Ansys
2025 R1
POWERING INNOVATION THAT DRIVES HUMAN ADVANCEMENT
® 2025 ANSYS, Inc. or its affiliated companies
Unauthorized use, distribution, or duplication is prohibited.

Parallel Processing Guide

ANSYS, Inc.
Southpointe
2600 Ansys Drive
Canonsburg, PA 15317
ansysinfo@ansys.com
http://www.ansys.com
(T) 724-746-3304
(F) 724-514-9494

Release 2025 R1
January 2025

ANSYS, Inc. and ANSYS Europe, Ltd. are UL registered ISO 9001: 2015 companies.

Table of Contents

List of Figures

List of Tables

Chapter 1: Overview of Parallel Processing

Solving a large model with millions of DOFs or a medium-sized model with nonlinearities that needs many iterations to reach convergence can require many CPU hours. To decrease simulation time, Ansys, Inc. offers different parallel processing options that increase the model-solving power of Ansys products by using multiple processors (p. 7) (also known as CPU cores). The following parallel processing capabilities are available:

- Shared-memory parallel processing (p. 11)
- Distributed-memory parallel processing (p. 23)
- Distributed + Shared-memory parallel processing (Hybrid parallel) (p. 55)
- GPU acceleration (a type of shared-memory parallel processing) (p. 15)

Multicore processors, and thus the ability to use parallel processing, are now widely available on all computer systems, from laptops to high-end servers. The benefits of parallel processing are compelling but are also among the most misunderstood. This chapter explains the types of parallel processing available in Ansys and also discusses the use of GPUs (considered a form of shared-memory parallel processing) and how they can further accelerate the time to solution.

Currently, the default scheme is to use four cores with distributed-memory parallelism. For many of the computations involved in a simulation, the speedups obtained from parallel processing are nearly linear as the number of cores is increased, making very effective use of parallel processing. However, the total benefit (measured by elapsed time) is problem dependent and is influenced by many different factors.

No matter what form of parallel processing is used, the maximum benefit attained will always be limited by the amount of work in the code that cannot be parallelized. If just 20 percent of the runtime is spent in nonparallel code, the maximum theoretical speedup is only 5X, assuming the time spent in parallel code is reduced to zero. However, parallel processing is still an essential component of any HPC system; by reducing wall clock elapsed time, it provides significant value when performing simulations.

Distributed-memory parallel (DMP) processing, shared-memory parallel (SMP) processing, hybrid parallel processing, and GPU acceleration can require HPC licenses. You can use up to four CPU cores or a combination of four CPUs and GPUs without using any HPC licenses. Additional licenses will be needed to run with more than four. See HPC Licensing (p. 9) for more information.

See the following sections for more details:

1.1. Specifying Processes, Threads, and Cores in SMP, DMP, and Hybrid Parallel

This section discusses the differences between the three types of parallel processing analyses available (shared-memory parallel (SMP) (p. 11), distributed-memory parallel (DMP) (p. 23), and hybrid (p. 55)) and how to specify values that determine the total cores used in each one. The main difference between SMP and DMP is that SMP shares the memory between threads in a single process while DMP distributes the memory across several processes running on different cores of a single machine or multiple machines. Hybrid parallel processing combines SMP and DMP.

You can specify the following values that determine the total core count for your analysis:

- the number of processes for DMP (p. 23)
- the number of threads per process for SMP (p. 11)
- both the number of processes and the number of threads per process for hybrid parallel (p. 55)

The rest of this section details how these values determine the total number of cores for the different types of parallel processing.

Figure 1.1: Schematic of Processes and Threads for DMP and Hybrid Parallel Processing (p. 3) shows a diagram comparing DMP and hybrid parallel processing. In distributed-memory parallel processing, while executing preprocessing and postprocessing tasks, only the master process uses multiple SMP threads. The worker processes are limited to one thread per process while executing solution tasks in parallel. In contrast, hybrid parallel enables multiple threads per process in all processes, master and worker, during solution. While not diagrammed, SMP shares the memory between threads in a single process and can be visualized as the master process alone without the worker processes in the figure.

Figure 1.1: Schematic of Processes and Threads for DMP and Hybrid Parallel Processing

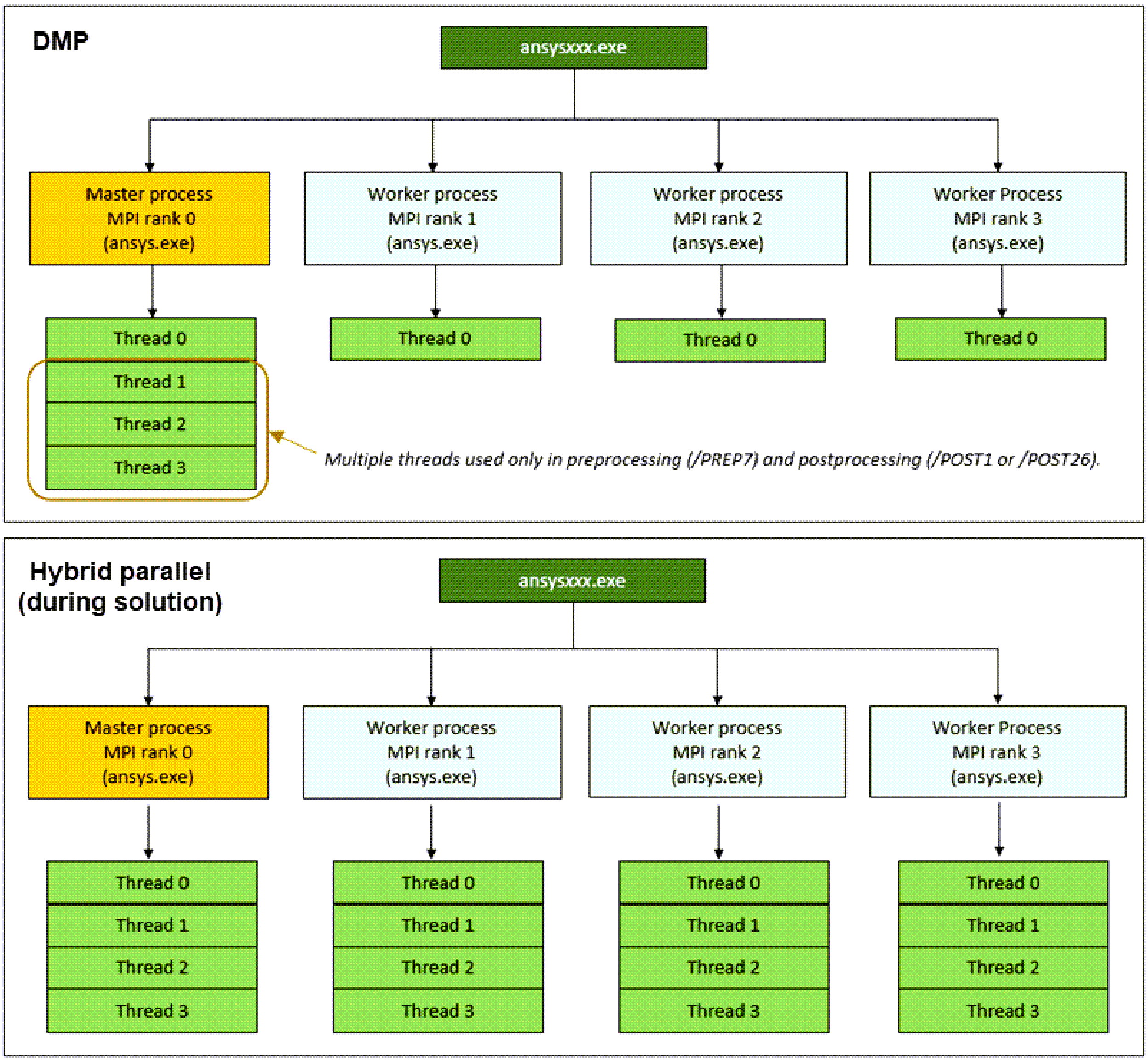


Table 1.1: Command Line Options to Specify Number of Processes and/or Threads per Process (p. 4) summarizes the command line options used to specify the type of parallel computing (SMP, DMP, or hybrid), the number of processes, and/or the number of threads per process. The following equation clarifies how these values relate to the total core count. For the definitions of these and other terms used in parallel processing, see Parallel Processing Terminology (p. 6).

$$P \times T_p = C \tag{1.1}$$

where:

P is the number of processes,

T_p is the number of threads per process, and

C is the total core count for the analysis.

Note:

To avoid oversubscribing the hardware, the number of threads per core is always one. Also, the total cores prescribed must be equal to or less than actual available cores (p. 5).

Since the number of processes for SMP is one by definition, the core counts for SMP by Equation 1.1 (p. 3) is simply equal to the number of threads per process, T_p, which you can specify by command line option `-np`.

As diagrammed in Figure 1.1: Schematic of Processes and Threads for DMP and Hybrid Parallel Processing (p. 3), DMP uses only one thread per process during solution. Therefore, by Equation 1.1 (p. 3), core counts for DMP is simply equal to the number of processes, P, which you can specify by command line option `-np`.

Table 1.1: Command Line Options to Specify Number of Processes and/or Threads per Process

parallel processing type	number of processes (P in Equation 1.1 (p. 3))	threads per process (T_p in Equation 1.1 (p. 3))
Shared-Memory Parallel (SMP)[a]	1	specified by command line option `-np` (default is 4)
Distributed-Memory Parallel (DMP)[b]	specified by command line option `-np` (default is 4)	1 [c]
Hybrid Parallel[d]	specified by command line option `-np` (default is 4)	specified by command line option `-nt` [d] (default is 1)

[a] Specify SMP by command line option `-smp`.

[b] Since DMP is the default parallel processing type, the `-dis` line option for DMP is not required.

[c] Since 1 is the default, it does not need to be set unless you want to turn off auto-hybrid (p. 5). To turn off auto-hybrid logic and force a DMP run, issue the command line option `-nt 1`.

[d] To invoke hybrid parallel, set `-nt` to a number greater than its default value of 1.

Note:

The meaning of the command line option `-np` is different for SMP compared to DMP and hybrid as listed in Table 1.1: Command Line Options to Specify Number of Processes and/or Threads per Process (p. 4):

- For SMP, `-np` specifies threads/process.

- For DMP and hybrid, `-np` specifies the number of processes.

To summarize how the total core count is determined for the different parallel processing types:

- For SMP and DMP, the total core count is equal to the number entered after the `-np` command line option, which is the number of threads/process for SMP and number of processes for DMP.
- For hybrid, the total core count is $P \times T_p$, or the product of the number of processes (that you can specify by command line option `-np`) multiplied by the number of threads/process (specified by command line option `-nt`). For example, the following command:

  ```
  ansys251 -b -np 4 -nt 4 <test.dat> out
  ```

  specifies a hybrid parallel run using 4 MPI processes with 4 threads/process, and the total core count is 16.

  If hybrid parallel is invoked with the `-machines` command line option, the value specified for `-nt` multiplies to all of the requested core counts on each machine. For example:

  ```
  ansys252 -b -nt 4 -machines machine1:8:machine2:8
  ```

  results in 8 MPI processes with 4 processes per thread on each machine. Therefore, the total core count is 8 x 4 x 2 = 64.

  Note that the number of threads per process is determined by the number of MPI ranks launched on the compute node, up to a maximum of 16 threads.

Total cores prescribed must be equal to or less than actual available cores

In all cases (SMP, DMP, and hybrid), the parallel processing run is limited by the actual number of physical cores available[1]. If the number of cores prescribed exceeds the number of physical cores available[1], the number of threads/processes will be automatically reduced for SMP, DMP, and hybrid so that the total cores calculated by Equation 1.1 (p. 3) do not exceed the physical CPU cores available.

1.1.1. Auto-hybrid

If the specifics of your analysis indicate that using hybrid parallel will improve performance, the program will automatically invoke hybrid parallel. The decision to activate auto-hybrid is made according to a set of heuristics at the beginning of the first load step solution, just before the domain decomposition step is performed. The auto-hybrid feature is on by default as long as no `–nt` value is provided on the command line. If auto-hybrid is invoked during solution:

- The program automatically sets values for `-nt` and `-np`, keeping the same number for the total cores you prescribed for your analysis, and the solution runs in hybrid parallel. Note that the program specifies an `-nt` value per process. This may result in one or more domains utilizing additional threads while other domains use a single thread. There is no mechanism available to specify an unbalanced number of threads per process so that some processes (or domains) have more or less threads than other processes (or domains).

[1] Note that additional licenses are required to run a parallel processing solution with more than four cores in all cases (SMP, DMP, and hybrid). See HPC Licensing (p. 9) for details.

- An output message will report that hybrid parallel has been automatically invoked with the values used for the number of processes (`-np`) and threads per process (`-nt`).

To Turn off Auto-hybrid

If you specify a value for the number of threads/process by issuing the `-nt` command line option, the auto-hybrid feature is shut off. For example, if you are sure that you want an analysis to run using pure DMP with only one thread/process during solution, you can turn off auto-hybrid via the command line option to specify one thread per process: `-nt 1`. Likewise, you can activate a uniform two threads per process during solution with `-nt 2`. When specified on the command line, the single `-nt` value is applied to all processes.

1.2. Parallel Processing Terminology

It is important to fully understand the terms we use, both relating to our software and to the physical hardware. The terms *shared-memory parallel (SMP) processing* and *distributed-memory parallel (DMP) processing* refer to our software offerings, which run on shared-memory or distributed-memory hardware configurations. The term *GPU accelerator capability* refers to our software offering which allows the program to take advantage of certain GPU (graphics processing unit) hardware to accelerate the speed of the solver computations.

1.2.1. Hardware Terminology

The following terms describe the hardware configurations used for parallel processing:

Shared-memory hardware	This term refers to a physical hardware configuration in which a single shared-memory address space is accessible by multiple CPU cores; each CPU core "shares" the memory with the other cores. Common examples of a shared-memory system is any single machine: a Windows desktop machine, a laptop, or a workstation with one or two multicore processors.
Distributed-memory hardware	This term refers to a physical hardware configuration in which multiple machines are connected together on a network (that is, a cluster). Each machine on the network (that is, each compute node on the cluster) has its own memory address space. Communication between machines is handled by interconnects (Gigabit Ethernet, Infiniband, etc.). Virtually all clusters involve both shared-memory and distributed-memory hardware. Each compute node on the cluster typically contains at least two or more CPU cores, which means there is a shared-memory environment within a compute node. The distributed-memory environment requires communication between the compute nodes involved in the cluster.
GPU hardware	A graphics processing unit (GPU) is a specialized microprocessor that off-loads and accelerates graphics rendering from the microprocessor. Their highly parallel structure makes GPUs more effective than general-purpose CPUs for a range of complex algorithms. In

	a personal computer, a GPU on a dedicated video card is more powerful than a GPU that is integrated on the motherboard.
Processor	Also called the central processing unit (CPU) or microprocessor chip, the processor is the part of a computer that interprets and executes instructions. In the past, processors generally had just one CPU core, so it unambiguously referred to a chip that is inserted into a socket. With the advent of multi-core processors, which are common today, the term "processor" has become ambiguous since it is used as a synonym for CPU core and also to refer to the chip itself, which typically has multiple CPU cores. To avoid confusion, the term "core" is used rather than "processor" throughout this guide.
CPU Core	A CPU core (also called, more simply, "core") is a distinct physical component of the CPU that performs tasks.
Hyperthreading	An Intel hardware innovation by which a CPU divides up its physical cores into "virtual or logical cores" that are treated as if they are actually physical cores by the operating system. When hyperthreading is active, the CPU exposes two execution contexts per physical core so that one physical core now works like two "virtual or logical cores" that can handle different software threads (p. 8). To avoid oversubscribing hardware, "virtualized cores" are not used.
Head compute node	In a distributed-memory parallel (DMP) run, the machine or node on which the master process runs (that is, the machine on which the job is launched). The head 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 typically run the application.

1.2.2. Software Terminology

The following terms describe our software offerings for parallel processing:

Shared-memory parallel (SMP) processing	This term refers to running across multiple cores on a single machine (for example, a desktop workstation or a single compute node of a cluster). Shared-memory parallelism is invoked, which allows each core involved to share data (or memory) as needed to perform the necessary parallel computations. When run within a shared-memory architecture, most computations in the solution phase and many pre- and postprocessing operations are performed in parallel. For more information, see Using Shared-Memory Parallel (SMP) Processing (p. 11).
Distributed-memory parallel (DMP) processing	This term refers to running across multiple cores on a single machine (for example, a desktop workstation or a single compute node of a cluster) or across multiple machines (for example, a cluster). Distributed-memory parallelism is invoked, and each process communicates data needed to perform the necessary

	parallel computations through the use of MPI (Message Passing Interface) software. With distributed-memory parallel processing, all computations in the solution phase are performed in parallel (including the stiffness matrix generation, linear equation solving, and results calculations). Pre- and postprocessing do not make use of the distributed-memory parallel processing, but these steps can exploit shared-memory parallelism. See Using Distributed-Memory Parallel (DMP) Processing (p. 23) for more details.
Hybrid parallel processing	This term refers to running across multiple cores on a single machine (for example, a desktop workstation or a single compute node of a cluster) or across multiple machines (for example, a cluster) using a combination of distributed-memory parallelism and shared-memory parallelism. With hybrid parallel, all computations in the solution phase are performed in parallel (including the stiffness matrix generation, linear equation solving, and results calculations). Pre- and postprocessing do not make use of distributed-memory parallel processing, but these steps can make use of shared-memory parallelism. See Using Hybrid Parallelism (p. 55) for more details.
GPU accelerator capability	This capability takes advantage of the highly parallel architecture of the GPU hardware to accelerate the speed of solver computations and, therefore, reduce the time required to complete a simulation. Some computations of certain equation solvers can be off-loaded from the CPU(s) to the GPU, where they are often executed much faster. The CPU core(s) will continue to be used for all other computations in and around the equation solvers. For more information, see GPU Accelerator Capability (p. 15).
Parallel processing	Parallel processing is a method in computing of running two or more CPU cores to handle separate parts of an overall task. Breaking up different parts of a task among multiple cores helps reduce the amount of time to run a program.
Processes	Also referred to as Mechanical APDL™ processes, Ansys Processes, MPI processes, or MPI ranks, processes are the execution of instructions to perform smaller tasks, which comprise the overall task.
Master process	The first process launched on the head compute node in a distributed-memory parallel processing run.
Worker process	A distributed-memory parallel processing process other than the master process.
Thread	A software thread is a unit of execution on concurrent programming. It is the virtual component which manages the tasks of the cores (p. 7).

Shared-memory parallel (SMP) processing analyses can only be run on shared-memory hardware (a single machine). However, distributed-memory parallel (DMP) analyses can be run on both shared-memory hardware (a single machine) or distributed-memory hardware (a cluster). While both forms of hardware can achieve a significant speedup, with DMP processing, only running on distributed-memory hardware allows you to take advantage of increased resources (for example, available memory and disk space, as well as memory and I/O bandwidths) by using multiple machines. The GPU accelerator capability can be used with either SMP or DMP.

1.3. HPC Licensing

Ansys, Inc. offers the following high performance computing license options:

Ansys HPC - These physics-neutral licenses can be used to run a single analysis across multiple cores .

Ansys HPC Packs - These physics-neutral licenses share the same characteristics of the Ansys HPC licenses but are combined into predefined packs to give you greater value and scalability.

For detailed information on these HPC license options, see HPC Licensing in the *Ansys, Inc. Licensing Guide*.

The order in which HPC licenses are used is specified by your user license preferences setting. See Setting HPC User Preferences in the *Ansys, Inc. Licensing Guide* for more information on setting user license product order.

You can choose a particular HPC license by using the Preferred Parallel Feature command line option. The format is `ansys251 -ppf` <*`license feature name`*>, where <*`license feature name`*> is the name of the HPC license option that you want to use. This option forces Mechanical APDL to use the specified license feature for the requested number of parallel cores or GPUs. If the license feature is entered incorrectly or the license feature is not available, a license failure occurs.

All types of parallel processing (DMP, SMP, and hybrid) allow you to use four CPU cores without using any HPC licenses. Ansys HPC licenses add cores to this base functionality, while the Ansys HPC Pack licenses function independently of the four included cores.

In a similar way, you can use up to four CPU cores and GPUs combined without any HPC licensing (for example, one CPU and three GPUs). The combined number of CPU cores and GPUs used cannot exceed the task limit allowed by your specific license configuration.

Chapter 2: Using Shared-Memory Parallel (SMP) Processing

When running a simulation, the solution time is typically dominated by three main parts: the time spent to create the element matrices and form the global matrices, the time to solve the linear system of equations, and the time spent calculating derived quantities (such as stress and strain) and other requested results for each element.

Shared-memory parallel (SMP) processing can run a solution over multiple cores on a single machine. When using shared-memory parallel processing, you can reduce each of the three main parts of the overall solution time by using multiple cores. However, this approach is often limited by the memory bandwidth; you typically see very little reduction in solution time beyond four cores.

The main program functions that run in parallel on shared-memory hardware are:

- Solvers such as the Sparse, Mixed, PCG, Block Lanczos, PCG Lanczos, Supernode, and Subspace running over multiple cores but sharing the same memory address. These solvers typically have limited scalability when used with shared-memory parallelism. In general, very little reduction in time occurs when using more than four cores.
- Forming element matrices and load vectors.
- Computing derived quantities and other requested results for each element.
- Pre- and postprocessing functions such as graphics, selecting, sorting, and other data and compute intensive operations.

2.1. Activating Parallel Processing in a Shared-Memory Architecture

1. By default, SMP processing uses four cores and does not require any HPC licenses. Additional HPC licenses are required to run with more than four cores. Several HPC license options are available. See HPC Licensing (p. 9) for more information.
2. Open the Mechanical APDL Product Launcher:

   Windows:

   Start >Programs >Ansys 2025 R1 >Mechanical APDL Product Launcher

   Linux:

   ```
   launcher251
   ```

3. Select the correct environment and license.

4. Go to the **High Performance Computing Setup** tab. Select **Use Shared-Memory Parallel (SMP)**. Specify the number of threads (or CPU cores) to use.

5. Alternatively, you can specify the number of cores to use via the `-np` command line option:

   ```
   ansys251 -smp -np N
   ```

   where *N* represents the number of threads (or CPU cores) to use. For a general discussion on processes, threads, and cores in parallel processing, see Specifying Processes, Threads, and Cores in SMP, DMP, and Hybrid Parallel (p. 2) in the overview section.

   For large multiprocessor servers, Ansys, Inc. recommends setting *N* to a value no higher than the number of available cores *minus one*. For example, on an eight-core system, set *N* to 7. However, on multiprocessor workstations, you may want to use all available cores to minimize the total solution time. The program automatically limits the maximum number of cores used to be less than or equal to the number of physical cores on the machine. This is done to avoid running the program on virtual cores (for example, by means of hyperthreading), which typically results in poor per-core performance. For optimal performance, consider closing down all other applications before launching Ansys.

   If you have more than one HPC license feature, you can use the `-ppf` command line option to specify which HPC license to use for the parallel run. See HPC Licensing (p. 9) for more information.

6. If working from the launcher, click **Run** to launch Ansys.

7. Set up and run your analysis as you normally would.

2.1.1. System-Specific Considerations

For shared-memory parallel processing, the number of cores that the program uses is limited to the *lesser* of one of the following:

- The number of Ansys HPC licenses available (plus the first four cores which do not require any licenses)
- The number of cores indicated via the `-np` command line argument
- The actual number of cores available

You can specify multiple settings for the number of cores to use during a session. However, Ansys, Inc. recommends that you issue the **/CLEAR** command before resetting the number of cores for subsequent analyses.

2.2. Troubleshooting

This section describes problems which you may encounter while using shared-memory parallel processing as well as methods for overcoming these problems. Some of these problems are specific to a particular system, as noted.

Job fails with SIGTERM signal (Linux Only)

Occasionally, when running on Linux, a simulation may fail with the following message: "process killed (SIGTERM)". This typically occurs when computing the solution and means that the system has killed the Ansys process. The two most common occurrences are (1) Ansys is using too much of the hardware resources and the system has killed the Ansys process or (2) a user has manually killed the Ansys job (that is, **kill -9** system command). Users should check the size of job they are running in relation to the amount of physical memory on the machine. Most often, decreasing the model size or finding a machine with more RAM will result in a successful run.

Poor Speedup or No Speedup

As more cores are utilized, the runtimes are generally expected to decrease. The biggest relative gains are typically achieved when using two cores compared to using a single core. When significant speedups are not seen as additional cores are used, the reasons may involve both hardware and software issues. These include, but are not limited to, the following situations.

Hardware

Oversubscribing hardware

In a multiuser environment, this could mean that more physical cores are being used by Ansys simulations than are available on the machine. It could also mean that hyperthreading is activated. Hyperthreading typically involves enabling extra virtual cores, which can sometimes allow software programs to more effectively use the full processing power of the CPU. However, for compute-intensive programs such as Ansys, using these virtual cores rarely provides a significant reduction in runtime. Therefore, it is recommended you disable hyperthreading; if hyperthreading is enabled, it is recommended you do not exceed the number of physical cores.

Lack of memory bandwidth

On some systems, using most or all of the available cores can result in a lack of memory bandwidth. This lack of memory bandwidth can affect the overall scalability of the Ansys software.

Dynamic Processor Speeds

Many new CPUs have the ability to dynamically adjust the clock speed at which they operate based on the current workloads. Typically, when only a single core is being used the clock speed can be significantly higher than when all of the CPU cores are being utilized. This can have a negative effect on scalability as the per-core computational performance can be much higher when only a single core is active versus the case when all of the CPU cores are active.

Software

Simulation includes non-supported features

The shared- and distributed-memory parallelisms work to speed up certain compute-intensive operations in **/PREP7**, **/SOLU** and **/POST1**. However, not all operations are parallelized. If a particular operation that is not parallelized dominates the simulation time, then using additional cores will not help achieve a faster runtime.

Simulation has too few DOF (degrees of freedom)

Some analyses (such as transient analyses) may require long compute times, not because the number of DOF is large, but because a large number of calculations are performed (that is, a very large number of time steps). Generally, if the number of DOF is relatively small, parallel processing will not significantly decrease the solution time. Consequently, for small models with many time steps, parallel performance may be poor because the model size is too small to fully utilize a large number of cores.

I/O cost dominates solution time

For some simulations, the amount of memory required to obtain a solution is greater than the physical memory (that is, RAM) available on the machine. In these cases, either virtual memory (that is, hard disk space) is used by the operating system to hold the data that would otherwise be stored in memory, or the equation solver writes extra files to the disk to store data. In both cases, the extra I/O done using the hard drive can significantly affect performance, making the I/O performance the main bottleneck to achieving optimal performance. In these cases, using additional cores will typically not result in a significant reduction in overall time to solution.

Different Results Relative to a Single Core

Shared-memory parallel processing occurs in various preprocessing, solution, and postprocessing operations. Operational randomness and numerical round-off inherent to parallelism can cause slightly different results between runs on the same machine using the same number of cores or different numbers of cores. This difference is often negligible. However, in some cases the difference is appreciable. This sort of behavior is most commonly seen on nonlinear static or transient analyses which are numerically unstable. The more numerically unstable the model is, the more likely the convergence pattern or final results will differ as the number of cores used in the simulation is changed.

With shared-memory parallelism, you can use the **PSCONTROL** command to control which operations actually use parallel behavior. For example, you could use this command to show that the element matrix generation running in parallel is causing a nonlinear job to converge to a slightly different solution each time it runs (even on the same machine with no change to the input data). This can help isolate parallel computations which are affecting the solution while maintaining as much other parallelism as possible to continue to reduce the time to solution.

Chapter 3: GPU Accelerator Capability

In an effort to provide faster performance during solution, Mechanical APDL supports offloading key solver computations onto graphics cards to accelerate those computations. Only high-end graphics cards, the ones with the most amount of cores and memory, can be used to accelerate the solver computations. For details on which GPU devices are supported and the corresponding driver versions, see the GPU requirements outlined in the Windows Installation Guide and the Linux Installation Guide.

It is important to understand that a GPU does not replace the CPU core(s) on which a simulation typically runs. One or more CPU cores must be used to run the Mechanical APDL program. The GPUs are used in support of the CPU to process certain calculations. The CPU continues to handle most operations and will automatically offload some of the time-intensive parallel operations performed by certain equation solvers. These parallel solver operations can usually be performed much faster on the highly parallel architecture of a GPU, thus accelerating these solvers and reducing the overall time to solution.

GPU acceleration can be used with both shared-memory parallel (SMP) processing and distributed-memory parallel (DMP) processing. In SMP processing, one or multiple GPU accelerator devices can be utilized during solution. In DMP processing, one or multiple GPU accelerator devices *per machine or compute node* can be utilized during solution.

As an example, when using DMP processing on a cluster involving eight compute nodes with each compute node having two supported GPU accelerator devices, either a single GPU per node (a total of eight GPU cards) or two GPUs per node (a total of sixteen GPU cards) can be used to accelerate the solution. The GPU accelerator device usage must be consistent across all compute nodes. For example, if running a simulation across all compute nodes, it is not possible to use one GPU for some compute nodes and zero or two GPUs for the other compute nodes.

On machines containing multiple GPU accelerator devices, the program automatically selects the GPU accelerator device (or devices) to be used for the simulation. The program cannot detect if a GPU device is currently being used by other software, including another Mechanical APDL simulation. Therefore, in a multiuser environment, users should be careful not to oversubscribe the GPU accelerator devices by simultaneously launching multiple simulations that attempt to use the same GPU (or GPUs) to accelerate the solution. For more information, see Oversubscribing GPU Hardware (p. 20) in the troubleshooting discussion.

The GPU accelerator capability is supported on Windows 64-bit and Linux x64 platforms.

Note, some GPU cards may contain multiple GPUs in them, and a GPU may contain multiple GPU compute engines. HPC licensing for Mechanical APDL is per CPU core and GPU (not per GPU compute engine). You can use up to four GPUs and CPU cores combined without any HPC licensing (for example, one CPU core and three GPUs). To use more than four, you need one or more Ansys HPC licenses or Ansys HPC Pack licenses. For more information see HPC Licensing in the *Ansys, Inc. Licensing Guide*.

The following GPU accelerator topics are available:

3.1. Activating the GPU Accelerator Capability

Following is the general procedure to use the GPU accelerator capability:

1. Before activating the GPU accelerator capability, you must have at least one GPU card installed with the proper driver level. If you have more than four GPUs or CPU cores combined, you will also need some type of HPC license; see HPC licensing for details.

2. Open the Mechanical APDL Product Launcher.

   Windows:

   Start >Programs >Ansys 2025 R1 >Mechanical APDL Product Launcher

   Linux[1]**:**

   ```
   launcher251
   ```

3. Select the correct environment and license.

4. Go to the **High Performance Computing Setup** tab, select a GPU device from the **GPU Accelerator** drop-down menu, and specify the number of GPU accelerator devices.

5. Alternatively, you can activate the GPU accelerator capability via the `-acc` command line option[1]:

   ```
   ansys251 -acc nvidia -na N
   ```

   or

   ```
   ansys251 -acc amd -na N
   ```

   The `-na` command line option followed by a number (*N*) indicates the number of GPU accelerator devices to use per machine or compute node. If only the `-acc` option is specified, the program uses a single GPU device per machine or compute node by default (that is, `-na 1`).

   If you have more than one HPC license feature, you can use the `-ppf` command line option to specify which HPC license to use for the parallel run. See HPC Licensing (p. 9) for more information.

6. If working from the launcher, click Run to launch Mechanical APDL.

7. Set up and run your analysis as you normally would.

With the GPU accelerator capability, the acceleration obtained by using the parallelism on the GPU hardware occurs only during the solution operations. Operational randomness and numerical round-off inherent to any parallel algorithm can cause slightly different results between runs on the same machine when using or not using the GPU hardware to accelerate the simulation.

[1] When using GPU acceleration on a Linux machine, the Mechanical APDL graphical user interface (GUI) is not supported.

The **ACCOPTION** command can also be used to control activation of the GPU accelerator capability.

3.2. Supported GPU Hardware, Analysis Types, and Features

Some analysis types and features are not supported by the GPU accelerator capability. Supported functionality also depends on the specified GPU hardware. The following section gives general guidelines on what is and is not supported.

These are not comprehensive lists, but represent major features and capabilities found in the Mechanical APDL program.

3.2.1. GPU Hardware

For a list of recommended GPU devices, see the Windows Installation Guide and the Linux Installation Guide.

3.2.2. Supported Analysis Types

The following analysis types are supported and will use the GPU to accelerate the solution.

- Static linear or nonlinear analyses using the sparse, Preconditioned Conjugate Gradient (PCG), Jacobi Conjugate Gradient (JCG), or mixed solver.
- Buckling analyses using the Block Lanczos or subspace eigensolver.
- Modal analyses using the Block Lanczos, subspace, PCG Lanczos, QR damped, unsymmetric, or damped eigensolver.
- Harmonic analyses using the full method and the sparse solver.
- Transient linear or nonlinear analyses using the full method and the sparse, PCG, JCG, or mixed solver.
- Substructuring analyses, generation pass only, including the generation pass of component mode synthesis (CMS) analyses.

In situations where the analysis type is not supported by the GPU accelerator capability, the solution will continue but GPU acceleration will not be used.

Performance Issue for Some Solver/Hardware Combinations

When using the PCG , JCG, or mixed solver, or the PCG Lanczos eigensolver, any of the recommended GPU devices can be expected to achieve good performance.

When using the sparse solver or eigensolvers based on the sparse solver (for example, Block Lanczos or subspace), only GPU devices with significant double precision performance (FP64) are recommended in order to achieve good performance. For a list of these devices, see the Windows Installation Guide and the Linux Installation Guide.

When using AMD GPUs and DMP processing, less than eight MPI processes per GPU is recommended.

Shared-Memory Parallel Behavior

For the sparse solver, mixed solver, (and eigensolvers based on the sparse solver), if one or more GPUs are requested, only a single GPU is used no matter how many are requested.

For the PCG and JCG solvers (and eigensolvers based on the PCG solver), all requested GPUs are used.

When using AMD GPUs with the PCG and JCG solvers (and eigensolvers based on the PCG solver) on Linux, only one GPU is used.

Distributed-Memory Parallel Behavior

For the sparse solver (and eigensolvers based on the sparse solver), if the number of GPUs exceeds the number of processes (the `-na` value is greater than the `-np` value on the command line), the number of GPUs used equals the `-np` value. If the number of GPUs is less than the number of processes (`-na` is less than `-np`), all requested GPUs are used.

For the PCG and JCG solvers (and eigensolvers based on the PCG solver), if the number of GPUs exceeds the number of processes (`-na` is greater than `-np`), all requested GPUs are used. If the number of GPUs is less than the number of processes (`-na` is less than `-np`), all requested GPUs are used.

When using AMD GPUs with the PCG and JCG solvers (and eigensolvers based on the PCG solver) on Linux, at most the `-np` value of GPUs is used.

3.2.3. Supported Features

As the GPU accelerator capability currently only pertains to the equation solvers, virtually all features and element types are supported when using this capability with the supported equation solvers listed in Supported Analysis Types (p. 17). A few limitations exist and are listed below. In these situations, the solution will continue but GPU acceleration will not be used (unless otherwise noted):

- Partial pivoting is activated when using the sparse solver. This most commonly occurs when using current technology elements with mixed u-P formulation, Lagrange multiplier based MPC184 elements, Lagrange multiplier based contact elements (TARGE169 through CONTA178), or certain circuit elements (CIRCU94, CIRCU124).
- The memory saving option is activated (**MSAVE**,ON) when using the PCG solver. In this particular case, the **MSAVE** option is turned off and GPU acceleration is used.

3.3. Troubleshooting

This section describes problems which you may encounter while using the GPU accelerator capability, as well as methods for overcoming these problems. Some of these problems are specific to a particular system, as noted.

GPU devices support various compute modes (for example, Exclusive thread, Exclusive process). Only the default compute mode is supported. Using other compute modes may cause the program to fail to launch.

To list the GPU devices installed on the machine, set the **ANSGPU_PRINTDEVICES** environment variable to a value of 1. The printed list may or may not include graphics cards used for display purposes, along with any graphics cards used to accelerate your simulation.

No Devices

Be sure that a recommended GPU device is properly installed and configured. Check the driver level to be sure it is current or newer than the driver version supported for your particular device. (See the GPU requirements outlined in the Windows Installation Guide and the Linux Installation Guide.)

When using GPU devices, use of the CUDA_VISIBLE_DEVICES environment variable can block some or all of the GPU devices from being visible to the program. Try renaming this environment variable to see if the supported devices can be used.

Note:

On Windows, the use of Remote Desktop may disable the use of a GPU device. Launching Mechanical APDL through the Ansys Remote Solve Manager (RSM) when RSM is installed as a service may also disable the use of a GPU. In these two scenarios, the GPU Accelerator Capability cannot be used. Using the TCC (Tesla Compute Cluster) driver mode, if applicable, can circumvent this restriction.

No Valid Devices

A GPU device was detected, but it is not a recommended GPU device. Be sure that a recommended GPU device is properly installed and configured. Check the driver level to be sure it is current or newer than the supported driver version for your particular device. (See the GPU requirements outlined in the Windows Installation Guide and the Linux Installation Guide.) Consider using the ANSGPU_OVERRIDE environment variable to override the check for valid GPU devices.

When using GPU devices, use of the CUDA_VISIBLE_DEVICES environment variable can block some or all of the GPU devices from being visible to the program. Try renaming this environment variable to see if the supported devices can be used.

Poor Acceleration or No Acceleration

Simulation includes non-supported features

A GPU device will only accelerate certain portions of a simulation, mainly the solution time. If the bulk of the simulation time is spent outside of solution, the GPU cannot have a significant effect on the overall analysis time. Even if the bulk of the simulation is spent inside solution, you must be sure that a supported equation solver is utilized during solution and that no unsupported options are used. Messages are printed in the output to alert users when a GPU is being used, as well as when unsupported options/features are chosen which deactivate the GPU accelerator capability.

Simulation has too few DOF (degrees of freedom)

Some analyses (such as transient analyses) may require long compute times, not because the number of DOF is large, but because a large number of calculations are performed (that is, a very large number of time steps). Generally, if the number of DOF is relatively small, GPU acceleration will not significantly decrease the solution time. Consequently, for small models with many time steps, GPU acceleration may be poor because the model size is too small to fully utilize a GPU.

Simulation does not fully utilize the GPU

Only simulations that spend a lot of time performing calculations that are supported on a GPU can expect to see significant speedups when a GPU is used. Only certain computations are supported for GPU acceleration. Therefore, users should check to ensure that a high percentage of the solution time was spent performing computations that could possibly be accelerated on a GPU. This can be done by reviewing the equation solver statistics files as described below. See Measuring Performance in the *Performance Guide* for more details on the equation solver statistics files.

- **PCG solver file:** The `.PCS` file contains statistics for the PCG iterative solver. You should first check to make sure that the GPU was utilized by the solver. This can be done by looking at the line which begins with: "Number of cores used". The string "GPU acceleration enabled" will be added to this line if the GPU hardware was used by the solver. If this string is missing, the GPU was not used for that call to the solver. Next, you should study the elapsed times for both the "Preconditioner Factoring" and "Multiply With A22" computations. GPU hardware is only used to accelerate these two sets of computations. The wall clock (or elapsed) times for these computations are the areas of interest when determining how much GPU acceleration is achieved.

- **Sparse solver files:** The `.dsp` file contains statistics for the sparse direct solver. You should first check to make sure that the GPU was utilized by the solver. This can be done by looking for the following line: "GPU acceleration activated". This line will be printed if the GPU hardware was used. If this line is missing, the GPU was not used for that call to the solver. Next, you should check the percentage of factorization computations (flops) which were accelerated on a GPU. This is shown by the line: "percentage of GPU accelerated flops". Also, you should look at the time to perform the matrix factorization, shown by the line: "time (cpu & wall) for numeric factor". GPU hardware is only used to accelerate the matrix factor computations. These lines provide some indication of how much GPU acceleration is achieved.

- **Eigensolver files:** The Block Lanczos and Subspace eigensolvers support the use of GPU devices; however, no statistics files are written by these eigensolvers. The `.PCS` file is written for the PCG Lanczos eigensolver and can be used as described above for the PCG iterative solver.

- **Mixed solver files:** The mixed solver produces two files: a `.PCS` file and a `.dsp` file (see descriptions above).

Using multiple GPU devices

When using the sparse solver in a shared-memory parallel solution, it is expected that running a simulation with multiple GPU devices will not improve performance compared to running with a single GPU device. In a shared-memory parallel solution, the sparse solver can only make use of one GPU device.

Oversubscribing GPU hardware

The program automatically determines which GPU devices to use. In a multiuser environment, this could mean that one or more of the same GPUs are picked when multiple simulations are run simultaneously, thus oversubscribing the hardware.

- If only a single GPU accelerator device exists in the machine, then only a single user should attempt to make use of it, much in the same way users should avoid oversubscribing their CPU cores.
- If multiple GPU accelerator devices exist in the machine, you can set the **ANSGPU_DEVICE** environment variable, in conjunction with the **ANSGPU_PRINTDEVICES** environment variable mentioned above, to specify which particular GPU accelerator devices to use during the solution.

  For example, consider a scenario where **ANSGPU_PRINTDEVICES** shows that four GPU devices are available with device ID values of 1, 3, 5, and 7 respectively, and only the second and third devices are supported for GPU acceleration. To select only the second supported GPU device, set **ANSGPU_DEVICE** = 5. To select the first and second supported GPU devices, set **ANSGPU_DEVICE** = 3:5.

Error Code 100 for AMD GPU devices

When using AMD GPU devices, if you encounter the following error,

```
Error code = 100 which translates to: no ROCm-capable device is detected.
Please check your GPU device driver level and verify that a supported GPU device has been installed correctly.
```

add your user name (*LOGNAME*) to the list of users in the video group using the following command.

```
sudo usermod -a -G video $LOGNAME
```

Solver/hardware combination

When using GPU devices, some solvers may not achieve good performance on certain devices. For more information, see Performance Issue for Some Solver/Hardware Combinations (p. 17).

Chapter 4: Using Distributed-Memory Parallel (DMP) Processing

When running a simulation, the solution time is typically dominated by three main parts: the time spent to create the element matrices and form the global matrices or global systems of equations, the time to solve the linear system of equations, and the time spent calculating derived quantities (such as stress and strain) and other requested results for each element.

Distributed-memory parallelism allows the entire solution phase to run in parallel, including the stiffness matrix generation, linear equation solving, and results calculations. As a result, a simulation using distributed-memory parallel processing usually achieves much faster solution times than a similar run performed using shared-memory parallel processing (p. 11), particularly at higher core counts.

You can run a DMP solution over multiple cores on a single machine or on multiple machines (that is, a cluster). It automatically decomposes the model into smaller domains, transfers the domains to each core, solves each domain simultaneously, and creates a complete solution to the model. The memory and disk space required to complete the solution can also be distributed over multiple machines. By utilizing all of the resources of a cluster (computing power, RAM, memory and I/O bandwidth), distributed-memory parallel processing can be used to solve very large problems much more efficiently compared to the same simulation run on a single machine.

DMP Solution Behavior

Distributed-memory parallel processing works by launching multiple processes on either a single machine or on multiple machines (as specified by one of the following command line options: -np, -machines, or -mpifile). The machine that the distributed run is launched from is referred to as the head compute node, and the other machines are referred to as the compute nodes. The first process launched on the head compute node is referred to as the master process, and all other processes are referred to as worker processes.

Each DMP process is essentially running a shared-memory parallel (SMP) process. These processes are launched through the specified MPI software layer. The MPI software allows each DMP process to communicate, or exchange data, with the other processes involved in the distributed simulation.

DMP processing is not currently supported for all of the analysis types, elements, solution options, etc. that are available with SMP processing (see Supported Features (p. 37)). In some cases, the program stops the DMP analysis to avoid performing an unsupported action. If this occurs, you must launch an SMP analysis to perform the simulation. In other cases, the program automatically disables the distributed-memory parallel capability and performs the operation using shared-memory parallelism. This disabling of the distributed-memory parallel processing can happen at various levels in the program.

The master process handles the inputting of commands as well as all of the pre- and postprocessing actions. Only certain commands (for example, the **SOLVE** command and supporting commands such as **/SOLU**, **FINISH**, **/EOF**, **/EXIT**, and so on) are communicated to the worker processes for execution. Therefore, outside of the SOLUTION processor (**/SOLU**), a DMP analysis behaves very similar to an SMP analysis. The master process works on the entire model during these pre- and postprocessing steps and may use shared-memory parallelism to improve performance of these operations. During this time, the worker processes wait to receive new commands from the master process.

Once the **SOLVE** command is issued, it is communicated to the worker processes and all DMP processes become active. At this time, the program makes a decision as to which mode to use when computing the solution. In some cases, the solution will proceed using only a distributed-memory parallel (DMP) mode. In other cases, similar to pre- and postprocessing, the solution will proceed using only a shared-memory parallel (SMP) mode. In a few cases, a mixed mode may be implemented which tries to use as much distributed-memory parallelism as possible for maximum performance. These three modes are described further below.

Pure DMP mode

All features in the simulation support distributed-memory parallelism, and it is used throughout the solution. This mode typically provides optimal performance.

Mixed mode

The simulation involves a particular set of computations that do not support DMP processing. Examples include certain equation solvers and remeshing due to mesh nonlinear adaptivity. In these cases, distributed-memory parallelism is used throughout the solution, except for the unsupported set of computations. When that step is reached, the worker processes simply wait while the master process uses shared-memory parallelism to perform the computations. After the computations are finished, the worker processes continue to compute again until the entire solution is completed.

Pure SMP mode

The simulation involves an analysis type or feature that does not support DMP processing. In this case, distributed-memory parallelism is disabled at the onset of the solution, and shared-memory parallelism is used instead. The worker processes are not involved at all in the solution but simply wait while the master process uses shared-memory parallelism to compute the entire solution.

When using shared-memory parallelism within a DMP run (in mixed mode or SMP mode, including all pre- and postprocessing operations), the master process will not use more cores on the head compute node than the total cores you specify to be used for the DMP solution. This is done to avoid exceeding the requested CPU resources or the requested number of licenses.

The following table shows which steps, including specific equation solvers, can be run in parallel using SMP, DMP, and hybrid parallel processing.

Table 4.1: Parallel Capability in SMP, DMP, and Hybrid Parallel Processing

Solvers/Feature	SMP	DMP and Hybrid
Sparse	Y	Y
PCG	Y	Y
Mixed	Y	Y
JCG	Y	Y [a] [b]
Block Lanczos eigensolver	Y	Y
PCG Lanczos eigensolver	Y	Y
Supernode eigensolver	Y	Y [a]

Solvers/Feature	**SMP**	**DMP and Hybrid**
Subspace eigensolver	Y	Y
Unsymmetric eigensolver	Y	Y
Damped eigensolver	Y	Y
QRDAMP eigensolver	Y	Y
Element formulation, results calculation	Y	Y
Graphics and other pre- and postprocessing	Y	Y [a]

[a] This solver/operation only runs in mixed mode.

[b] For static analyses and transient analyses using the full method (**TRNOPT**,FULL), the JCG equation solver runs in pure DMP mode only when the matrix is symmetric. Otherwise, it runs in SMP mode.

The maximum number of cores allowed in a DMP analysis is currently set at 16384. Therefore, assuming the appropriate HPC licenses are available, you can run a DMP analysis using anywhere from 2 to 16384 cores. Performance results vary widely for every model when using any form of parallel processing. For every model, there is a point where using more cores does not significantly reduce the overall solution time. Therefore, it is expected that most models run in DMP can not efficiently make use of hundreds or thousands of cores.

Files generated by a DMP analysis are named `Jobname`*`n`*`.ext`, where *n* is the process number. (See Differences in General Behavior (p. 39) for more information.) The master process is always numbered 0, and the worker processes are 1, 2, etc. When the solution is complete and you issue the **FINISH** command in the SOLUTION processor, the program combines all `Jobname`*`n`*`.rst` files into a single `Jobname.rst` file, located on the head compute node. Other files, such as `.mode`, `.esav`, `.emat`, etc., may be combined as well upon finishing a distributed solution. (See Differences in Postprocessing (p. 44) for more information.)

The remaining sections explain how to configure your environment for distributed-memory parallel processing, how to run a DMP analysis, and what features and analysis types support distributed-memory parallelism. You should read these sections carefully and fully understand the process before attempting to run a distributed analysis. The proper configuration of your environment and the installation and configuration of the appropriate MPI software are critical to successfully running a distributed analysis.

Caution:

In an elastic licensing environment using DMP mode, terminating a worker process from the operating system (Linux: kill -9 command; Windows: Task Manager) could result in a license charge for up to an hour of usage for any fraction of an hour of actual time used. To avoid the inexact licensing charge, terminate the process from within Mechanical APDL (**/EXIT**) or terminate the master process from the operating system.

4.1. Configuration Requirements for DMP Processing

To run a DMP analysis on a single machine, no additional setup is required.

To run a DMP analysis on a cluster, some configuration is required as described in the following sections:

4.1.1. Prerequisites for Running a DMP Analysis

Whether you are running on a single machine or multiple machines, the following condition is true:

- By default, a DMP analysis uses four cores and does not require any HPC licenses. Additional licenses are needed to run a distributed solution with more than four cores. Several HPC license options are available. For more information, see HPC Licensing (p. 9).

If you are running on a single machine, there are no additional requirements for running a distributed solution.

If you are running across multiple machines (for example, a cluster), your system must meet these additional requirements to run a distributed solution.

- Homogeneous network: All machines in the cluster must be the same type, OS level, chip set, and interconnects.

- You must be able to remotely log in to all machines, and all machines in the cluster must have identical directory structures (including the Ansys 2025 R1 installation, MPI installation, and working directories). Do not change or rename directories after you've launched Mechanical APDL. For more information, see Directory Structure Across Machines (p. 36).

- All machines in the cluster must have Ansys 2025 R1 installed, or must have an NFS mount to the Ansys 2025 R1 installation. If not installed on a shared file system, Ansys 2025 R1 must be installed in the same directory path on all systems.

- All machines must have the same version of MPI software installed and running. The table below shows the MPI software and version level supported for each platform.

4.1.1.1. MPI Software

The MPI software supported for DMP processing depends on the platform (see the table below).

The files needed to use Intel MPI, MS MPI, or Open MPI are included on the installation media and are installed automatically when you install Ansys 2025 R1. Therefore, when running on a single machine (for example, a laptop, a workstation, or a single compute node of a cluster) on Windows or Linux, or when running on a Linux cluster, no additional software is needed. However, when running on multiple Windows machines you must use a cluster setup, and you must install the MPI software separately (see Installing the Software (p. 27) later in this section).

Table 4.2: Platforms and MPI Software

Platform	**MPI Software**[a]
Linux (Intel CPU)	Intel MPI 2021.13.0 (default)[b] Open MPI 4.0.5 (optional) [c] Intel MPI 2018.3.222 (optional)

Platform	MPI Software[a]
Linux (AMD CPU)	Open MPI 4.0.5 (default)[c] Intel MPI 2021.13.0 (optional)[b] Intel MPI 2018.3.222 (optional)
Windows 10 or Windows 11 Windows Server 2022 or Windows Server 2019 (single machine)	Intel MPI 2021.13.0 (default) MS MPI v10.1.12 (optional)
Windows Server 2019 (cluster)	Windows Server 2019 HPC Pack [d]

[a] Ansys chooses the default MPI based on robustness and performance. The optional MPI versions listed here are available if necessary for troubleshooting (see Using an alternative MPI version (p. 51)).

[b] If Mechanical APDL detects a Mellanox OFED driver version older than 5.0 or an AMD cluster with InfiniBand using any OFED, then Intel MPI 2018.3.222 is automatically set as the default. If it detects an AMD cluster with Amazon EFA, then the default is set to Open MPI 4.0.5 instead.

[c] Mellanox OFED driver version 4.4 or higher is required.

[d] If you are running across multiple Windows machines, you must use Microsoft HPC Pack (MS MPI) and the HPC Job Manager to start a DMP run (see Starting a DMP Analysis (p. 31)).

By default, Open MPI is launched with a set of default runtime parameters that are chosen to improve performance, particularly on machines with AMD-based processors.

4.1.1.2. Installing the Software

Install Ansys 2025 R1 following the instructions in the *Ansys, Inc. Installation Guide* for your platform. Be sure to complete the installation, including all required post-installation procedures.

To run on a cluster, you must:

- Install Ansys 2025 R1 on all machines in the cluster, in the exact same location on each machine.

- For Windows, you can use shared drives and symbolic links. Install Ansys 2025 R1 on one Windows machine (for example, `C:\Program Files\ANSYS Inc\V251`) and then *share* that installation folder. On the other machines in the cluster, create a symbolic link (at `C:\Program Files\ANSYS Inc\V251`) that points to the UNC path for the shared folder. On Windows systems, you must use the Universal Naming Convention (UNC) for all file and path names for the DMP analysis to work correctly.

- For Linux, you can use the exported NFS file systems. Install Ansys 2025 R1 on one Linux machine (for example, at `/ansys_inc/v251`), and then export this directory. On the other machines in the cluster, create an NFS mount from the first machine to the same local directory (`/ansys_inc/v251`).

Installation files for MS MPI on Windows

Microsoft MPI is installed and ready for use as part of the Ansys 2025 R1 installation, and no action is needed if you are running on a single machine. If you require MS MPI on another machine, the installer can be found at `C:\Program Files\ANSYS Inc\V251\tp\MPI\Microsoft\10.1.12498.18\Windows\MSMpiSetup.exe`

Microsoft HPC Pack 2019 (Windows Server 2019)

You must complete certain post-installation steps before running a DMP analysis on a Microsoft HPC Server 2019 system. The post-installation instructions provided below assume that Microsoft Windows Server 2019 and Microsoft HPC Pack (which includes MS MPI) are already installed on your system. The post-installation instructions can be found in the following README files:

```
C:\Program Files\ANSYS Inc\V251\tp\MPI\WindowsHPC\README.mht
```

or

```
C:\Program Files\ANSYS Inc\V251\tp\MPI\WindowsHPC\README.docx
```

Microsoft HPC Pack examples are also located in `C:\Program Files\ANSYS Inc\V251\tp\MPI\WindowsHPC`. Jobs are submitted to the Microsoft HPC Job Manager either from the command line or the Job Manager GUI.

To submit a job via the GUI, go to **Start> All Programs> Microsoft HPC Pack> HPC Job Manager**. Then click on **Create New Job from Description File**.

4.1.2. Setting Up the Cluster Environment for DMP

After you've ensured that your cluster meets the prerequisites and you have Ansys 2025 R1 and the correct version of MPI installed, you need to configure your distributed environment using the following procedure.

1. Obtain the machine name for each machine on the cluster.

   Windows 10 (or Windows 11) and Windows Server 2019:

   From the **Start** menu, pick **Settings >System >About**. The full computer name is listed under **PC Name**. Note the name of each machine (not including the domain).

   Linux:

   Type **`hostname`** on each machine in the cluster. Note the name of each machine.

2. **Linux only:** First determine if the cluster uses the secure shell (ssh) or remote shell (rsh) protocol.

   - **For ssh:** Use the ssh-keygen command to generate a pair of authentication keys. Do not enter a passphrase. Then append the new public key to the list of authorized keys on each compute node in the cluster that you wish to use.
   - **For rsh:** Create a `.rhosts` file in the home directory. Add the name of each compute node you wish to use on a separate line in the `.rhosts` file. Change the permissions of

the `.rhost` file by issuing: **`chmod 600 .rhosts`**. Copy this `.rhosts` file to the home directory on each compute node in the cluster you wish to use.

Verify communication between compute nodes on the cluster via ssh or rsh. You should not be prompted for a password. If you are, correct this before continuing. Note, all compute nodes must be able to communicate with all other compute nodes on the cluster (or at least all other nodes within the same partition/queue as defined by a job scheduler) via rsh or ssh without being prompted for a password. For more information on using ssh/rsh without passwords, search online for "Passwordless SSH" or "Passwordless RSH", or see the man pages for ssh or rsh.

3. **Windows only:** Verify that all required environment variables are properly set. If you followed the post-installation instructions described above for Microsoft HPC Pack (Windows HPC Server), these variables should be set automatically.

   On the head compute node, where Ansys 2025 R1 is installed, check these variables:

   ANSYS251_DIR=C:\Program Files\ANSYS Inc\v251\ansys

   ANSYSLIC_DIR=C:\Program Files\ANSYS Inc\Shared Files\Licensing

   where `C:\Program Files\ANSYS Inc` is the location of the product install and `C:\Program Files\ANSYS Inc\Shared Files\Licensing` is the location of the licensing install. If your installation locations are different than these, specify those paths instead.

   On Windows systems, you must use the Universal Naming Convention (UNC) for all Ansys, Inc. environment variables on the compute nodes for DMP to work correctly.

   On the compute nodes, check these variables:

   ANSYS251_DIR=\\head_node_machine_name\ANSYS Inc\v251\ansys

   ANSYSLIC_DIR=\\head_node_machine_name\ANSYS Inc\Shared Files\Licensing

4. **Windows only:** Share out the `ANSYS Inc` directory on the head node with full permissions so that the compute nodes can access it.

4.1.2.1. Optional Setup Tasks

The tasks explained in this section are optional. They are not required to run a DMP analysis correctly, but they may be useful for achieving the most usability and efficiency, depending on your system configuration.

On Linux systems, you can also set the following environment variables:

- **ANSYS_NETWORK_START** - This is the time, in seconds, to wait before timing out on the start-up of the client (default is 15 seconds).

- **ANSYS_NETWORK_COMM** - This is the time to wait, in seconds, before timing out while communicating with the client machine (default is 5 seconds).

- **ANS_SEE_RUN_COMMAND** - Set this environment variable to 1 to display the actual command issued by the program.

On Windows systems, you can set the following environment variables to display the actual command issued by the program:

- **ANS_SEE_RUN** = TRUE
- **ANS_CMD_NODIAG** = TRUE

4.1.2.2. Using the `mpitest` Program

The `mpitest` program performs a simple communication test to verify that the MPI software is set up correctly. The `mpitest` program should start without errors. If it does not, check your paths and permissions; correct any errors and rerun.

When running the mpitest program, you must use an even number of processes. We recommend you start with the simplest test between two processes running on a single node. This can be done via the procedures outlined here for each platform and MPI type.

The command line arguments `-np`, `-machines`, and `-mpifile` work with the `mpitest` program in the same manner as they do in a DMP analysis (see Starting a DMP Analysis via Command Line (p. 32)).

On Linux:

For Intel MPI (default), issue the following command:

```
mpitest251 -np 2
```

which is equivalent to:

```
mpitest251 -machines machine1:2
```

For Open MPI, issue the following command:

```
mpitest251 -mpi openmpi -np 2
```

which is equivalent to:

```
mpitest251 -mpi openmpi -machines machine1:2
```

On Windows:

For Intel MPI (default), issue the following command.

```
ansys251 -np 2 -mpitest
```

4.1.2.3. Interconnect Configuration

Using a slow interconnect reduces the performance you experience in a distributed parallel simulation. For optimal performance, we recommend an interconnect with a high communication bandwidth (2000 megabytes/second or higher) and a low communication latency (5 microseconds or lower). This is due to the significant amount of data that must be transferred between processes during a distributed parallel simulation.

Distributed-memory parallelism is supported on the following interconnects. Not all interconnects are available on all platforms; see the Platform Support section of the Ansys Website for a current list of supported interconnects. Other interconnects may work but have not been tested.

- InfiniBand (recommended)
- Omni-Path (recommended)
- GigE

On Windows x64 systems, use the Network Wizard in the Compute Cluster Administrator to configure your interconnects. See the Compute Cluster Pack documentation for specific details on setting up the interconnects. You may need to ensure that Windows Firewall is disabled for distributed-memory parallelism to work correctly.

4.2. Starting a DMP Analysis

After you've completed the configuration steps, you can use several methods to start a DMP anlaysis: We recommend that you use the Mechanical APDL Product Launcher to ensure the correct settings. All methods are explained here.

- Use the launcher (p. 31)
- Use the command line (p. 32)
- Use the HPC Job Manager on Windows x64 systems to run across multiple machines (p. 34)
- Use Remote Solve in Ansys Workbench. (p. 35)

Notes on Running a DMP Analysis:

You can use an NFS mount to the Ansys 2025 R1 installation on Linux, or shared folders on Windows. However, we do not recommend using either NFS-mounts or shared folders for the working directories. Doing so can result in significant declines in performance.

Only the master process reads the `config.ans` file. A DMP analysis ignores the **/CONFIG**,NOELDB and **/CONFIG**,FSPLIT commands.

The program limits the number of processes used to be less than or equal to the number of physical cores on the machine. This is done to avoid running the program on virtual cores (for example, by means of hyperthreading), which typically results in poor per-core performance. For optimal performance, consider closing down all other applications before launching Mechanical APDL.

4.2.1. Starting a DMP Analysis via the Launcher

Use the following procedure to start a DMP analysis via the launcher.

1. Open the Mechanical APDL Product Launcher:

   Windows 10 (or Windows 11) and Windows Server 2019:

   Start >Ansys 2025 R1 >Mechanical APDL Product Launcher 2025 R1

Linux:

```
launcher251
```

2. Select the correct environment and license.

3. Go to the **High Performance Computing Setup** tab. Select **Use Distributed Computing (DMP)**.

   Note:

   On Linux systems, you cannot use the Launcher to start the program in interactive (GUI) mode with distributed computing; this combination is blocked.

   Specify the MPI type to be used for this distributed run. MPI types include:

   - Intel MPI
   - Open MPI (Linux only)
   - MS MPI (Windows only)

   See the MPI table in the beginning of this chapter for the specific MPI version for each platform. On Windows, you cannot specify multiple hosts or an MPI file.

   Choose whether you want to run on a local machine, specify multiple hosts, or specify an existing MPI file (such as an Intel MPI configuration file or an Open MPI host file (on Linux)):

   - If local machine, specify the number of cores you want to use on that machine (indicated by the yellow arrow with -np in Figure 5.1: Product Launcher - High Performance Computing Setup (p. 56)).
   - If multiple hosts, use the **New Host** button to add machines to the **Selected Hosts** list.
   - If specifying an MPI file (Linux only), type in the full path to the file, or browse to the file. If typing in the path, you must use the absolute path.

   Additional Options for Linux Systems

   On Linux systems, you can choose to use the remote shell (rsh) protocol instead of the secure shell (ssh) protocol; ssh is the default.

   Also in the launcher, you can select the **Use launcher-specified working directory on all nodes** option on the **High Performance Computing Setup** tab. This option uses the working directory as specified on the **File Management** tab as the directory structure on the head compute node and all other compute nodes. If you select this option, all machines will require the identical directory structure matching the working directory specified on the launcher.

4. Click **Run** to launch Mechanical APDL.

4.2.2. Starting a DMP Analysis via Command Line

You can also start a DMP analysis via the command line using the following procedures.

Running on a Local Host

If you are running a DMP analysis locally (that is, running across multiple cores on a single machine), you need to specify the number of cores you want to use via the `-np` command line option:

```
ansys251 -dis -np n
```

(For a general discussion on processes, threads, and cores in parallel processing, see Specifying Processes, Threads, and Cores in SMP, DMP, and Hybrid Parallel (p. 2) in the overview section.)

If you are using Intel MPI, you do not need to specify the MPI software via the command line option. To specify an alternative MPI software, use the `-mpi` command line option. For example, if you run a job in batch mode on a local host using eight cores with an input file named `input1` and an output file named `output1`, the launch commands for Linux and Windows would be as shown below.

On Linux:

```
ansys251 -dis -np 8 -b < input1 > output1  (for default Intel MPI 2021.11.0)
```

or

```
ansys251 -dis -mpi intelmpi2018 -np 8 -b < input1 > output1  (for Intel MPI 2018.3.222)
```

or

```
ansys251 -dis -mpi openmpi -np 8 -b < input1 > output1  (for Open MPI)
```

On Windows:

```
ansys251 -dis -np 8 -b -i input1 -o output1  (for default Intel MPI 2021.11.0)
```

or

```
ansys251 -dis -mpi msmpi -np 8 -b -i input1 -o output1  (for MS MPI)
```

Running on Multiple Hosts

If you are running a DMP analysis across multiple hosts, you need to specify the number of cores you want to use on each machine:

```
ansys251 -dis -machines machine1:np:machine2:np:machine3:np
```

On Linux, you may also need to specify the shell protocol used by the MPI software. DMP analyses use the secure shell protocol by default, but in some cluster environments it may be necessary to force the use of the remote shell protocol. This can be done via the `-usersh` command line option.

Consider the following examples which assume a batch mode run using two machines (using four cores on one machine and using two cores on the other machine), with an input file named `input1` and an output file named `output1`. The launch commands for Linux and Windows would be as shown below.

On Linux:

```
ansys251 -dis -b -machines machine1:4:machine2:2 < input1 > output1  (for default Intel MPI 2021.11.0)
```

or

```
ansys251 -dis -mpi intelmpi2018 -b -machines machine1:4:machine2:2 < input1 > output1  (for Intel MPI 2018.3.222
```

or

```
ansys251 -dis -mpi openmpi -b -machines machine1:4:machine2:2 < input1 > output1 (for Open MPI)
```

or

```
ansys251 -dis -b -machines machine1:4:machine2:2 -usersh < input1 > output1  (for remote shell protocol between
```

Note:

The `-np` and `-machines` command line options are mutually exclusive. If both are specified, `-machines` takes precedence and `-np` is ignored.

On Windows, users must launch jobs via the **Windows HPC Job Manager (p. 34)**.

Specifying a Preferred Parallel Feature License

If you have more than one HPC license feature, you can use the `-ppf` command line option to specify which HPC license to use for the parallel run. See HPC Licensing (p. 9) for more information.

4.2.3. Starting a DMP Analysis via the HPC Job Manager

If you are running a DMP analysis across multiple Windows x64 systems, you must start it using the Microsoft HPC Pack (MS MPI) and the HPC Job Manager. For more information, refer to the following README files:

`Program Files\ANSYS Inc\V251\tp\MPI\WindowsHPC\README.mht`

or

`Program Files\ANSYS Inc\V251\tp\MPI\WindowsHPC\README.docx`

4.2.4. Starting a DMP Analysis via a Linux Job Scheduler

There are additional considerations if you are running a DMP analysis under a job scheduler. Mechanical APDL should be launched either with an explicit resource list (obtained from the job scheduler and provided using -machines or an MPI file) or via "tight integration." The MPI software may be able to detect resources assigned to a scheduled job and automatically launch Mechanical APDL accordingly. This is referred to as tight integration.

To utilize tight integration, set the **ANS_MULTIPLE_NODES** environment variable in your job execution script before calling `ansys252`. For Intel MPI, also set the **I_MPI_HYDRA_BOOSTRAP** variable to the appropriate value for your scheduler. Do not specify `-machines`, and instead specify a total `-np` value. Here is a script snippet to obtain two 32-core nodes with the Slurm job scheduler, for example:

```
ANS_MULTIPLE_NODES=1
I_MPI_HYDRA_BOOTSTRAP=slurm
ansys252 -np 64 -b -i input.dat -o output.out
```

The MPI should obtain the host list from Slurm, then launch Mechanical APDL place the processes accordingly.

4.2.5. Starting a DMP Solution in the Mechanical application (via Ansys Workbench)

If you are running Ansys Workbench, you can start a a DMP solution in the Mechanical application. Go to **Tools > Solve Process Settings**; select the remote solve process you want to use and click the **Advanced** button. To enable distributed-memory parallel processing, ensure that **Distribute Solution (if possible)** is selected (this is the default). If necessary, enter any additional command line arguments to be submitted; the options are described in Starting a DMP Analysis via Command Line (p. 32). If **Distribute Solution (if possible)** is selected, you do not need to specify the `-dis` flag on the command line.

If you are running a remote solution on multiple machines, use the `-machines` option to specify the machines and cores on which to run the job. If you are running a remote solution on one machine with multiple cores, specify the number of cores in the **Max Number of Utilized Cores** field; no command line arguments are needed.

For more information on running a distributed solution in Ansys Workbench, see Using Solve Process Settings.

4.2.6. Using MPI Files

You can specify an existing MPI file (such as an Intel MPI configuration file or an Open MPI hostfile) on the command line rather than typing out multiple hosts or a complicated command line:

```
ansys251 -dis -mpifile file_name
```

The format of the mpifile is specific to the MPI library being used. Refer to the documentation provided by the MPI vendor for the proper syntax.

If the file is not in the current working directory, you will need to include the full path to the file. The file must reside on the local machine.

You cannot use the `-mpifile` option in conjunction with the `-np` (local host) or `-machines` (multiple hosts) options.

Example MPI files for Intel MPI and Open MPI are shown in the following sections. For use on your system, modify the hostnames (`mach1`, `mach2`), input filename (`inputfile`), and output filename (`outputfile`) accordingly. Additional command-line arguments, if needed, can be added at the end of each line.

4.2.6.1. Intel MPI Configuration File Examples

Intel MPI uses a configuration file to define the machine(s) that will be used for the simulation. Typical Intel MPI configuration files are shown below for two nodes with two cores per node.

On Linux:

```
-host mach1 -np 2 /ansys_inc/v251/ansys/bin/ansysdis251 -dis -b -i inputfile -o outputfile
-host mach2 -np 2 /ansys_inc/v251/ansys/bin/ansysdis251 -dis -b -i inputfile -o outputfile
```

4.2.6.2. Open MPI Hostfile Example

Open MPI uses hostfiles to define the machine(s) that will be used for the simulation. Hostfiles are simple text files with one line for each machine (including the head compute node as well as all other compute nodes) that specifies the machine name and the number of cores (slots) to be used on each machine . The example below shows a typical hostfile for two machines, one running with 3 cores and the other with 2.

On Linux:

```
mach1 slots=3
mach2 slots=2
```

4.2.7. Directory Structure Across Machines

During a DMP run, the program writes files to the head compute node and all other compute nodes as the analysis progresses.

The working directory for each machine can be on a local drive or on a network shared drive. For optimal performance, use local disk storage rather than network storage. Set up the same working directory path structure on the head compute node and all other compute nodes.

When setting up your cluster environment to run a DMP analysis, consider that the program:

- Cannot launch if identical working directory structures have not been set up on the head compute node and all other compute nodes.
- Always uses the current working directory on the head compute node and expects identical directory structures to exist on all other compute nodes. (If you are using the launcher, the working directory specified on the **File Management** tab is the one that the program expects.)

4.3. Supported Analysis Types and Features

Distributed-memory parallel processing does not support all analysis types, elements, solution options, etc. that SMP processing supports. As a result, the program may:

- Stop an analysis to avoid performing an unsupported action, requiring you to launch an SMP analysis to perform the simulation.
- Disable the distributed-memory parallel processing capability and perform the operation using shared-memory parallelism.

  The disabling of the distributed-memory parallel processing can occur at various levels in the program. For more information, see DMP Solution Behavior (p. 23).

4.3.1. Supported Analysis Types

This section lists analysis capabilities that support DMP processing. This is not a comprehensive list, but represents major features and capabilities found in the Ansys program.

Most element types are valid in an analysis that uses distributed-memory parallel processing (including but not limited to the elements mentioned below). For those element types that do not support DMP processing, a restriction is included in the element description (see the *Element Reference*). Check the assumptions/restrictions list for the element types you are using.

Supported Analysis Types

The following analysis types are supported and use distributed-memory parallelism throughout the solution. (That is, the solution runs in pure DMP mode (p. 24).)

- Linear static and nonlinear static analyses for single-field structural problems (DOFs: UX, UY, UZ, ROTX, ROTY, ROTZ, WARP) and single-field thermal problems (DOF: TEMP).
- Buckling analyses using the Subspace or Block Lanczos eigensolver (**BUCOPT**,SUBSP; LANB).
- Modal analyses using the Subspace, PCG Lanczos, Block Lanczos, Unsymmetric, Damped, or QR damped eigensolver (**MODOPT**,SUBSP; LANPCG; LANB; UNSYM; DAMP; or QRDAMP).
- Harmonic analyses.
- Transient dynamic analyses.
- Substructuring analyses, including component mode synthesis (CMS) analyses.
- Spectrum analyses.
- Radiation analyses using the radiosity method. Note that view factors are calculated in parallel only for 3D analyses when they are initiated by a **SOLVE** command in the /SOLU processor. For more details on view factors, see Other View Factor Options and Considerations in the Thermal Analysis Guide.
- Low-frequency electromagnetic analyses.
- Coupled-field analyses.
- Superelements in the use pass of a substructuring analysis.
- Cyclic symmetry analyses (except mode-superposition harmonic).

The following analysis types are supported and use distributed-memory parallelism throughout the solution, except for the equation solver which uses shared-memory parallelism. (In these cases, the solution runs in mixed mode (p. 24).)

- Static and full transient analyses (linear or nonlinear) that use the JCG equation solver. Note that when the JCG equation solver is used in these analysis types, it will run using distributed-memory parallelism (that is, pure DMP mode) if the matrix is symmetric.
- Modal analyses using the Supernode eigensolver (**MODOPT**,SNODE).

4.3.2. Supported Features

This section list features that support DMP processing and features that are blocked. These are not comprehensive lists, but represent major features and capabilities found in the program.

Supported Features:

The following features are supported and use distributed-memory parallelism throughout the solution. (That is, the solution runs in pure DMP mode (p. 24).)

- Large deformations (**NLGEOM**,ON).
- Line search (**LNSRCH**,ON).
- Auto time stepping (**AUTOTS**,ON).
- Initial conditions (**IC**).
- Initial state (**INISTATE**).
- Nonlinear material properties specified by the **TB** command.
- Gasket elements and pre-tension elements.
- Lagrange multiplier based mixed u-P elements and TARGE169 - CONTA178.
- Contact nonlinearity (TARGE169 - CONTA178)
- User programmable features, including the user-defined element (USER300).
- Multi-frame restarts.
- Arc-length method (**ARCLEN**).
- Prestress effects (**PSTRES**).
- Inertia relief (**IRLF**,1), including the mass summary option (**IRLF**,-1).
- Multiple load steps and enforced motion in modal analyses (**MODCONT**).
- Residual vectors calculations (**RESVEC**).

The following feature is supported and uses distributed-memory parallelism throughout the solution, except for the remeshing procedure. (In this case, the solution runs in mixed mode (p. 24).)

- Mesh nonlinear adaptivity (**NLADAPTIVE**).

The following analysis type is supported and uses distributed-memory parallelism throughout the solution, except for the 3D model-creation procedure:

- 2D to 3D analysis (**MAP2DTO3D**)

The following features are supported, but do not use distributed-memory parallelism within a DMP run. (The solution runs in pure SMP mode (p. 24).)

- Mode-superposition harmonic cyclic symmetry analysis.
- VCCT Crack Growth Simulation when fracture parameters in addition to **CINT**,VCCT are calculated on the same geometric crack front. See VCCT Crack-Growth Simulation Assumptions in the *Fracture Analysis Guide* for more information.

Blocked Features:

The following feature does not support distributed-memory parallelism:

- Element morphing.

4.4. Understanding the Working Principles and Behavior of Distributed-memory Parallelism

The fundamental difference between distributed-memory parallel (DMP) processing and shared memory parallel (SMP) processing is that *N* number of processes will be running at the same time (where *N* is the total number of CPU cores used) for one model. These *N* processes may be running on a single machine (in the same working directory) or on multiple machines. These processes are not aware of each other's existence unless they are communicating (sending messages). The program along with the MPI software provide the means by which the processes communicate with each other in the right location and at the appropriate time.

The following topics give a summary of behavioral differences between DMP and SMP processing in specific areas of the program:

4.4.1. Differences in General Behavior

File Handling Conventions	Upon startup and during a DMP solution, the program appends *n* to the current jobname (where *n* stands for the process rank). The master process rank is 0 and the worker processes are numbered from 1 to *N* - 1. In the rare case that the user-supplied jobname ends in a numeric value (0...9) or an underscore, an underscore is automatically appended to the jobname prior to appending the process rank. This is done to avoid any potential conflicts with the new file names. Therefore, upon startup and during a parallel solution, each process will create and use files named `Jobname`*`n`*`.`*`ext`*. These files contain the local data that is specific to each DMP process. Some common examples include the `.log` and `.err` files as well as most files created during solution such as `.esav`, `.full`, `.rst`, and `.mode`. See Program-Generated Files in the *Basic Analysis Guide* for more information on files that Mechanical APDL typically creates. Actions that are performed only by the master process (**/PREP7**, **/POST1**, etc.) will work on the global `Jobname.`*`ext`* files by default. These files (such as `Jobname.db` and `Jobname.rst`) contain the global data for the entire model. For example, only the master process will save and resume the `Jobname.db` file.

After a parallel solution successfully completes, the program automatically merges some of the (local) `Jobname`*n*`.`*ext* files into a single (global) file named `Jobname.`*ext*. These include files such as `Jobname.rst`, `Jobname.esav`, `Jobname.emat`, and so on (see the **DMPOPTION** command for a complete list of these files). This action is performed when the **FINISH** command is executed upon leaving the solution processor. These files contain the same information about the final computed solution as files generated for the same model computed with SMP processing. Therefore, all downstream operations (such as postprocessing) can be performed using SMP processing (or in the same manner as SMP processing) by using these global files.

If any of these global `Jobname.`*EXT* files are not needed for downstream operations, you can reduce the overall solution time by suppressing the file combination for individual file types (see the **DMPOPTION** command for more information). If it is later determined that a global `Jobname.`*EXT* file is needed for a subsequent operation or analysis, the local files can be combined by using the **COMBINE** command.

DMPOPTION also has an option to combine the results file at certain time points during the distributed solution. This enables you to postprocess the model while the solution is in progress, but leads to slower performance due to increased data communication and I/O.

The program will not delete most files written by the worker processes when the analysis is completed. If you choose, you can delete these files when your analysis is complete (including any restarts that you may wish to perform). If you do not wish to have the files necessary for a restart saved, you can issue **RESCONTROL**,NORESTART.

File copy, delete, and rename operations can be performed across all processes by using the *`DistKey`* option on the **/COPY**, **/DELETE**, and **/RENAME** commands. This provides a convenient way to manage local files created by a distributed parallel solution. For example, **/DELETE**,`Fname`,`ext`,,ON automatically appends the process rank number to the specified file name and deletes `Fname`*n*`.ext` from all processes. If the local files (`Jobname.*`) are not needed in downstream analyses, you can issue **/FCLEAN** to delete them all to save space. See the **/COPY**, **/DELETE**, **/FCLEAN**, and **/RENAME** command descriptions for more information. In addition, the **/ASSIGN** command can be used to control the name and location of specific local and global files created during a distributed-memory parallel analysis.

Batch and Interactive Mode

You can launch a DMP analysis in either interactive or batch mode for the master process. However, the worker processes are always in batch mode. The worker processes cannot read the `start.ans` or `stop.ans` files. The master process sends all **/CONFIG**,*LABEL* commands to the worker processes as needed.

On Windows systems, there is no Mechanical APDL output console window when running a DMP analysis in the GUI (interactive mode). All standard output from the master process will be written to a file named `file0.out`. (Note that the `Jobname` is not used.)

Output Files

When a DMP analysis is executed, the output for the master process is written in the same fashion as an SMP run. By default, the output is written to the screen; or, if you specified an output file via the launcher or the `-o` command line option, the output for the master process is written to that file.

The worker processes automatically write this output to Jobname*n*.out. Typically, these worker process output files have little value because all of the relevant job information is written to the screen or the master process output file. The exception is when the domain decomposition method is automatically chosen to be FREQ (frequency-based for a harmonic analysis) or CYCHI (harmonic index-based for a cyclic symmetry analysis); see the **DDOPTION** command for more details. In these cases, the solution information for the harmonic frequencies or cyclic harmonic indices solved by the worker processes are only written to the output files for those processes (Jobname*n*.out).

Error Handling

The same principle also applies to the error file Jobname*n*.err. When a warning or error occurs on one of the worker processes during a DMP solution, the process writes that warning or error message to its error file and then communicates the warning or error message to the master process. Typically, this allows the master process to write the warning or error message to its error file and output file and, in the case of an error message, allows all of the distributed processes to exit the program simultaneously.

In some cases, an error message may fail to be fully communicated to the master process. If this happens, you can view each Jobname*n*.err and/or Jobname*n*.out file in an attempt to learn why the job failed. The error files and output files written by all the processes will be incomplete but may still provide some useful information as to why the job failed.

In some rare cases, the job may hang. When this happens, you can use the cleanup script (or `.bat` file on Windows) to kill the processes. The cleanup script is automatically written into the initial working directory of the master process and is named cleanup-ansys-[*machineName*]-[*processID*].sh (or `.bat`).

Use of Mechanical APDL Commands

In pre- and postprocessing, APDL works the same in a DMP analysis as in an SMP analysis. However, in the solution processor (**/SOLU**), a DMP analysis does not support certain ***GET** items. In

	general, DMP processing supports global solution ***GET** results such as total displacements and reaction forces. It does not support element level results specified by **ESEL**, **ESOL**, and **ETABLE** labels. Unsupported items will return a ***GET** value of zero.
Condensed Data Input	Multiple commands entered in an input file can be condensed into a single line if the commands are separated by the $ character (see Condensed Data Input in the *Command Reference*). DMP processing cannot properly handle condensed data input. Each command must be placed on its own line in the input file for a DMP run.

4.4.2. Differences in Solution Processing

Domain Decomposition (**DDOPTION** Command)	Upon starting a solution in a DMP analysis, the program automatically decomposes the problem into *N* CPU domains so that each process (or each CPU core) works on only a portion of the simulation. This domain decomposition is done only by the master process. Typically, the optimal domain decomposition method is automatically chosen based on a variety of factors (analysis type, number of CPU cores, available RAM, and so on). However, you can control which domain decomposition method is used by setting the `Decomp` argument on the **DDOPTION** command. For most simulations, the program automatically chooses the mesh-based domain decomposition method (`Decomp` = MESH), which means each CPU domain is a group or subset of elements within the whole model. For certain harmonic analyses, the domain decomposition may be based on the frequency domain (`Decomp` = FREQ), in which case each CPU domain computes the harmonic solution for the entire model at a different frequency point. For certain cyclic symmetry analyses, the domain decomposition may be based on the harmonic indices (`Decomp` = CYCHI), in which case each CPU domain computes the cyclic solution for a different harmonic index. The `NPROCPERSOL` argument on the **DDOPTION** command gives you the flexibility to combine the FREQ or CYCHI decomposition methods with the mesh-based domain decomposition method. Consider a harmonic analysis with 50 frequency points requested (**NSUBST**,50) run on a workstation executing a DMP solution with 16 cores (`-dis-np` 16). Using mesh decomposition (**DDOPTION**,MESH) essentially solves one frequency at a time with 16 groups of elements (1x16). Using **DDOPTION**,FREQ,1 solves 16 frequencies at a time with 1 group of elements; that is, the entire FEA model (16x1). Using the `NPROCPERSOL` field allows you to consider alternative combinations in between these 2 scenarios. You could try **DDOPTION**,FREQ,2 to solve 8 frequencies at a time with 2 groups of elements per solution (8x2), or **DDOPTION**,FREQ,4 to solve 4 frequencies with 4 groups of elements (4x4), and so on. Note that the total core count specified at startup cannot be

altered, and the program works to maintain the *NPROCPERSOL* value as input, which means the number of frequency or cyclic harmonic index solutions solved at a time may need to be adjusted to fit within the other defined parameters.

In the case of a linear perturbation analysis, you should also set the *NUMSOLFORLP* argument to ensure the best solution performance. For decomposition based on the frequency domain (*Decomp* = FREQ or automatically chosen when *Decomp* = AUTO), set *NUMSOLFORLP* to the number of frequency solutions in the subsequent harmonic analysis. For decomposition based on harmonic index (*Decomp* = CYCHI or automatically chosen when *Decomp* = AUTO), set *NUMSOLFORLP* to the number of harmonic index solutions in the subsequent cyclic modal analysis.

There are pros and cons for each domain decomposition approach. For example, with the default MESH method, the total amount of memory required to solve the simulation in *N* processes is typically not much larger than the amount of memory required to solve the simulation on one process. However, when using the FREQ or CYCHI domain decomposition method, the amount of memory required to solve the simulation on *N* processes is typically *N* times greater than the amount of memory to solve the simulation on one process. As another example, with the MESH domain decomposition method, the application requires a significant amount of data communication via MPI, thus requiring a fast interconnect for optimal performance. With the FREQ and CYCHI methods, very little data communication is necessary between the processes and, therefore, good performance can still be achieved using slower interconnect hardware.

All FEA data (elements, nodes, materials, sections, real constants, boundary conditions, etc.) required to compute the solution for each CPU domain is communicated to the worker processes by the master process. Throughout the solution, each process works only on its piece of the entire model. When the solution phase ends (for example, **FINISH** is issued in the solution processor), the master process in a DMP analysis works on the entire model again (that is, it behaves like SMP processing).

Print Output (**OUTPR** Command)

In DMP processing, the **OUTPR** command prints NSOL and RSOL in the same manner as in SMP processing. However, for other items such as ESOL, a DMP analysis prints only the element solution for the group of elements belonging to the CPU domain of the master process. Therefore, **OUTPR**, ESOL has incomplete information and is not recommended. Also, the order of elements is different from that of SMP processing due to domain decomposition. A direct one-to-one element comparison with SMP processing will be different if using **OUTPR**.

When using the frequency domain decomposition in a harmonic analysis, the NSOL, RSOL, and ESOL data is printed into the output

	file written by the master process that computes the frequency solutions. The **OUTPR** print for the other frequency solutions is written to the output files created by the worker processes (`Jobname`*n*`.out`).
Large Number of CE/CP and Contact Elements	Both SMP processing and DMP processing can handle a large number of coupling and constraint equations (**CE**/**CP**) and contact elements. However, specifying too many of these items can force a DMP solution to communicate more data among each process, resulting in longer elapsed time to complete a distributed parallel job. You should reduce the number of **CE**/**CP** if possible and make potential contact pairs in a smaller region to achieve non-deteriorated performance. In addition, for assembly contact pairs or small sliding contact pairs, you can use the command **CNCHECK**,TRIM to remove contact and target elements that are initially in far-field (open and not near contact). This trimming option will help to achieve better performance in DMP runs.

4.4.3. Differences in Postprocessing

Postprocessing with Database File and **SET** Commands	SMP processing can postprocess the last set of results using the `Jobname.db` file (if the solution results were saved), as well as using the `Jobname.rst` file. A DMP analysis, however, can only postprocess using the `Jobname.rst` file and cannot use the `Jobname.db` file as solution results are not entirely written to the database. You will need to issue a **SET** command before postprocessing.
Postprocessing with Multiple Results Files	By default, a DMP analysis will automatically combine the local results files (for example, `Jobname`*n*`.rst`) into a single global results file (`Jobname.rst`). This step can be expensive depending on the number of load steps and the amount of results stored for each solution. It requires each local results file to be read by each worker process, communicated to the master process, and then combined together and written by the master process. As a means to reduce the amount of communication and I/O performed by this operation, the **DMPOPTION** command can be used to skip the step of combining the local results file into a single global results file. Then the **RESCOMBINE** command macro can be used in **/POST1** to individually read each local results file until the entire set of results is placed into the database for postprocessing. If needed, a subsequent **RESWRITE** command can then be issued to write a global results file for the distributed solution. Note that if the step of combining the results file is skipped, it may affect downstream analyses that rely on a single global results file for the entire model. If it is later determined that a global results file (for example, `Jobname.rst`) is needed for a subsequent operation, you can use the **COMBINE** command to combine the local results files into a single, global results file.

4.4.4. Restarts with DMP processing

DMP processing supports multiframe restarts for nonlinear static, full transient, and mode-superposition transient analyses. The procedures and command controls are the same as described in the *Basic Analysis Guide*. However, restarts with DMP processing have additional limitations based on the procedure used, as described in the following sections:

- Procedure 1: Use the Same Number of Cores (p. 45)
- Procedure 2: Use a Different Number of Cores (p. 46)

See also Additional Consideration for the Restart (p. 47) for more information on restarting a distributed-memory parallel solution.

4.4.4.1. Procedure 1 - Use the Same Number of Cores

This procedure requires that you use the same number of cores for the restart as in the original run. It does not require any additional commands beyond those used in a typical multiframe restart procedure.

- The total number of cores used when restarting a DMP analysis must not be altered following the first load step and first substep.

  Example 1: If you use the following command line for the first load step:

  ```
  ansys251 -dis -np 8 -i input -o output1
  ```

  then, you must also use 8 cores (`-dis -np 8`) for the multiframe restart, and the files from the original analysis that are required for the restart must be located in the current working directory. Additionally, this means you cannot perform a restart using SMP processing if DMP processing was used prior to the restart point.

- When running across machines, the job launch procedure (or script) used when restarting a DMP analysis must not be altered following the first load step and first substep. In other words, you must use the same number of machines, the same number of cores for each of the machines, and the same head compute node / other compute nodes relationships among these machines in the restart job that follows.

  Example 2: If you use the following command line for the first load step where the head compute node (which always appears first in the list of machines) is `mach1`, and the other compute nodes are `mach2` and `mach3`:

  ```
  ansys251 –dis –machines mach1:4:mach2:1:mach3:2 –i input –o output1
  ```

  then for the multiframe restart, you must use a command line such as this:

  ```
  ansys251 –dis –machines mach7:4:mach6:1:mach5:2 –i restartjob –o output2
  ```

  This command line uses the same number of machines (3), the same number of cores for each machine in the list (4:1:2), and the same head compute node / other compute nodes relationship (4 cores on the head compute node, 1 core on the second compute node, and 2 cores on the third compute node) as the original run. Any alterations in the `-machines` field, other than the actual machine names, will result in restart failure. Finally, the files from the original analysis that are required for the restart must be located in the current working directory on each of the machines.

- The files needed for a restart must be available on the machine(s) used for the restarted analysis. Each machine has its own restart files that are written from the previous run. The restart process needs to use these files to perform the correct restart actions.

  For Example 1 above, if the two analyses (`-dis –np 8`) are performed in the same working directory, no action is required; the restart files will already be available. However, if the restart is performed in a new directory, all of the restart files listed in Table 4.3: Required Files for Multiframe Restart - Procedure 1 (p. 46) must be copied (or moved) into the new directory before performing the multiframe restart.

  For Example 2 above, the restart files listed in the "Head Compute Node" column in Table 4.3: Required Files for Multiframe Restart - Procedure 1 (p. 46) must be copied (or moved) from `mach1` to `mach7`, and all of the files in the "Other Compute Nodes" column must be copied (or moved) from `mach2` to `mach6` and from `mach3` to `mach5` before performing the multiframe restart.

Table 4.3: Required Files for Multiframe Restart - Procedure 1

Head Compute Node	Other Compute Nodes
`Jobname.ldhi`	--
`Jobname.rdb`	--
`Jobname.rd`*`nn`* if remeshed due to nonlinear adaptivity, where *nn* is the number of remeshings before the restart	--
`Jobname0.`*`xnnn`* [1]	`Jobname`*`n`*`.`*`xnnn`* (where *n* is the process rank and *nnn* is a restart file identifier)
`Jobname0.rst` (this is the local `.rst` file for this domain) [2]	`Jobname`*`n`*`.rst` (this is the local `.rst` file for this domain) [2]

1. The `.`*`Xnnn`* file extension mentioned here refers to the `.R`*`nnn`* and `.M`*`nnn`* files discussed in Multiframe File Restart Requirements in the *Basic Analysis Guide*.

2. The `Jobname`*`n`*`.rst` files are optional. The restart can be performed successfully without them.

4.4.4.2. Procedure 2 - Use a Different Number of Cores

In this procedure, the total number of cores used when restarting a DMP analysis can be altered following the first load step and first substep. In addition, you can perform the restart using either a distributed-memory parallel solution or a shared-memory parallel solution. Some additional steps beyond the typical multiframe restart procedure are required to ensure that necessary files are available.

Note:

This procedure is not available when performing a restart for the following analysis types: mode-superposition transient analysis, an analysis that includes mesh nonlinear adaptivity, and 2D to 3D analysis.

- In this procedure, the `Jobname.r`*nnn* file must be available. This file is not generated by default in DMP processing, even when restart controls are activated via the **RESCONTROL** command. You must either use the command **DMPOPTION**,RNN,YES in the prior (base) analysis, or you must manually combine the `Jobname`*n*`.r`*nnn* files into the `Jobname.r`*nnn* file using the **COMBINE** command.
- For example, if you use the following command line for the first load step:

  ```
  ansys251 -dis -np 8 -i input -o output1
  ```

  then for the multiframe restart, you can use more or less than 8 cores (`-np` *N*, where *N* does not equal 8).
- The files from the original analysis that are required for the restart must be located in the current working directory. If running across machines, the restart files are only required to be in the current working directory on the head compute node.

Table 4.4: Required Files for Multiframe Restart - Procedure 2

Head Compute Node	Other Compute Nodes
`Jobname.ldhi`	--
`Jobname.RDB`	--
`Jobname.RD`*nn* if remeshed due to nonlinear adaptivity, where *nn* is the number of remeshings before the restart	--
`Jobname.R`*nnn*	--
`Jobname.rst` [1]	--

1. The `Jobname.rst` file is optional. The restart can be performed successfully without it.

4.4.4.3. Additional Considerations for the Restart

The advantage of using Procedure 1 (same number of cores) is faster performance by avoiding the potentially costly steps of combining the `Jobname`*n*`.r`*nnn* files and later splitting the `Jobname`*n*`.R`*nnn* (and possibly the `Jobname.rst/rth/rmg`) files during the restarted analysis. The disadvantage is that there are more files that need to be managed during the restart process.

The advantage of Procedure 2 (different number of cores) is less files to manage during the restart process, but at the cost of additional time to combine the local restart files and then, later on, to split this file data during the restarted analysis. Depending on the size of the simulation, this overhead may be insignificant.

In all restarts, the `Jobname.rst` results file (or `Jobname.rth` or `Jobname.rmg`) on the head compute node is recreated after each solution by merging the `Jobname`*n*`.rst` files again.

If you do not require a restart, issue **RESCONTROL**,NORESTART in the run to remove or to avoid writing the necessary restart files on the head compute node and all other compute nodes. If you use this command, the worker processes will not have files such as `.esav`, `.osav`, `.rst`, or `.x000`, in the working directory at the end of the run. In addition, the master process will not have files such as `.esav`, `.osav`, `.x000`, `.rdb`, or `.ldhi` at the end of the run. The program will remove all of the above scratch files at the end of the solution phase (**FINISH** or **/EXIT**). This option is useful for file cleanup and control.

4.5. Example Problem

This section contains a tutorial for running a DMP analysis on a cluster using the Linux platform. To download the input file for this example, click .

The tutorial walks you through setting up your distributed environment and then directly running a sample problem. It is designed so that you can modify settings (such as the number of compute nodes used, number of cores on each compute node, etc.), but we strongly recommend that the first time you run this tutorial, you follow the steps exactly. Once you're familiar with the process, you can then modify the tutorial to more closely match your particular environment. You can use the files generated in future GUI or batch runs.

4.5.1. Example: Running a DMP Analysis on Linux

The following tutorial walks you through the setup of your distributed-memory parallel processing environment and is applicable only to systems running Ansys 2025 R1 on a Linux cluster with Intel MPI or Open MPI. The Ansys 2025 R1 installation includes Intel MPI and Open MPI.

The sample input file, `carrier.inp`, is required to complete the tutorial. Once you've downloaded the file, save the input file to your working directory before beginning the tutorial. You can run the sample problem using the problem setup described here.

Part A: Setup and Run mpitest

Follow the instructions outlined in Configuration Requirements for DMP Processing (p. 25) to ensure your Linux cluster is setup and working properly. Run the mpitest program to confirm that the Ansys software is properly installed and that the cluster environment is setup and working properly.

Part B: Setup and Run a Distributed Solution

1. Type hostname on each machine in the cluster, and note the name of each machine. You may need this name to set up the `.rhosts` file.

2. Start the program using the launcher:

   ```
   launcher251
   ```

3. Select the correct environment and license.

4. Go to the **High Performance Computing Setup** tab. Select **Use Distributed Computing (DMP)**. You must also specify either local machine or multiple hosts. For multiple hosts, use the **New Host** button to add machines to the **Selected Hosts** list. If necessary, you can also run remote shell (rsh) by selecting **Use Remote Shell instead of Secure Shell (rsh instead of ssh)**.
5. Click **Run** to launch Mechanical APDL.
6. In Mechanical APDL, select **File>Read Input From** and navigate to `carrier.inp`
7. The example will progress through the building, loading, and meshing of the model. When it stops, select **Main Menu>Solution>Analysis Type>Sol'n Controls**.
8. On the **Solution Controls** dialog box, click the **Sol'n Options** tab.
9. Select the **Pre-Condition CG solver**.
10. Click **OK** on the **Solution Controls** dialog box.
11. Solve the analysis. Choose **Main Menu>Solution>Solve>Current LS** and click **OK**.
12. When the solution is complete, you can postprocess results as you would with any analysis. For example, you could select **Main Menu>General Postproc>Read Results>First Set** and select the desired result item to display.

4.6. Troubleshooting

This section describes problems which you may encounter while using distributed-memory parallel processing, as well as methods for overcoming these problems. Some of these problems are specific to a particular system, as noted.

4.6.1. Setup and Launch Issues

To aid in troubleshooting, you may need to view the actual MPI run command line. On Linux the command is mpirun, and you can view the command line by setting the **ANS_SEE_RUN_COMMAND** environment variable to 1. On Windows the command is mpiexec, and you can view the command line by setting the **ANS_SEE_RUN** and **ANS_CMD_NODIAG** environment variables to TRUE.

Job fails to launch

The first thing to check when a a DMP job fails to launch is that the MPI software you wish to use is installed and properly configured (see Configuration Requirements for DMP Processing (p. 25)).

Next, if running across multiple machines, ensure that the working directory path is identical on all machines (or that you are using a shared network directory) and that you have permission to write files into the working directory used by each machine.

Finally, make sure that you are running the distributed solution on a homogeneous cluster. The OS level and processors must be identical on all nodes in the cluster. If they are not, you may encounter problems. For example, when running a DMP analysis across machines using Intel MPI, if the involved cluster nodes have different processor models, the program may hang (that is, fail to launch). In this situation, no data is written to any files and no error message is output.

Error executing Ansys. Refer to System-related Error Messages in the Mechanical APDL online help. If this was a DMP run, verify that your MPI software is installed correctly, check your environment settings, or check for an invalid command line option.

You may see this error if ANSYS `Inc\v251\ansys\bin\<`*`platform`*`>` (where `<`*`platform`*`>` is intel or winx64) is not in your PATH.

If you need more detailed debugging information, use the following:

1. Open a Command Prompt window and set the following:

   ```
   SET ANS_SEE_RUN=TRUE
   SET ANS_CMD_NODIAG=TRUE
   ```

2. Run the following command line: **`ansys251 -b -dis -i myinput.inp -o myoutput.out`**.

A DMP analysis fails to launch when running from a fully-qualified pathname.

A DMP analysis will fail if the Ansys 2025 R1 installation path contains a **space followed by a dash** if **`%ANSYS251_DIR%\bin\<`*`platform`*`>`** (where `<`*`platform`*`>` is intel or winx64) is not in the system PATH. Add **`%ANSYS251_DIR%\bin\<`*`platform`*`>`** to the system PATH and invoke **`ansys251`** (without the fully qualified pathname). For example, if your installation path is:

```
C:\Program Files\Ansys -Inc\v251\bin\<platform>
```

The following command to launch a DMP analysis will fail:

```
"C:\Program Files\Ansys -Inc\v251\bin\<platform>\ansys251.exe" -g
```

However, if you add **`C:\Program Files\Ansys -Inc\v251\bin\<`*`platform`*`>`** to the system PATH, you can successfully launch a DMP analysis by using the following command:

```
ansys251 -g
```

A DMP analysis fails to launch when using the Slurm job scheduler.

When using Slurm job scheduler, depending on the cluster configuration for Slurm and the command line syntax for Mechanical APDL, the program may fail to launch and cause a crash in the ansOpenMP (libansOpenMP.so) or Intel OpenMP library (libiomp5.so). This may be avoided by:

- setting the KMP_AFFINITY environment variable to something other than "norespect" (for example, "disabled" or "none") before launching the simulation, or
- by switching to Open MPI (via `-mpi openmpi`).

The required licmsgs.dat file, which contains licensing-related messages, was not found or could not be opened. The following path was determined using environment variable ANSYS251_DIR. This is a fatal error - - exiting.

Check the **ANSYS251_DIR** environment variable to make sure it is set properly. Note that for Windows HPC clusters, the **ANSYS251_DIR** environment variable should be set to \\HEADNODE\Ansys Inc\v251\ansys, and the **ANSYSLIC_DIR** environment variable should be set to \\HEADNODE\Ansys Inc\Shared Files\Licensing on all nodes.

Possible runtime error if you installed an older version of MS MPI

The supported version of MS MPI (listed in Table 4.2: Platforms and MPI Software (p. 26)) is automatically installed when you install Ansys 2025 R1. If you install an older version of MS MPI on your machine, be aware that it can supersede the supported version and may cause runtime errors. If this occurs, you can either:

- uninstall the older version and reinstall the supported version MS MPI (see Installation files for MS MPI on Windows (p. 28)), or
- use Intel MPI instead of MS MPI.

4.6.2. Stability Issues

This section describes potential stability issues that you may encounter while running a DMP analysis.

Recovering from a Computer, Network, or Program Crash

When a distributed-memory parallel processing job crashes unexpectedly (for example, seg vi, floating point exception, out-of-disk space error), an error message may fail to be fully communicated to the master process and written into the output file. If this happens, you can view all of the output and/or error files written by each of the worker processes (e.g., `Jobnamen.out` and/or `Jobnamen.err`) in an attempt to learn why the job failed. In some rare cases, the job may hang. When this happens, you must manually kill the processes; the error files and output files written by all the processes will be incomplete but may still provide some useful information as to why the job failed.

Be sure to kill any lingering processes (Linux: type **kill -9** from command level; Windows: use **Task Manager**) on all cores and start the job again.

Using an alternative MPI version

Ansys chooses the default MPI based on robustness and performance. On rare occasions, you may want to try an alternative MPI (specified by the comand line option `-mpi`) as a workaround if unexpected issues arise (see the table below and Table 4.2: Platforms and MPI Software (p. 26) for supported MPI software versions).

Platform	Default MPI Software	Command Line Option to Specify Alternative MPI Software
Linux	Intel MPI 2021.11.0	`-mpi intelmpi2018` (for Intel MPI 2018.3.222) `-mpi openmpi` (for Open MPI 4.0.5)
Windows 10	Intel MPI 2021.11.0	`-mpi msmpi` (for MS MPI 10.1.12)

"Rename operation failed" Error in DMP Mode

When running in distributed memory parallel (DMP) mode using multiple nodes with Open MPI, you may encounter the following error message: Rename operation failed. If this happens, switch to Intel MPI or use a single compute node.

Job Fails with SIGTERM Signal (Linux Only)

Occasionally, when running on Linux, a simulation may fail with a message like the following:

MPI Application rank 2 killed before MPI_Finalize() with signal 15

forrtl: error (78): process killed (SIGTERM)

This typically occurs when computing the solution and means that the system has killed the Mechanical APDL process. The two most common occurrences are (1) the program is using too much of the hardware resources and the system has killed the Mechanical APDL process or (2) a user has manually killed the job (that is, **kill -9** system command). Users should check the size of job they are running in relation to the amount of physical memory on the machine. Most often, decreasing the model size or finding a machine with more RAM will result in a successful run.

Instability when using the Open MPI Library and Clusters Containing AMD Architecture with a High Core Count

When running a DMP analysis that uses the Open MPI library with clusters containing AMD architecture and Infiniband network interconnects, the program may crash if the total core count is higher than 512. If you encounter the problem, try rerunning with DC transport to fix the issue. Example of the troubleshooting command:

```
ansys251 -mpiopt "-x UCX_TLS=dc,sm,self -x UCX_NUM_PPN=176 -x UCX_NUM_EPS=1056"
```

Note that using `-mpiopt` will unset the following defaults, which should be supplied on the command line manually:

- `-x UCX_NUM_PPN=`*ppn*
- `-x UCX_NUM_EPS=`*np*
- `-x UCX_TLS=rc,sm,self`
- `--map-by socket`
- `--rank-by core`

where *ppn* is the number of processes per node, and *np* is the total number of requested processes.

4.6.3. Solution and Performance Issues

This section describes solution and performance issues that you may encounter while running a DMP analysis.

Poor Speedup or No Speedup

As more cores are utilized, the runtimes are generally expected to decrease. The biggest relative gains are typically achieved when using two cores compared to using a single core. When significant speedups are not seen as additional cores are used, the reasons may involve both hardware and software issues. These include, but are not limited to, the following situations.

Hardware

Oversubscribing hardware

In a multiuser environment, this could mean that more physical cores are being used by multiple simulations than are available on the machine. It could also mean that hyperthreading is activated. Hyperthreading typically involves enabling extra virtual cores, which can sometimes allow software programs to more effectively use the full processing power of the CPU. However, for compute-intensive programs such as Mechanical APDL, using these virtual cores rarely provides a significant reduction in runtime. Therefore, it is recommended you do not use hyperthreading; if hyperthreading is enabled, it is recommended you do not exceed the number of physical cores.

Lack of memory bandwidth

On some systems, using most or all of the available cores can result in a lack of memory bandwidth. This lack of memory bandwidth can affect the overall scalability.

Slow interconnect speed

When running a DMP analysis across multiple machines, the speed of the interconnect (GigE, Infiniband, etc.) can have a significant effect on the performance. Slower interconnects cause each DMP process to spend extra time waiting for data to be transferred from one machine to another. This becomes especially important as more machines are involved in the simulation. See Interconnect Configuration at the beginning of this chapter for the recommended interconnect speed.

Software

Simulation includes non-supported features

The shared and distributed-memory parallelisms work to speed up certain compute-intensive operations in **/PREP7**, **/SOLU** and **/POST1**. However, not all operations are parallelized. If a particular operation that is not parallelized dominates the simulation time, then using additional cores will not help achieve a faster runtime.

Simulation has too few DOF (degrees of freedom)

Some analyses (such as transient analyses) may require long compute times, not because the number of DOF is large, but because a large number of calculations are performed (that is, a very large number of time steps). Generally, if the number of DOF is relatively small, parallel processing will not significantly decrease the solution time. Consequently, for small models with many time steps, parallel performance may be poor because the model size is too small to fully utilize a large number of cores.

I/O cost dominates solution time

For some simulations, the amount of memory required to obtain a solution is greater than the physical memory (that is, RAM) available on the machine. In these cases, either virtual memory (that is, hard disk space) is used by the operating system to hold the data that would otherwise be stored in memory, or the equation solver writes extra files to the disk to store data. In both cases, the extra I/O done using the hard drive can significantly affect performance, making the I/O performance the main bottleneck to achieving optimal performance. In these cases, using additional cores will typically not result in a significant reduction in overall time to solution.

Large contact pairs

For simulations involving contact pairs with a large number of elements relative to the total number of elements in the entire model, the performance of distributed-memory parallelism is often negatively affected. These large contact pairs require a DMP analysis to do extra communication and often cause a load imbalance between each of the cores (that is, one core might have two times more computations to perform than another core). In some cases, using **CNCHECK**,TRIM can help trim any unnecessary contact/target elements from the larger contact pairs. In other cases, however, manual interaction will be required to reduce the number of elements involved in the larger contact pairs.

Different Results Relative to a Single Core

Distributed-memory parallel processing initially decomposes the model into domains. Typically, the number of domains matches the number of cores. Operational randomness and numerical round-off inherent to parallelism can cause slightly different results between runs on the same machine(s) using the same number of cores or different numbers of cores. This difference is often negligible. However, in some cases the difference is appreciable. This sort of behavior is most commonly seen on nonlinear static or transient analyses which are numerically unstable. The more numerically unstable the model is, the more likely the convergence pattern or final results will differ as the number of cores used in the simulation is changed.

Inexact Licensing Charge After Manually Killing a Worker Process from the Operating System

In an elastic licensing environment using DMP mode, terminating a worker process from the operating system (Linux: kill -9 command; Windows: Task Manager) could result in a license charge for up to an hour of usage for any fraction of an hour of actual time used. To avoid the inexact licensing charge, terminate the process from within Mechanical APDL (**/EXIT**) or terminate the master process from the operating system.

Chapter 5: Using Hybrid Parallel Processing (SMP + DMP)

Hybrid parallel processing is a combination of distributed-memory parallel (DMP) (p. 23) and shared-memory parallel (SMP) (p. 11). With hybrid parallel, all processes are run in parallel with the number of processes specified by the `-np` command line option, like DMP. However, hybrid parallel differs from DMP because all processes (master and worker) in the solution phase can have multiple threads per process, specified by the `-nt` command line option.

Enabling multiple threads per process during solution, hybrid parallel offers the following performance improvements:

Reduced memory usage

With hybrid parallel, all computations during solution (including stiffness generation, linear equation solving, and results calculations) are performed in parallel with the ability to use multiple threads per MPI process. Since using more SMP threads does not significantly increase memory use, hybrid parallel reduces memory usage by using less MPI processes per compute node compared to DMP.

More effective use of available hardware

To run larger models on a fixed amount of memory with DMP, you would use less processes per compute node to increase memory and solution efficiency, but this can lead to underutilization of compute nodes in the cluster. Hybrid parallel addresses this problem to fully utilize available hardware and licenses.

Improved scalability of large models with high element load balance ratios

Hybrid parallel can improve efficiency for large models with contact pairs that cannot be split, which reduce the overall scalability of the code at higher core counts. By using SMP to handle contact pairs, it can improve scaling for models with high element load balance ratios since it decomposes into less domains.

Depending on the details of an analysis, hybrid parallel can be faster or slower than DMP. The auto-hybrid feature has been designed based on a set of heuristics to automatically switch to hybrid parallel if the specifics of your analysis indicate it would improve performance.

5.1. Activating Hybrid Parallel Processing

As summarized in Table 1.1: Command Line Options to Specify Number of Processes and/or Threads per Process (p. 4), DMP with the number of processes set to 4 is the default behavior if you do not specify any parallel processing options.

Hybrid parallel can be activated in two ways:

1. You specify multiple threads per process explicitly (see Activating Hybrid Parallel by Command or Product Launcher (p. 56) for details).

2. If no value has been set for `-nt` and the specifics of your analysis indicate that using hybrid parallel will improve performance, the program automatically invokes hybrid parallel (see Auto-hybrid (p. 5) for details).

5.1.1. Activating Hybrid Parallel by Command or Product Launcher

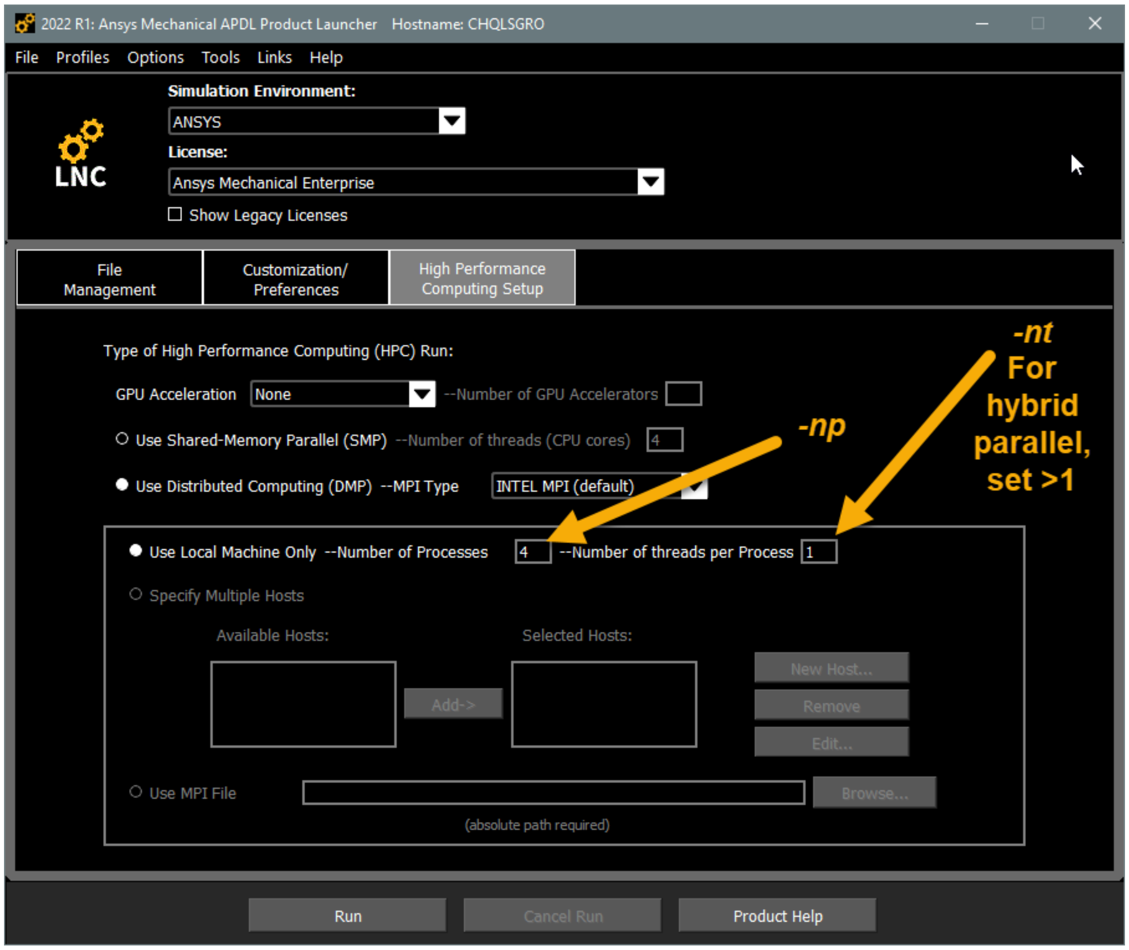
To activate hybrid parallel explicitly, specify multiple threads per process using the `-nt` command line option, which is 1 by default. For example, the following command line

```
ansys251 -b -np 8 -nt 2
```

specifies a hybrid parallel run using 8 MPI processes with 2 threads per process, and by Equation 1.1 (p. 3), the total core count is 16. For a complete list of command line options, see Starting a Mechanical APDL Session from the Command Level in the *Operations Guide*.

Alternatively, you can set the number of processes and number of threads per process in the product launcher. Figure 5.1: Product Launcher - High Performance Computing Setup (p. 56) shows the default values for a DMP run with 4 processes. The arrows in the figure indicate where you can specify the **Number of Processes** (for a DMP or hybrid parallel run) and invoke hybrid parallel by increasing the **Threads per Process** to a value larger than 1.

Figure 5.1: Product Launcher - High Performance Computing Setup


2022 R1: Ansys Mechanical APDL Product Launcher Hostname: CHQLSGRO

Simulation Environment: ANSYS

License: Ansys Mechanical Enterprise

☐ Show Legacy Licenses

Type of High Performance Computing (HPC) Run:

GPU Acceleration: None --Number of GPU Accelerators

○ Use Shared-Memory Parallel (SMP) --Number of threads (CPU cores) 4

● Use Distributed Computing (DMP) --MPI Type: INTEL MPI (default)

● Use Local Machine Only --Number of Processes: 4 --Number of threads per Process: 1

○ Specify Multiple Hosts

○ Use MPI File (absolute path required)

5.1.2. System-Specific Considerations

For hybrid parallel processing, the number of cores that the program uses is limited to the *lesser* of one of the following:

- The number of cores prescribed, calculated by the number of processes multiplied by the number of threads per process.
- The actual number of cores available.

If the number of cores prescribed is larger than the actual number of physical cores available, the number of threads/process will be reduced.

You can specify multiple settings for the number of cores to use during a session. To do so, it is recommended that you issue the **/CLEAR** command before resetting the number of cores for subsequent analyses.