved. - Contains proprietary and confidential information of ANSYS, Inc. and its subsidiaries and affiliates.

that the Block Lanczos solver run will run in-core 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 in-core 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. In-core 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.

Table 7.1: Summary of Block Lanczos Memory Guidelines

Memory Mode	Guideline
In-core Lanczos runs	• Use only if in-core memory < 90% of total physical memory of system ("comfortable" in-core memory).
Out-of-core Lanczos runs	• Increase block size using the **MODOPT** command when poor I/O performance is seen in order to reduce number of block solves. *or* • 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. • 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 in-core factorization (results in sluggish system performance and limits system caching of all other files).	

7.2. Distributed ANSYS Examples

The following topics are discussed in this section:

7.2.1. Distributed ANSYS Memory and I/O Considerations
7.2.2. Distributed ANSYS Sparse Solver Example
7.2.3. Guidelines for Iterative Solvers in Distributed ANSYS

7.2.1. Distributed ANSYS Memory and I/O Considerations

Distributed ANSYS is a distributed memory parallel version of ANSYS that uses MPI (message passing interface) for communication between Distributed ANSYS processes. Each MPI process is a separate ANSYS process, with each opening ANSYS files and allocating memory. In effect, each MPI process

Release 15.0 - © SAS IP, Inc. All rights reserved. - Contains proprietary and confidential information of ANSYS, Inc. and its subsidiaries and affiliates.

functions as a separate ANSYS job for much of the execution time. The global solution in a Distributed ANSYS run is obtained through communication using an MPI software library. The first, or master, MPI process initiates Distributed ANSYS 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 resource requirements for the primary compute node (that is, the machine that contains the master process).

Since each MPI process in Distributed ANSYS has separate I/O to its own set of files, the I/O demands for a cluster system can be substantial. Cluster systems typically have one of the two configurations for I/O discussed here. First, some cluster systems use a centralized I/O setup where all processing nodes write to a single 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 Distributed ANSYS. The performance bottleneck occurs, in part, because the MPI communication needed to perform the Distributed ANSYS solution must wait for the I/O transfer over the same interconnect, and also because the Distributed ANSYS processes all write to the same disk (or set of disks). Often, the interconnect and I/O configuration can not keep up with all the I/O done by Distributed ANSYS.

The other common I/O configuration for cluster systems is to simply use independent, local disks at each compute node on the cluster. This I/O configuration avoids any extra communication over the interconnect and provides a natural scaling for I/O performance, provided only one core per node is used on the cluster. With the advent of multicore servers and multicore compute nodes, it follows that users will want to use multiple MPI processes with Distributed ANSYS on a multicore server/node. However, this leads to multiple ANSYS processes competing for access to the file system and, thus, creates another performance bottleneck for Distributed ANSYS.

To achieve optimal performance, it is important to understand the I/O and memory requirements for each solver type, direct and iterative; they are discussed separately with examples in the following sections.

7.2.2. Distributed ANSYS Sparse Solver Example

The distributed sparse solver used in Distributed ANSYS is not the same sparse solver used for SMP ANSYS runs. It operates both in-core and out-of-core, just as the SMP sparse solver; but there are important differences. Three important factors affect the parallel performance of the distributed sparse solver: 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 in-core 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 cores are used.

By contrast, the total memory required for the in-core memory mode for the distributed sparse solver is essentially constant as more cores are used. Thus, the memory per process to factor the matrix in-core actually shrinks when more cores are used. Interestingly, when enough cores are used, the in-core and optimal out-of-core memory modes become equivalent. This usually occurs with more than 4 cores. In this case, distributed sparse solver runs may switch from I/O dominated out-of-core performance to in-core performance. This means that when running on a handful of nodes on a cluster, the solver may not have enough memory to run in-core (or get in-core type performance with the system buffer caching the solver files), so the only option is to run out-of-core. However, by simply using more nodes

Release 15.0 - © SAS IP, Inc. All rights reserved. - Contains proprietary and confidential information of ANSYS, Inc. and its subsidiaries and affiliates.

on the cluster, the solver memory (and I/O) requirements are spread out such that the solver can begin to run with in-core performance.

The load balance factor cannot be directly controlled by the user. The distributed sparse 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. It is important to note, however, that using different numbers of cores can affect the load balance indirectly. For example, if the load balance using 8 cores seems poor in the solver, you may find that it improves when using either 7 or 9 cores, resulting in better overall performance. This is because the solver's internal decomposition of the matrix changes completely as different numbers of processes are involved in the computations.

Initial experience with Distributed ANSYS can lead to frustration with performance due to any combination of the effects mentioned above. Fortunately, cluster systems are becoming much more powerful since memory configurations of 16 GB per node are commonplace today. The following detailed example illustrates a model that requires more memory to run in-core than is available on 1 or 2 nodes. A cluster configuration with 16 GB per node would run this example very well on 1 and 2 nodes, but an even larger model would eventually cause the same performance limitations illustrated below if a sufficient number of nodes is not used to obtain optimal performance.

Parts 1, 2, and 3 of Example 7.8: Distributed Sparse Solver Run for 750k DOF Static Analysis (p. 62) show distributed sparse solver performance statistics that illustrate the memory, I/O and load balance factors for a 750k DOF static analysis. In these runs the single core times come from the sparse solver in ANSYS. The memory required to run this model in-core is over 13 GB on one core. The cluster system used in this example has 4 nodes, 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 in-core, except that the total memory is not globally addressable. Therefore, only runs that use all 4 nodes can run in-core. All I/O goes through the system interconnect, and all files used are accessed from a common file system located on the host node (or primary compute node). 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 processor cores share the same disk resource and must transfer I/O using the system interconnect hardware.

Part 1 of Example 7.8: Distributed Sparse Solver Run for 750k DOF Static Analysis (p. 62) shows performance statistics for runs using the sparse solver in ANSYS and using the distributed sparse solver in Distributed ANSYS on 2 cores (single cores on two different nodes). This example shows that the I/O cost significantly increases both the factorization and the solves for the Distributed ANSYS run. This is best seen by comparing the CPU and elapsed times. Time spent waiting for I/O to complete is not attributed towards the CPU time. Thus, having elapsed times that are significantly greater than CPU times usually indicates a high I/O cost.

In this example, the solve I/O rate measured in the Distributed ANSYS run was just over 30 MB/sec (A), while the I/O rate for the sparse solver in ANSYS was just over 50 MB/sec (B). This I/O performance difference reflects the increased cost of I/O from two cores sharing a common disk through the standard, inexpensive interconnect. The memory required for a 2 core Distributed ANSYS run exceeds the 6 GB available on each node; thus, the measured effective I/O rate is a true indication of I/O performance on this configuration. Clearly, having local disks on each node would significantly speed up the 2 core job as the I/O rate would be doubled and less communication would be done over the interconnect.

Part 2 of Example 7.8: Distributed Sparse Solver Run for 750k DOF Static Analysis (p. 62) shows performance statistics for runs using the out-of-core and in-core memory modes with the distributed sparse solver and 4 cores on the same system. In this example, the parallel runs were configured to use all

Release 15.0 - © SAS IP, Inc. All rights reserved. - Contains proprietary and confidential information of ANSYS, Inc. and its subsidiaries and affiliates.

available nodes as cores were added, rather than using all available cores on the first node before adding a second node. Note that the 4 core out-of-core run shows a substantial improvement compared to the 2 core run and also shows a much higher effective I/O rate—over 260 MB/sec (C). Accordingly, the elapsed time for the forward/backward solve drops from 710 seconds (D) to only 69 seconds (E). This higher performance reflects the fact that the 4 core out-of-core run now has enough local memory for each process to cache its part of the large matrix factor. The 4 core in-core run is now possible because the in-core memory requirement per process, averaging just over 4 GB (F), is less than the available 6 GB per node. The performance gain is nearly 2 times over the out-of-core run, and the wall time for the forward/backward solve is further reduced from 69 seconds (E) to just 3 seconds (G).

Part 3 of Example 7.8: Distributed Sparse Solver Run for 750k DOF Static Analysis (p. 62) shows in-core runs using 6 and 8 cores for the same model. The 6 core run shows the effect of load balancing on parallel performance. Though load balancing is not directly measured in the statistics shown here, the amount of memory required for each core is an indirect indicator that some processes have more of the matrix factor to store (and compute) than other processes. Load balance for the distributed sparse solver is never perfectly even and is dependent on the problem and the number of cores involved. In some cases, 6 cores will provide a good load balance, while in other situations, 4 or 8 may be better.

For all of the in-core runs in Parts 2 and 3, the amount of memory required per core decreases, even though total memory usage is roughly constant as the number of cores increases. It is this phenomenon that provides a speedup of over 6X on 8 cores 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 multinode cluster configuration.

Example 7.8: Distributed Sparse Solver Run for 750k DOF Static Analysis

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

```
 Sparse solver in ANSYS using 1 core, 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 Rate)

      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)


 Distributed sparse solver in Distributed ANSYS using 2 cores, out-of-core mode

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


Release 15.0 - © SAS IP, Inc. All rights reserved. - Contains proprietary and confidential information of ANSYS, Inc. and its subsidiaries and affiliates.

```
time (cpu & wall) for structure input              0.67        0.69
time (cpu & wall) for ordering                    15.27       16.38
time (cpu & wall) for value input                  2.30        2.33
time (cpu & wall) for matrix distrib.             12.89       25.60
time (cpu & wall) for numeric factor             791.77     1443.04
computational rate (mflops) for factor          5940.07     3259.21
time (cpu & wall) for numeric solve               23.48      710.15 <---D (Solve Time)
ccomputational rate (mflops) for solve           253.25        8.37
effective I/O rate (MB/sec) for solve            964.90       31.90 <---A (I/O Rate)

Memory allocated on core   0         =   831.216 MB
Memory allocated on core   1         =   797.632 MB
Total Memory allocated by all cores  =  1628.847 MB

DSP Matrix Solver         CPU Time (sec) =        846.38
DSP Matrix Solver     ELAPSED Time (sec) =       2198.19
```

Part 2: Out-of-Core and In-core Performance Statistics on 4 Cores

Distributed sparse solver in Distributed ANSYS using 4 cores, out-of-core mode

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

time (cpu & wall) for numeric factor            440.33      722.55
computational rate (mflops) for factor        10587.26     6451.98
time (cpu & wall) for numeric solve               9.11       86.41
computational rate (mflops) for solve           652.52       68.79 <---E (Solve Time)
effective I/O rate (MB/sec) for solve          2486.10      262.10 <---C (I/O Rate)

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

DSP Matrix Solver         CPU Time (sec) =        475.88
DSP Matrix Solver     ELAPSED Time (sec) =        854.82
```

Distributed sparse solver in Distributed ANSYS using 4 cores, in-core mode

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

time (cpu & wall) for numeric factor             406.11      431.13
computational rate (mflops) for factor         11479.37    10813.12
time (cpu & wall) for numeric solve                2.20        3.27 <---G (Solve Time)
computational rate (mflops) for solve           2702.03     1819.41
effective I/O rate (MB/sec) for solve          10294.72     6931.93

Memory allocated on core   0         =  4734.209 MB <---F
Memory allocated on core   1         =  4264.859 MB <---F
Memory allocated on core   2         =  4742.822 MB <---F
Memory allocated on core   3         =  4361.079 MB <---F
Total Memory allocated by all cores  = 18102.970 MB

DSP Matrix Solver         CPU Time (sec) =        435.22
DSP Matrix Solver     ELAPSED Time (sec) =        482.31
```

Part 3: Out-of-Core and In-core Performance Statistics on 6 and 8 Cores

Distributed sparse solver in Distributed ANSYS using 6 cores, in-core mode

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

time (cpu & wall) for numeric factor            254.85      528.68
computational rate (mflops) for factor        18399.17     8869.37
time (cpu & wall) for numeric solve               1.65        6.05
computational rate (mflops) for solve          3600.12      981.97
```

Release 15.0 - © SAS IP, Inc. All rights reserved. - Contains proprietary and confidential information of ANSYS, Inc. and its subsidiaries and affiliates.

```
effective I/O rate (MB/sec) for solve           13716.45     3741.30

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

DSP Matrix Solver         CPU Time (sec) =        281.44
DSP Matrix Solver     ELAPSED Time (sec) =        582.25
```

Distributed sparse solver in Distributed ANSYS using 8 cores, in-core mode

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

time (cpu & wall) for numeric factor             225.36      258.47
computational rate (mflops) for factor          20405.51    17791.41
time (cpu & wall) for numeric 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

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

DSP Matrix Solver         CPU Time (sec) =        252.21
DSP Matrix Solver     ELAPSED Time (sec) =        307.06
```

7.2.3. Guidelines for Iterative Solvers in Distributed ANSYS

Iterative solver performance in Distributed ANSYS 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.

It is important to note that the optimal value for *Lev_Diff* changes with the number of cores used. In other words, the optimal value of *Lev_Diff* for a given model using one core is not always the optimal value for the same model when using, for example, 8 cores. Typically, lower Lev_Diff values scale better, so the general rule of thumb is to try lowering the Lev_Diff value used (if possible) when running the PCG solver in Distributed ANSYS on 8 or more cores. Users may choose *Lev_Diff* = 1 because this option will normally exhibit the best parallel speedup. However, if a model exhibits very slow convergence, evidenced by a 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

Release 15.0 - © SAS IP, Inc. All rights reserved. - Contains proprietary and confidential information of ANSYS, Inc. and its subsidiaries and affiliates.

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 7.9: PCG Solver Run for 5 MDOF (p. 65) show portions of `Jobname.PCS` for the run on a 4 node 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 implicitly assembled with 0 nonzeros in the global assembled matrix). For *Lev_Diff* = 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 7X on 16 cores versus 5.5X in Part 2). However, the total elapsed time is lower in Part 2 for both 1 and 16 cores, respectively. The reduced parallel speedups in Part 2 result from the higher preconditioner cost and decreased scalability for the preconditioner used when *Lev_Diff* = 2.

If more cores were used with this problem (for example, 32 or 64 cores), one might expect that the better scalability of the *Lev_Diff* = 1 runs might result in a lower elapsed time than when using the *Lev_Diff* = 2 preconditioner. This example demonstrates the general idea that the algorithms which are fastest on a single core are not necessarily the fastest algorithms at 100 cores. Conversely, the fastest algorithms at 100 cores are not always the fastest on one core. Therefore, it becomes a challenge to define scalability. However, to the end user the fastest time to solution is usually what matters the most. 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 7.9: PCG Solver Run for 5 MDOF (p. 65) report memory use for both preconditioner options. The peak memory usage for the PCG solver often 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 in relatively small pieces during the matrix assembly portion of the PCG solver, and performance is not dependent on a single large memory allocation. 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 core on a Windows 32-bit system with the /3GB switch enabled.

Example 7.9: PCG Solver Run for 5 MDOF

Part 1: Lev_Diff = 1 on 1 and 16 Cores

```
 Number of Cores Used: 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 Core Used (Distributed Memory Parallel): 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 Cores

```
 Number of Cores Used: 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)
```


Release 15.0 - © SAS IP, Inc. All rights reserved. - Contains proprietary and confidential information of ANSYS, Inc. and its subsidiaries and affiliates.

```
******************************************************************************

 Number of Cores Used (Distributed Memory Parallel): 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
******************************************************************************
```

Release 15.0 - © SAS IP, Inc. All rights reserved. - Contains proprietary and confidential information of ANSYS, Inc. and its subsidiaries and affiliates.

Release 15.0 - © SAS IP, Inc. All rights reserved. - Contains proprietary and confidential information
of ANSYS, Inc. and its subsidiaries and affiliates.

Appendix A. Glossary

This appendix contains a glossary of terms used in the *Performance Guide*.

Cache
High speed memory that is located on a CPU or core. Cache memory can be accessed much faster than main memory, but it is limited in size due to high cost and the limited amount of space available on multicore processors. Algorithms that can use data from the cache repeatedly usually perform at a much higher compute rate than algorithms that access larger data structures that cause cache misses.

Clock cycle
The time between two adjacent pulses of the oscillator that sets the tempo of the computer processor. The cycle time is the reciprocal of the clock speed, or frequency. A 1 GHz (gigahertz) clock speed has a clock cycle time of 1 nanosecond (1 billionth of a second).

Clock speed
The system frequency of a processor. In modern processors the frequency is typically measured in GHz (gigahertz - 1 billion clocks per second). A 3 GHz processor producing 2 adds and 2 multiplies per clock cycle can achieve 12 Gflops.

Cluster system
A system of independent processing units, called blades or nodes, each having one or more independent processors and independent memory, usually configured in a separate chassis rack-mounted unit or as independent CPU boards. A cluster system uses some sort of interconnect to communicate between the independent nodes through a communication middleware application.

Core
A core is essentially an independent functioning processor that is part of a single multicore CPU. A dual-core processor contains two cores, and a quad-core processor contains four cores. Each core can run an individual application or run a process in parallel with other cores in a parallel application. Cores in the same multicore CPU share the same socket in which the CPU is plugged on a motherboard.

CPU time
As reported in the solver output, CPU time generally refers to the time that a processor spends on the user's application; it excludes system and I/O wait time and other idle time. For parallel systems, CPU time means different things on different systems. Some systems report CPU time summed across all threads, while others do not. It is best to focus on “elapsed” or “wall” time for parallel applications.

DANSYS
A shortcut name for Distributed ANSYS.

Database space
The block of memory that ANSYS uses to store the ANSYS database (model geometry, material properties, boundary conditions, and a portion of the results).

DIMM
A module (double in-line memory module) containing one or several random access memory (RAM) chips on a small circuit board with pins that connect to the computer motherboard.

Release 15.0 - © SAS IP, Inc. All rights reserved. - Contains proprietary and confidential information of ANSYS, Inc. and its subsidiaries and affiliates.

Distributed ANSYS

The distributed memory parallel (DMP) version of ANSYS. Distributed ANSYS can run over a cluster of machines or use multiple processors on a single machine (e.g., a desktop or workstation machine). It works by splitting the model into different parts during solution and distributing those parts to each machine/processor.

Distributed memory parallel (DMP) system

A system in which the physical memory for each process is separate from all other processes. A communication middleware application is required to exchange data between the processors.

Gflops

A measure of processor compute rate in terms of billions of floating point operations per second. 1 Gflop equals 1 billion floating point operations in one second.

Gigabit (abbreviated Gb)

A unit of measurement often used by switch and interconnect vendors. One gigabit = 1024x1024x1024 bits. Since a byte is 8 bits, it is important to keep units straight when making comparisons. Throughout this guide we use GB (gigabytes) rather than Gb (gigabits) when comparing both I/O rates and communication rates.

Gigabyte (abbreviated GB)

A unit of computer memory or data storage capacity equal to 1,073,741,824 (2^{30}) bytes. One gigabyte is equal to 1,024 megabytes (or 1,024 x 1,024 x 1,024 bytes).

Graphics processing unit (GPU)

A graphics processing unit (GPU) is a specialized microprocessor that offloads and accelerates 3-D or 2-D graphics rendering from the microprocessor. Modern GPUs are very efficient at manipulating computer graphics, and their highly parallel structure makes them more effective than general-purpose CPUs for a range of complex algorithms. In a personal computer, a GPU can be present on a video card, or it can be on the motherboard. Integrated GPUs are usually far less powerful than those on a dedicated video card.

Hyperthreading

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 does not increase the number of physical cores working on an application and is best suited for multicore systems running lightweight tasks that outnumber the number of physical cores available.

High performance computing (HPC)

The use of parallel processing software and advanced hardware (for example, large memory, multiple CPUs) to run applications efficiently, reliably, and quickly.

In-core mode

A memory allocation strategy in the shared memory and distributed memory sparse solvers that will attempt to obtain enough memory to compute and store the entire factorized matrix in memory. The purpose of this strategy is to avoid doing disk I/O to the matrix factor file.

Interconnect

A hardware switch and cable configuration that connects multiple cores (CPUs) or machines together.

Interconnect Bandwidth

The rate (MB/sec) at which larger-sized messages can be passed from one MPI process to another.

Release 15.0 - © SAS IP, Inc. All rights reserved. - Contains proprietary and confidential information of ANSYS, Inc. and its subsidiaries and affiliates.

Interconnect Latency
The measured time to send a message of zero length from one MPI process to another.

Master process
The first process started in a Distributed ANSYS run (also called the rank 0 process). This process reads the user input file, decomposes the problems, and sends the FEA data to each remaining MPI process in a Distributed ANSYS run to begin the solution. It also handles any pre- and postprocessing operations.

Megabit (abbreviated Mb)
A unit of measurement often used by switch and interconnect vendors. One megabit = 1024x1024 bits. Since a byte is 8 bits, it is important to keep units straight when making comparisons. Throughout this guide we use MB (megabytes) rather than Mb (megabits) when comparing both I/O rates and communication rates.

Megabyte (abbreviated MB)
A unit of computer memory or data storage capacity equal to 1,048,576 (2^{20}) bytes (also written as 1,024 x 1,024 bytes).

Memory bandwidth
The amount of data that the computer can carry from one point to another inside the CPU processor in a given time period (usually measured by MB/second).

Mflops
A measure of processor compute rate in terms of millions of floating point operations per second; 1 Mflop equals 1 million floating point operations in one second.

Multicore processor
An integrated circuit in which each processor contains multiple (two or more) independent processing units (cores).

MPI software
Message passing interface software used to exchange data among processors.

NFS
The Network File System (NFS) is a client/server application that lets a computer user view and optionally store and update files on a remote computer as if they were on the user's own computer. On a cluster system, an NFS system may be visible to all nodes, and all nodes may read and write to the same disk partition.

Node
When used in reference to hardware, a node is one machine (or unit) in a cluster of machines used for distributed memory parallel processing. Each node contains its own processors, memory, and usually I/O.

Non-Uniform Memory Architecture (NUMA)
A memory architecture for multi-processor/core systems that includes multiple paths between memory and CPUs/cores, with fastest memory access for those CPUs closest to the memory. The physical memory is globally addressable, but physically distributed among the CPU. NUMA memory architectures are generally preferred over a bus memory architecture for higher CPU/core counts because they offer better scaling of memory bandwidth.

OpenMP
A programming standard which allows parallel programming with SMP architectures. OpenMP consists of a software library that is usually part of the compilers used to build an application, along with a

Release 15.0 - © SAS IP, Inc. All rights reserved. - Contains proprietary and confidential information of ANSYS, Inc. and its subsidiaries and affiliates.

defined set of programing directives which define a standard method of parallelizing application codes for SMP systems.

Out-of-core mode
A memory allocation strategy in the shared memory and distributed memory sparse solvers that uses disk storage to reduce the memory requirements of the sparse solver. The very large matrix factor file is stored on disk rather than stored in memory.

Parallel processing
Running an application using multiple cores, or processing units. Parallel processing requires the dividing of tasks in an application into independent work that can be done in parallel.

Physical memory
The memory hardware (normally RAM) installed on a computer. Memory is usually packaged in DIMMS (double in-line memory module) which plug into memory slots on a CPU motherboard.

Primary compute node
In a Distributed ANSYS run, the machine or node on which the master process runs (that is, the machine on which the ANSYS job is launched). The primary compute node should not be confused with the host node in a Windows cluster environment. The host node typically schedules multiple applications and jobs on a cluster, but does not always actually run the application.

Processor
The computer hardware that responds to and processes the basic instructions that drive a computer.

Processor speed
The speed of a CPU (core) measured in MHz or GHz. See "clock speed" and "clock cycle."

RAID
A RAID (redundant array of independent disks) is multiple disk drives configured to function as one logical drive. RAID configurations are used to make redundant copies of data or to improve I/O performance by striping large files across multiple physical drives.

SAS drive
Serial-attached SCSI drive is a method used in accessing computer peripheral devices that employs a serial (one bit at a time) means of digital data transfer over thin cables. This is a newer version of SCSI drive found in some HPC systems.

SATA drive
Also known as Serial ATA, SATA is an evolution of the Parallel ATA physical storage interface. Serial ATA is a serial link; a single cable with a minimum of four wires creates a point-to-point connection between devices.

SCSI drive
The Small Computer System Interface (SCSI) is a set of ANSI standard electronic interfaces that allow personal computers to communicate with peripheral hardware such as disk drives, printers, etc.

Scalability
A measure of the ability of an application to effectively use parallel processing. Usually, scalability is measured by comparing the time to run an application on p cores versus the time to run the same application using just one core.

Release 15.0 - © SAS IP, Inc. All rights reserved. - Contains proprietary and confidential information of ANSYS, Inc. and its subsidiaries and affiliates.

Scratch space
The block of memory used by ANSYS for all internal calculations: element matrix formulation, equation solution, and so on.

Shared memory parallel (SMP) system
A system that shares a single global memory image that may be distributed physically across multiple nodes or processors, but is globally addressable.

SIMM
A module (single inline memory module) containing one or several random access memory (RAM) chips on a small circuit board with pins that connect to the computer motherboard.

Slave process
A Distributed ANSYS process other than the master process.

SMP ANSYS
Shared-memory version of ANSYS which uses a shared-memory architecture. Shared memory ANYS can use a single or multiple processors, but only within a single machine.

Socket configuration
A set of plug-in connectors on a motherboard that accepts CPUs. Each multicore CPU on a motherboard plugs into a separate socket. Thus, a dual socket CPU on a motherboard accepts two dual or quad core CPUs for a total of 4 or 8 cores. On a single mother board, the cores available are mapped to specific sockets and numbered within the CPU.

Solid state drive (SSD)
A solid-state drive (SSD) is a data storage device that uses solid-state memory to store data. Unlike traditional hard disk drives (HDDs), SSDs use microchips and contain no moving parts. Compared to traditional HDDs, SSDs are typically less susceptible to physical shock, quieter, and have lower access time and latency. SSDs use the same interface as hard disk drives, thus SSDs can easily replace HHDs in most applications.

Terabyte (abbreviated TB)
A unit of computer memory or data storage capacity equal to 1,099,511,627,776 (2^{40}) bytes. One terabyte is equal to 1,024 gigabytes.

Wall clock time
Total elapsed time it takes to complete a set of operations. Wall clock time includes processing time, as well as time spent waiting for I/O to complete. It is equivalent to what a user experiences in real-time waiting for the application to run.

Virtual memory
A portion of the computer's hard disk used by the system to supplement physical memory. The disk space used for system virtual memory is called swap space, and the file is called the swap file.

Release 15.0 - © SAS IP, Inc. All rights reserved. - Contains proprietary and confidential information of ANSYS, Inc. and its subsidiaries and affiliates.

Release 15.0 - © SAS IP, Inc. All rights reserved. - Contains proprietary and confidential information
of ANSYS, Inc. and its subsidiaries and affiliates.

Index

A

ANSYS performance
- computing demands, 1, 7
- examples, 51
- hardware considerations, 3
- measuring performance, 33
- memory usage, 15
- scalability of Distributed ANSYS, 27
- solver performance statistics, 47

B

Block Lanczos solver
- example, 53
- guidelines for improving performance, 58
- memory usage, 21
- performance output, 37

bus architecture, 3

C

contact elements - effect on scalability performance, 31

cores
- definition of, 3
- recommended number of, 8

CPU speed, 5

D

Distributed ANSYS
- examples, 59
- memory and I/O considerations, 59
- scalability of, 27

distributed memory parallel (DMP) processing, 7

distributed PCG solver, 31

distributed sparse solver, 31
- example, 60
- performance output, 35

E

eigensolver memory usage, 21

G

GPU accelerator, 8

GPUs
- effect on scalability, 30

graphics processing unit, 3, 8

H

hardware considerations, 3

hardware terms and definitions, 3

I

I/O
- considerations for HPC performance, 13
- effect on scalability, 29
- hardware, 11
- in a balanced HPC system, 5
- in ANSYS, 11

in-core factorization, 15

interconnects, 3
- effect on scalability, 28

M

measuring ANSYS performance
- Block Lanczos solver performance, 37
- distributed sparse solver performance, 35
- PCG Lanczos solver performance, 41
- PCG solver performance, 39
- sparse solver performance, 33
- summary for all solvers, 47
- Supernode solver performance, 46

memory
- considerations for parallel processing, 10
- effect on performance, 15
- in a balanced HPC system, 5
- limits on 32-bit systems, 10
- requirements for linear equation solvers, 15
- requirements within ANSYS, 9

memory allocation, 9

modal solver memory usage, 21

multicore processors, 3
- effect on scalability, 28

N

NUMA, 3

O

out-of-core factorization, 15

P

parallel processing
- in ANSYS, 8

partial pivoting, 15

PCG Lanczos solver
- memory usage, 21
- performance output, 41

PCG solver
- guidelines for improving performance, 64
- in Distributed ANSYS, 31
- memory usage, 19
- performance output, 39

Release 15.0 - © SAS IP, Inc. All rights reserved. - Contains proprietary and confidential information of ANSYS, Inc. and its subsidiaries and affiliates.

physical memory, 3

R

RAID array, 3

S

scalability
    definition of, 27
    effect of contact elements on, 31
    hardware issues, 28
    measuring, 27
    software issues, 30
shared memory parallel (SMP) processing, 7
SMP ANSYS
    examples, 51
solver performance statistics, 47
sparse solver
    example, 51
    memory usage, 15
    performance output, 33
Supernode solver
    memory usage, 21
    performance output, 46

V

virtual memory, 3

Release 15.0 - © SAS IP, Inc. All rights reserved. - Contains proprietary and confidential information of ANSYS, Inc. and its subsidiaries and affiliates.