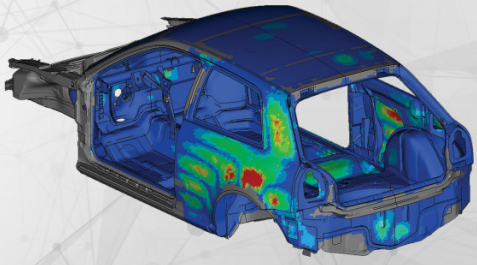
MSC **Nastran** 2019



High Performance Computing User's Guide

MSC Software

Corporate
MSC Software Corporation
4675 MacArthur Court, Suite 900
Newport Beach, CA 92660
Telephone: (714) 540-8900
Toll Free Number: 1 855 672 7638
Email: americas.contact@mscsoftware.com

Europe, Middle East, Africa
MSC Software GmbH
Am Moosfeld 13
81829 Munich, Germany
Telephone: (49) 89 431 98 70
Email: europe@mscsoftware.com

Japan
MSC Software Japan Ltd.
Shinjuku First West 8F
23-7 Nishi Shinjuku
1-Chome, Shinjuku-Ku
Tokyo 160-0023, JAPAN
Telephone: (81) (3)-6911-1200
Email: MSCJ.Market@mscsoftware.com

Asia-Pacific
MSC Software (S) Pte. Ltd.
100 Beach Road
#16-05 Shaw Tower
Singapore 189702
Telephone: 65-6272-0082
Email: APAC.Contact@mscsoftware.com

Worldwide Web
www.mscsoftware.com

Disclaimer

December 17, 2018

NA:V2019:Z:Z:Z:DC-HPC-PDF

Main Index

Documentation Feedback

At MSC Software, we strive to produce the highest quality documentation and welcome your feedback. If you have comments or suggestions about our documentation, write to us at: documentation-feedback@mscsoftware.com.

Please include the following information with your feedback:

- Document name
- Release/Version number
- Chapter/Section name
- Topic title (for Online Help)
- Brief description of the content (for example, incomplete/incorrect information, grammatical errors, information that requires clarification or more details and so on).
- Your suggestions for correcting/improving documentation

You may also provide your feedback about MSC Software documentation by taking a short 5-minute survey at: http://msc-documentation.questionpro.com.

Note:	The above mentioned e-mail address is only for providing documentation specific feedback. If you have any technical problems, issues, or queries, please contact Technical Support.

Contents

MSC Nastran High Performance Computing Guide

Preface

About this Book 8
Recent Developments 8
2019.0 MSC Nastran 8
2018.2 MSC Nastran 8
2018.0 MSC Nastran 9
2017.0 MSC Nastran 9
2016.0 MSC Nastran 9
2014.1 MSC Nastran 9
2014.0 MSC Nastran 9
2013.1 MSC Nastran 9
2013.0 MSC Nastran 9
2012.2 MSC Nastran 10
2012.0 MSC Nastran 10
2011 MD Nastran 10
2010 MD Nastran 10
List of MSC Nastran Books 11
Technical Support 12
Training and Internet Resources 12

1 Using this Manual

Introduction 16
Typographical Conventions 16
Key for Readers 16
The Content of this Manual 16
Chapter 1. Introduction 16
Chapter 2. Optimal Hardware Configuration for MSC Nastran 16
Chapter 3. Optimal Performance of Linear Static Analysis (SOL 101) 17
Chapter 4. Optimal Performance of Normal Mode Analysis (SOL 103) 17
Chapter 5. Optimal Performance of Direct Frequency Response Analysis (SOL 108) 17
Chapter 6. Optimal Performance of Modal Frequency Response Analysis (SOL 111) 17

Main Index

Chapter 7. Optimal Performance of Nonlinear Analysis (SOL 400) . . . 17
Chapter 8. Automatic Solver Selection . . . 18
Chapter 9. General Guidelines . . . 18
Changes to MSC Nastran 2019.0 . . . 18

2 Hardware

Introduction . . . 20
Hardware Details Affecting Simulation Performance . . . 20
Memory . . . 21
Summary . . . 22
Details . . . 22
Determining Your Current System Description . . . 25
Disks . . . 25
Summary . . . 25
Details . . . 26
Determining Your Current System Description . . . 28
CPUs . . . 29
Summary . . . 29
Details . . . 29
Determining Your Current System Description . . . 30
Hardware Accelerators . . . 31
Summary . . . 31
Details . . . 32
Determining Your Current System Description . . . 33
Network Connections . . . 34
Summary . . . 34
Details . . . 34
Determining Your Current System Description . . . 35
References . . . 35

3 Optimal Performance of Linear Static Analysis (SOL 101)

Introduction . . . 38
Memory Usage . . . 38
Memory Guidelines . . . 38
Examples . . . 41
Matrix Solver Options . . . 43
Direct Methods . . . 44
Iterative Methods . . . 46
Examples . . . 46

Parallel Options **49**
Examples 51
Output Request **54**
Example 55
Additional Topics **58**
High Matrix Connectivity 58
Inertia Relief 58
Model and Hardware Details **59**
Decision Tree for Linear Static Analysis **60**

4 Optimal Performance of Normal Mode Analysis (SOL 103)

Introduction **62**
Memory Usage **62**
EigenSolver Options **66**
ACMS 67
Lanczos 67
Method Selection 67
Parallel Options **69**
ACMS 70
Lanczos 71
Output Request **72**
Additional Topics **73**
Sturm Count 73
Number of Roots Found by ACMS 74
Additional Acceleration Options for the Accelerated ACMS Method 74
Models and Hardware Details **75**
Decision Tree for Normal Modes Analysis **76**

5 Optimal Performance of Direct Frequency Response Analysis (SOL 108)

Introduction **78**
Memory Usage **79**
Understanding Memory 79
Examples 81
Matrix Solver Options **83**
Symmetric 84
Unsymmetric 84
Parallel Options **85**

Number of Frequencies and Parallel Performance 86
Solver Options and Parallel Performance 87
Unsymmetric Case 88
GPU scaling 89
Output Request 89
Example 90
Additional Topics 90
Krylov Iterative Method 90
Matrix Iterative Method 91
Model and Hardware Details 91
Decision Tree for Direct Frequency Response Analysis 93

6 Optimal Performance of Modal Frequency Response Analysis (SOL 111)

Introduction 96
Memory Usage 96
SOL 111 using Lanczos 97
SOL 111 using Original ACMS 98
SOL 111 using Accelerated ACMS 99
Solver Options 100
EigenSolver Options 100
Frequency Response Solver Options 101
Parallel Options 103
Accelerated ACMS and Original ACMS 103
Lanczos 104
Output Request 104
MDOTM 105
SPARSEPH 105
SPARSEDR 105
Additional Topics 106
Fluid Eigensolution in 111 (DMP) 106
MDK4OPT 106
DMP parallelization of PFCALC module 107
Definition of Hardware/Models used in Examples 107
Decision Tree for Modal Frequency Analysis 108

7 Optimal Performance of Nonlinear Analysis (SOL 400)

Introduction 111
Memory Usage 111
Understanding Memory 111

Examples . . . 113
Matrix Solver Options . . . 114
Direct Methods . . . 115
Iterative Methods . . . 117
Examples . . . 118
Parallel Options . . . 120
Distributed Memory Parallel . . . 121
Shared Memory Parallel . . . 122
Examples . . . 124
Output Request . . . 126
Example . . . 127
Additional Topics . . . 128
Linear versus Nonlinear Elements . . . 128
LMT2MPC . . . 128
Special Settings Known to Affect Convergence . . . 129
Model and Hardware Details . . . 130
Decision Tree for Nonlinear Static Analysis . . . 131

8 Automatic Solver Selection

Introduction . . . 134
Details . . . 134
Model Size Grouping . . . 134
Solver Selection . . . 135
SOL 101 and 400 . . . 135
SOL 103 and 111 . . . 135
SOL 107 and 108 . . . 135
SOL 200 . . . 136
Parallel Settings . . . 136
Examples . . . 138
SOL 101 . . . 138
SOL 103 . . . 139
SOL 400 . . . 140
Machine Learning . . . 140
Simplified Decision Tree . . . 141
FAQ . . . 145
General Questions . . . 145

9 General Guidelines

System Settings **148**
Interpreting the F04 file **148**
Memory 148
Disk Usage 151
SMP Parallel CPU Usage 152
DMP Parallel Communication 152
Known Errors **152**
General Messages 152
Pardiso 152
ACMS 154
Lanczos 154
FAQ **155**
General Questions 155
Parallel Usage Questions 157
Hardware Questions 158
Memory Questions 159
ACMS and Lanczos Questions 160
Random Solution Sequence Questions 161
References **161**

Preface

- About this Book
- Recent Developments
- List of MSC Nastran Books
- Technical Support
- Training and Internet Resources

About this Book

MSC Nastran is a general purpose finite element program which solves a wide variety of engineering problems. It is developed, marketed, and supported by the MSC Software Corporation.

The MSC Nastran 2018.0 HPC Guide is intended to provide the MSC Nastran user with the knowledge to obtain the best performance out of MSC Nastran for static analysis, eigenvalue analysis, dynamic analysis, and nonlinear analysis. It describes the various options available within MSC Nastran for solving matrix equations, for obtaining parallel performance, and for reducing the costs of reading and writing to disk. It also provides regularly updated information on the best hardware configuration for MSC Nastran based on our experiences with customers and our own internal experiences, in addition to consultation with experts from hardware manufacturers.

This document provides details regarding algorithm options as well as parallel options that are available as of MSC 2018.0. Certain features (e.g., certain iterative solvers) that were prominent in much older releases have been left out of this document and only the latest features are discussed here. There are older user guides and articles on SimCompanion containing information regarding performance, but this document will become the standard for all future releases.

Recent Developments

New versions of MSC Nastran are released periodically. Details about the updates made in each released version are described in the Release Guide provided with MSC Nastran. For the reader's convenience, the major HPC enhancements that have been introduced from 2018.0 (current version) through 2010 are listed below.

Note that the new version of the ACMS method introduction in 2016.0 is referred to as the "Accelerated ACMS" method below and throughout the document.

2019.0 MSC Nastran

- Improved parallel performance for acoustic coupling matrix reduction using ACMS
- Improved performance for the RANDOM module
- New DMP implementation of modal participation factor calculations
- New SMP implementation of FASTFR (Fast Frequency Response)
- Changes to SOLVE=AUTO for SOL 111
- Enhancements to SOLVE=AUTO for SOL 200

2018.2 MSC Nastran

- SOLVE=AUTO may be put in a users RC file.
- CASI=NO may be specified on the command line or RC files.
- Enhanced DMP support in SOL 108 with SOLVE=AUTO.
- Improved performance and scalability for [K4] reduction using ACMS.
- Increased efficiency and reduced I/O for SOL 103 with full data recovery.

- Machine Learning for 3D models has been added for SOLVE=AUTO.

2018.0 MSC Nastran

- Improved parallelization of Accelerated ACMS method.
- Modernized MPYAD method with better parallel performance and method selection.
- Restart in SOL 103/111 or 112 with ACMS method.
- Automatic matrix solver selection option.

2017.0 MSC Nastran

- Reduction in memory of the Accelerated ACMS method by at least 25%.
- New LU solver for AUTOMSET calculations with better robustness.
- Support for the Accelerated ACMS method in External Superelement Analysis.
- DMP support for fluid eigensolution analysis in SOL 111.

2016.0 MSC Nastran

- New SMP parallel for NLEMG in SOL 400
- New Accelerated ACMS method with better SMP scalability
- New Pardiso Solver for SOL 101, 107, 108, 111, and 200 with better SMP performance

2014.1 MSC Nastran

- Upgraded CASI version with SMP scalability.

2014.0 MSC Nastran

- New Pardiso solver for SOL 400 with better SMP performance.
- New Unsymmetric FASTFR method for frequency response analysis.
- Improvements to Random Analysis processing

2013.1 MSC Nastran

- Post-order traversal for ACMS method to reduce I/O and improve performance by 30-50%.

2013.0 MSC Nastran

- Improved performance of DMP parallel for NLEMG in SOL 400.
- Changing memory settings for better defaults via use of mem and buffer pool.

2012.2 MSC Nastran

- New DMP parallel for NLEMG in SOL 400.
- Improvements to buffer pool with respect to I/O performance.

2012.0 MSC Nastran

- Improved use of DMP and SMP in ACMS for better overall performance.
- Improved performance in the unsymmetric sparse direct solver MSCLU.
- Addition of new complex eigenvalue extraction method: IRAM.

2011 MD Nastran

- Improvements in ACMS for the K4 damping and enforced motion in frequency dependent analysis.
- Performance improvements for matrix factorization and matrix multiplication via new math kernels.
- CASI iterative solver introduced for transient thermal analysis in SOL 400.

2010 MD Nastran

- Introduction of serial CASI iterative method in linear and nonlinear contact analysis.
- New DMP matrix-based iterative solver and DMP direct solver for SOL 101 and 400.
- New UMFPACK solver for complex eigenvalue analysis.

List of MSC Nastran Books

A list of some of the MSC Nastran documents is as follows:

Installation and Release Guides
■ Installation and Operations Guide
■ Release Guide
Reference Guides
■ Quick Reference Guide
■ DMAP Programmer's Guide
■ Reference Guide
■ Utilities Guide
Demonstration Guides
■ Linear Analysis
■ Implicit Nonlinear (SOL 400)
■ Explicit Nonlinear (Dytran based SOL 700)
User's Guides
■ Getting Started
■ Linear Static Analysis
■ Dynamic Analysis
■ Embedded Fatigue
■ Embedded Vibration Fatigue
■ Thermal Analysis
■ Superelements
■ Design Sensitivity and Optimization
■ Rotordynamics
■ Implicit Nonlinear (SOL 400)
■ Explicit Nonlinear (SOL 700)
■ Aeroelastic Analysis
■ User Defined Services
■ Non Linear (SOL 400)
■ High Performance Computing
■ DEMATD

You may find any of these documents from MSC Software at:
http://simcompanion.mscsoftware.com/infocenter/index?page=home

Technical Support

For technical support phone numbers and contact information, please visit:
http://www.mscsoftware.com/Contents/Services/Technical-Support/Contact-Technical-Support.aspx

Support Center (http://simcompanion.mscsoftware.com)

The SimCompanion link above gives you access to the wealth of resources for MSC Software products. Here you will find product and support contact information, product documentations, knowledge base articles, product error list, knowledge base articles and SimAcademy Webinars. It is a searchable database which allows you to find articles relevant to your inquiry. Valid MSC customer entitlement and login is required to access the database and documents. It is a single sign-on that gives you access to product documentation for complete list of products from MSC Software, allows you to manage your support cases, and participate in our discussion forums.

Training and Internet Resources

MSC Software (www.mscsoftware.com)

MSC Software corporate site with information on the latest events, products and services for the CAD/CAE/CAM marketplace.

http://simcompanion.mscsoftware.com

The SimCompanion link above gives you access to the wealth of resources for MSC Software products. Here you will find product and support contact information, product documentations, knowledge base articles, product error list, knowledge base articles and SimAcademy Webinars. It is a searchable database which allows you to find articles relevant to your inquiry. Valid MSC customer entitlement and login is required to access the database and documents. It is a single sign-on that gives you access to product documentation for complete list of products from MSC Software, allows you to manage your support cases, and participate in our discussion forums.

http://www.mscsoftware.com/msc-training

The MSC-Training link will point you to the schedule and description of MSC Seminars. The following courses are recommended for beginning MSC Nastran users.

NAS101A - Linear Static and Normal Modes Analysis using MSC Nastran

This course serves as an introduction to finite element analysis (FEA). It discusses the basic features available in MSC Nastran for solving structural engineering problems. In this course, all finite element models will be created and edited using a text editor, not a graphical pre-processor. Proper data structure of the MSC Nastran input file is covered. At the conclusion of the course, you will be familiar with fundamental usage of MSC Nastran.

NAS101B - Advanced Linear Analysis using MSC Nastran

This course is a continuation of NAS101A (Linear Static and Normal Modes Analysis using MSC Nastran). In this course, you will learn the following:

- Theory of buckling analysis
- How to perform a buckling analysis about rigid elements - MPC, RBAR,RBE2, and RBE3
- Modeling with interface element CINTC and connectors
- Lamination theory and composite materials
- MSC Nastran composite theory
- Failure theories
- Linear contact and permanent glued contact
- Different model checks
- Modeling tips and tricks

NAS120 - Linear Static Analysis using MSC Nastran and Patran

This course introduces the basic finite element analysis techniques for linear static, normal modes, and buckling analysis of structures using MSC Nastran and Patran. MSC Nastran data structure, the element library, modeling practices, model validation, and guidelines for efficient solutions are discussed and illustrated with examples and workshops.

Patran will be an integral part of the examples and workshops and will be used to generate and verify illustrative MSC Nastran models, manage analysis submission requests, and visualize results. This course provides the foundation required for intermediate and advanced MSC Nastran applications.

NAS122: -Basic Dynamic Analysis using MSC Nastran and Patran

The course covers a wide range of dynamic analysis topics from basic to advanced using an integrated approach. Patran is used for data set up and post-processing and MSC Nastran is used for the solver. It conyains the following information:

- Many unique practical hints and tips.
- Case studies for each topic to help understand the physics and engineering behind the techniques in a practical way.
- A comprehensive set of over 20 fully detailed student workshops,to obtain real "hands on" experience.

A strong emphasis is placed on engineering process so that you can rapidly relate the course with respect to yourproject requirements.

NAS127: Rotordynamic Analysis using MSC Nastran

This course covers rotordynamic analysis for coupled rotating and stationary components like jet engines, turbines, compressors, energy storage devices, etc. This course provides details for:

- How to setup and analyze structural models with one or more rotating components
- The types of analysis supported by the rotordynamics capability

  - Static analysis
  - Complex eigenvalue analysis (modal and direct)
  - Frequency response (modal and direct) and nonlinear frequency response
  - Transient response (direct linear and nonlinear)
- Damping effects and input methods for models with rotating components
- How to use Patran to create models and display results with animation or graphs

There are two sets of workshops for this course. One set uses Patran as the pre- and post-processor. The other directly edits the MSC Nastran input file.

1 Using this Manual

- Introduction 16
- Typographical Conventions 16
- The Content of this Manual 16
- Changes to MSC Nastran 2019.0 18

Introduction

The chapter provides a brief overview of the typographical conventions used in the document to help the user better follow the MSC Nastran HPC Guide. It also provides an overview of the structure of the document and a list of changes to MSC Nastran for the given release associated with this document.

Typographical Conventions

This section describes some syntax that will help the user in understanding text in the various chapters.

Key for Readers

This guide uses certain stylistic conventions to denote user action, to emphasize particular aspects of a MSC Nastran run or to signal other differences within the text.

Courier New	Represents command-line options of MSC Nastran and results from f04/f06 files. Example: `nast20180 memorymax=16gb myjob.dat`
Quoted Text	Represents command-line options of MSC Nastran for in-line text. Example: “memorymax=16gb”
Red Text	Represents items in the text that we want readers to emphasize. Example: smp=16
Bold Text	Represents items in the text that we want to emphasize. Example: **dmp=4**

The Content of this Manual

The guide is divided into eight chapters and one Appendix. A brief description of each of these follows:

Chapter 1. Introduction

This chapter provides an introduction to the manual and describes some of the formatting options that will be used throughout the manual. We also give a more thorough description of the performance improvements associated with the release associated with this version of the MSC Nastran HPC Guide.

Chapter 2. Optimal Hardware Configuration for MSC Nastran

This chapter provides advice to the MSC Nastran user on the type of hardware to purchase to get the best performance for MSC Nastran. The critical question of “where should money be spent” when purchasing a new system is discussed in the context of memory, CPU speed, and disk speed. This question is addressed with respect to the types of simulations that are performed as well as whether the user is purchasing a Linux or Windows system.

Chapter 3. Optimal Performance of Linear Static Analysis (SOL 101)

The chapter focuses specifically on linear static analysis in SOL101 where the user is interested in observing the structural response to loading conditions. In linear statics, the dominant part of the simulation is factorizing the linear system of equations to get the displacements from the given load. We will describe in this chapter the correct method choice for a given scenario to get optimal performance. We also describe how memory should be chosen if there is a large output request as well as when to use iterative solvers. Finally, as with all chapters, we provide a decision chart to give the guide the user on memory settings, method selection, and parallel settings.

Chapter 4. Optimal Performance of Normal Mode Analysis (SOL 103)

This chapter focus on normal mode analysis in SOL 103. There are two dominant choices in SOL 103 the standard Lanczos method and the ACMS method. For the ACMS method, there is the Original ACMS method and the New and Accelerated ACMS method. In this chapter, we will give guidance on when to use these two methods along with how to get the best performance out of these methods. We will specifically cover the best memory and parallel settings for each method. Finally, as with Chapter 3, we will provide a decision chart to give the user an easy guide on memory settings, method selection, and parallel settings.

Chapter 5. Optimal Performance of Direct Frequency Response Analysis (SOL 108)

This chapter focuses on direct frequency response analysis. The performance in SOL 108 is dominated by the time to solve the frequency response problem. The matrix problem in SOL 108 is much larger than in SOL 101 as the frequency response nature of the problem yields complex matrices. Furthermore, one matrix solution must be computed for each of the desired forcing frequencies. This is also a special case where DMP parallel is very effective and we provide details on the corresponding performance in this chapter. We again offer a decision chart to help guide the user on memory settings, method selection, and parallel settings.

Chapter 6. Optimal Performance of Modal Frequency Response Analysis (SOL 111)

This chapter focuses on modal frequency response analysis or SOL 111. In SOL 111, the performance is based on both the calculation of the modes (i.e., SOL 103) as well as the solution of the frequency response problem. We do not go into the details on the best way to find the modes but instead we refer to the reader to Chapter 4. The matrix problem in the frequency response problem for SOL 111 is several orders of magnitude smaller than in SOL 108. As with other Chapters, we will provide a decision chart to help guide the user on memory settings, method selection, and parallel settings.

Chapter 7. Optimal Performance of Nonlinear Analysis (SOL 400)

In SOL 400, the performance is based on computing element matrices, computing stress recovery, and solving the global stiffness matrix. Thus, we will describe in this chapter the optimal way to complete these operations. Through a combination of parallel settings, memory settings, and method selection, we will

guide the user to the best performance for their problem. We again provide a decision chart to allow users to directly determine their best options without a lengthy exposition on the meaning of these decisions.

Chapter 8. Automatic Solver Selection

This chapter focuses on the Automatic Solver Selection. Selecting specific solvers, memory partitions and parallel specifics can be challenging to users. Specifying `solve=auto` allows users a simple way to select the ideal settings automatically. The settings are derived from the implementation of Machine Learning for memory prediction for the Pardiso solver as well as a wealth of heuristics from MSC performance tests.

Chapter 9. General Guidelines

In the general guidelines section, we provide a summary of system settings, interpreting the F04 file to help in optimizing performance, and a known errors list. We also provide a large set of frequently asked questions and with answers.

Changes to MSC Nastran 2019.0

The following HPC features were added in the MSC Nastran 2019.0 release:

- **Improved parallel performance for acoustic coupling matrix reduction using ACMS**. Reducing the acoustic coupling matrix can be very expensive for mid- to high-frequency acoustic analysis. Performance improvements to the MDHREDX module speed up this process.
- **Improved performance for the RANDOM module**. For typical use cases, a 10X speedup is possible in Version 2019.0.
- **New DMP implementation of modal participation factor calculations**. Near linear parallel speedup is observed up to DMP=4 for real use cases.
- **New SMP implementation of FASTFR (Fast Frequency Response)**. Improved parallel efficiency is achieved when running the new FSTFR module in SMP mode.
- **Changes to SOLVE=AUTO for SOL 111**. In SOL 111, SOLVE=AUTO now considers memory available and will either increase SMP if there is sufficient memory, or decrease DMP if there is insufficient memory. This results in better overall resource utilization.

2 Hardware

- Introduction 20
- Hardware Details Affecting Simulation Performance 20
- Memory 21
- Disks 25
- CPUs 29
- Hardware Accelerators 31
- Network Connections 34

Main Index

Introduction

In 1971, when MSC Software first offered a version of MSC Nastran, the fastest supercomputer in the world was the CDC 7600 designed by Seymour Cray [1]. It had a clock speed of 36.4 MHz and a 65 Kword primary memory with a peak flop rate of 36 MFLOPS. The CDE 7600 was slightly faster that the IBM System/360 mainframe computer, which was architected by Gene Amdahl. For MSC Nastran, the limiting factor in those days was the memory available to solve reasonably large problems. In the first few versions, customers were happy to solve a linear statics FE problem with 100 DOFs in 12 hours.

In 2016, the fastest supercomputer in the world is the Sunway Taihulight supercomputer developed by China's National Research Center of Parallel Computer Engineering and Technology. It has available over 3 million processor cores to deliver a peak flop rate of 93 PETAFLOPS. While the number of processors cores has changed in today's supercomputers and the FLOP rates have greatly increased, software products like MSC Nastran still have similar demands on them that limit performance or model size or both.

In this section, we will discuss how a computer's hardware affects the performance of MSC Nastran and how that impacts the type of hardware that a customer should invest in when they are using MSC Nastran for finite element analysis. We will focus on memory (RAM), disk type, CPU, accelerators, and network connections.

Hardware Details Affecting Simulation Performance

The performance of computer-aided engineering (CAE) simulations is dependent on a computer's hardware as much as it is dependent on the underlying algorithms. I/O performance, memory bandwidth, memory capacity, and flops/core generally defines the overall performance of a single computer used for CAE simulations.

The typical configuration for many users of CAE software is to either run on a single, separate workstation consisting of the above configuration or to run on an HPC Cluster used by many employees at the engineering company where these nodes are "clustered" together in a network. Communication costs between these separate nodes is measured by network bandwidth and latency.

The impact of the hardware, whether it is network latency or CPU speed, is felt differently for different applications due to the underlying algorithms. A very enlightening image was published at The Next Platform that we have reproduced in Figure 2-1. We can see that if we are modeling structures, as with MSC Nastran SOL 101, or crash, as with MSC Nastran SOL 700, or fluids, as with Cradle Software, then we get a very different dependency on performance with respect to network latency, network bandwidth, flops/core, cores, memory capacity, memory bandwidth, and I/O performance.

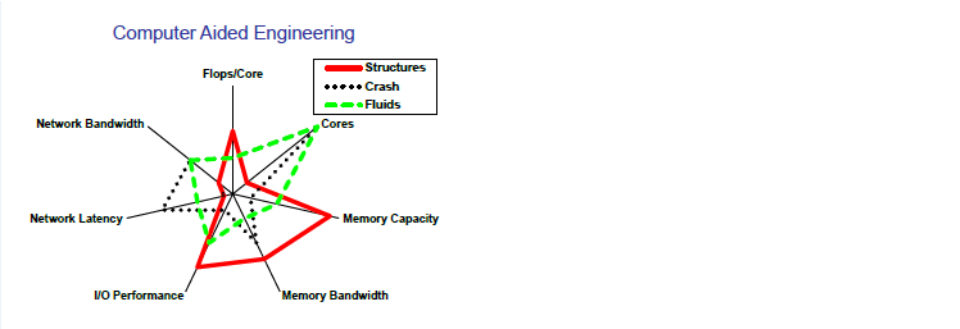


Figure 2-1 Relative importance of various different hardware features for structural analysis simulations, crash simulations, and fluid simulations.

As observed in Figure 2-1, fluid simulations get the most benefit from more cores because fluid simulations are using explicit methods that are highly parallelizable. It is for this reason that one often observes massively parallel computing (1000s or 10,000s of processors) used with fluid simulations. Crash simulations also depend on explicit calculations that are highly parallelizable and thus one can see from Figure 2-1 that cores are most important.

The structural simulations described in this image generally correspond to MSC Nastran simulations, in particular, SOL 101, 103, and 400. The dominant aspect of these simulations is usually the matrix factorization of the stiffness matrix, which represents how the corresponding structural displacements correspond to loads. If the matrix and the factorization all fit within existing memory, then memory bandwidth is critical in determining performance. Often the matrix and factorization process does not fit in existing memory and so I/O performance and memory capacity are critical. This can be the case with MSC Nastran although we will see in the remainder of this HPC Manual that there are times where flops/core or cores are equally beneficial.

In the remainder of this section, we describe in general terms how memory performance, CPU performance, I/O details, and parallel capabilities affect simulation performance.

Memory

Memory capacity and bandwidth is perhaps the most underestimated way to improve performance in FE analysis. Increasing memory capacity allows more of the problem to be solved in-core where access speeds can be orders of magnitude faster than out-of-core where disk speeds matter. Memory can also serve as a cache for I/O by the operating system, or as you will see below, by MSC Nastran itself with the use of Buffer Pool.

Summary

Memory may be purchased as DDR3/DDR4. Below is a table comparison:

Table 2-1 Speeds of various memory types.

Memory Type	Speed
DDR3	800-2133Mb/s
DDR4	1600-3200Mb/s

Additional memory may significantly reduce elapsed times for large models. The following is the chart of recommended memory for linear or nonlinear statics models. Dynamic models may also perform better with additional memory but it is more problem dependent and harder to gauge based on DOFs as it also depends on frequency range in the dynamic problem.

Table 2-2 Memory requirements

DOF	Memory
< 100,000	20 GB
< 1,000,000	64 GB
< 10,000,000	128 GB
< 20,000,000	256 GB
< 40,000,000	512 GB
< 100,000,000	1TB

Details

MSC Nastran may end up doing a significant amount of I/O. However, it has the ability to store its I/O in an internal cache referred to as Buffer Pool. In an MSC Nastran simulation, Buffer Pool amount can be defined by the following command line option:

```
nast20180 mem=40gb bpool=20gb
```

where this is a request for 40 GB of total memory with 20 GB being used by Buffer Pool. In Figure 2-2, we describe MSC Nastran's memory allocation. Note that Buffer Pool was improved in MSC Nastran 2013.0 along with the "mem=max" option. Previous to MSC Nastran 2013.0, there was a Buffer Pool option but its performance was suboptimal.

The "mem=40gb" command is executed in MSC Nastran via a "malloc" of "memory" (leaving the rest of memory for the OS) and then 20 GB of that memory is given to Buffer Pool. The details can be found in the F04 file and look like the following:

```
USER OPENCORE (HICORE)          =   588 MB
EXECUTIVE SYSTEM WORK AREA      =     7 MB
MASTER(RAM)                     =     2 MB
SCRATCH(MEM) AREA               =     6 MB(100 BUFFERS)
```

```
BUFFER POOL AREA (BPOOL4)       = 15779 MB(252434 BUFFERS)

TOTAL MSC NASTRAN MEMORY LIMIT = 16384 MB
```

The small amounts given to the executive system work area, master, scratch mem are of very little concern and users should focus on what is given to Open Core and Buffer Pool. These vary for each problem and memory amount and can affect performance.

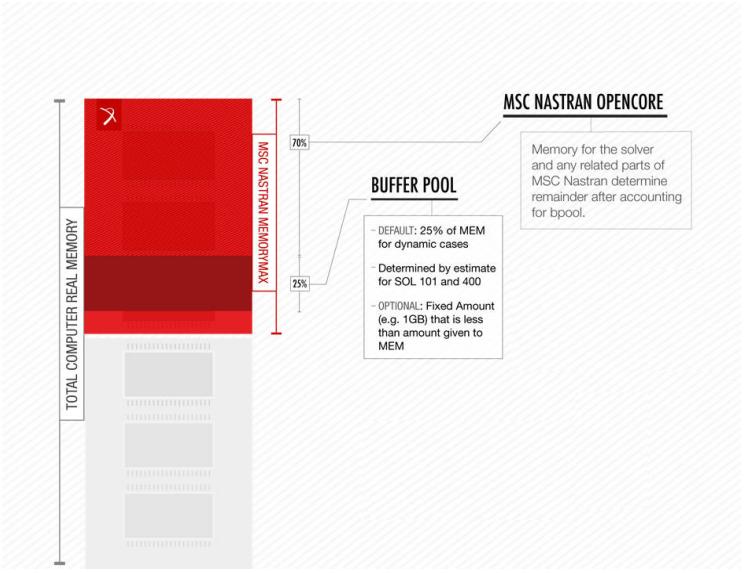


Figure 2-2 Memory Layout for MSC Nastran simulation

Having more memory on the machine allows more memory to be given to the Buffer pool. Figure 2-3, is a chart of elapsed time along with memory layout for a SOL 101 contact model with 2M DOFs run on three machines with identical hardware configuration except different memory amounts.

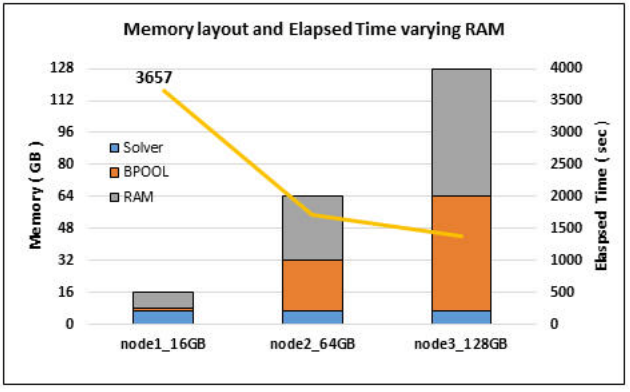


Figure 2-3 Example of Memory Layout (bar chart) and Performance (yellow line) for Varying Memory for SOL 101 Model

We see that increasing memory from 16 GB to 64 GB resulted in a 53% reduction of elapsed time. Increasing memory to 128 GB results in an overall reduction of 63% of elapsed times. The amount of time reduced is greatly dependent on the model. The “Database Usage Statistics” at the bottom of the F04 file may be used to evaluate how much additional memory may help. For example:

```
---------------- LOGICAL DBSETS ---------------+   +------------------------------ DBSET FILES ---------------------------+
 DBSET     ALLOCATED  BLOCKSIZE       USED    USED     FILE       ALLOCATED  ALLOCATED   HIWATER     HIWATER  I/O TRANSFERRED
            (BLOCKS)    (WORDS)   (BLOCKS)       %                (BLOCKS)        (GB)   (BLOCKS)        (GB)             (GB)
 MASTER        25000       8192         91    0.36    MASTER        25000        1.53       1803       0.110            0.111
 DBALL      10000000       8192         42    0.00    DBALL      10000000      610.35         42       0.003            0.003
 OBJSCR        25000       8192        423    1.69    OBJSCR        25000        1.53        423       0.026            0.033
 SCRATCH     8350100       8192        411    0.00   (MEMFILE         100        0.01        100       0.006            0.000)
                                                      SCRATCH     8000000      488.28     214653      13.101           37.551
                                                      SCR300       350000       21.36     277801      16.956          553.170
                                                                                                              ==============
                                                                                                  TOTAL:              590.869
```

We see in the above F04 file that this simulation had 590 GB (37 + 553) of I/O that would fit in 30 GB (13.1+16.9) of memory. Thus, if Buffer Pool had 30 GB of memory, then the I/O transferred would go to 0.0. Generally, setting Buffer Pool to 75% of HIWATER mark, i.e., "bpool=24gb" for this case, is sufficient to see noticeable benefit from Buffer Pool.

Determining Your Current System Description

Linux:

The amount of available memory may be determined with the "free" command:

```
hpccluster <87> free -g -t
              total        used        free      shared  buff/cache   available
Mem:            251         188          44           0          18          62
Swap:            59           3          56
Total:          311         191         100
```

The speed of the memory may be determined by logging on as "root" and issuing "/usr/sbin/dmidecode":

```
Memory Device
        Array Handle: 0x1000
        Error Information Handle: Not Provided
        Total Width: 72 bits
        Data Width: 64 bits
        Size: 16384 MB
        Form Factor: DIMM
        Set: 1
        Locator: A1
        Bank Locator: Not Specified
        Type: <OUT OF SPEC>
        Type Detail: Synchronous Registered (Buffered)
        Speed: 2133 MHz
```

Windows:

The amount of memory available on Windows may be determined by clicking on the "system" tag:

System	
Processor:	Intel(R) Xeon(R) CPU E5-2650 v2 @ 2.60GHz 2.59 GHz (2 processors)
Installed memory (RAM):	128 GB
System type:	64-bit Operating System, x64-based processor
Pen and Touch	No Pen or Touch Input is available for this Display

Disks

Once MSC Nastran exceeds the system memory cache buffer, the I/O will go to disk. Since finite element analysis can be intensive, the disk configuration may impact the performance of the analysis. Improving disk performance can be important to MSC Nastran performance.

Summary

- Do NOT use a network drive.
- SSD disks are very fast and highly recommended for scratch space.
- Faster disks, e.g., 15000 RPM, may help performance.

- Run with scr=yes if you do not need to save a database.
- RAID 0 with multiple disks for SDIR is recommended. Other levels of RAID (for example, RAID 5) have redundancy which is not needed for scratch files and may have some performance degradation.

Details

SSD comparison:

Local SSDs are notably faster than standard disks. The following are results for MSC Nastran performance of two jobs comparing HDD and SSD disks:

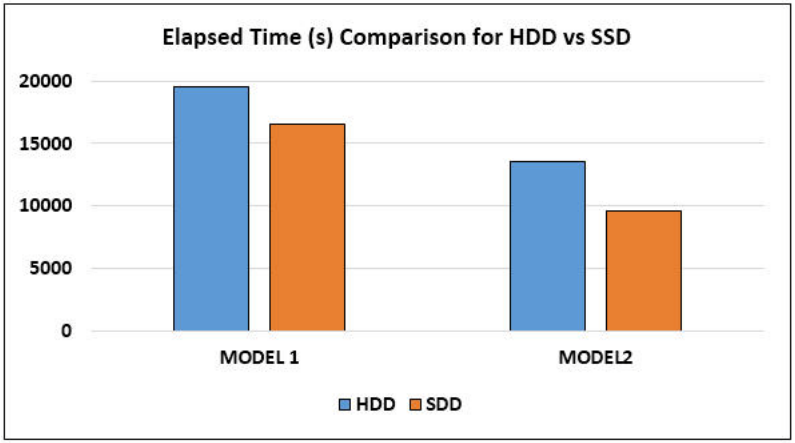


Figure 2-4 Comparison of Hard Disk Drives (HDD) to Solid State Drives (SSD) for two SOL 400 Models

RAID Comparison:

MSC Nastran HPC team recommends scratch disks to be configured with RAID 0. RAID 0 has no redundancy, but for scratch files is fastest. Below is a chart comparing the percent reduction of elapsed time for different RAID 0 configurations, with different numbers of disks, for various jobs.

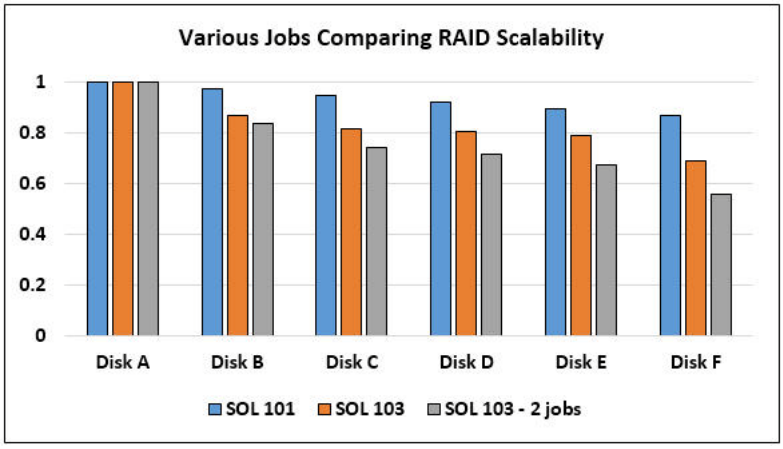


Figure 2-5 Performance comparison of RAID 0 scalability for various MSC Nastran jobs where the SOL 101 job is 4M DOFs and the SOL 103 job is 5M DOFs. The SOL 103 job is also ran twice on the same node to compare a different type of performance.

where

```
Disk A -  6 disks
Disk B -  8 disks
Disk C - 12 disks
Disk D - 14 disks
Disk E - 16 disks
Disk F - 28 disks
```

Determining Your Current System Description

Linux:

Disk Type may be determined by looking at /proc/scsi/scsi. E.g.:

```
[Node]# cat /proc/scsi/scsi
    Attached devices:

Host: scsi0 Channel: 00 Id: 00 Lun: 00
      Vendor: ATA      Model: SSD2SC240G3LC726 Rev: 524A
      Type:   Direct-Access          ANSI  SCSI revision: 05
Host: scsi1 Channel: 00 Id: 00 Lun: 00
      Vendor: ATA      Model: WDC WD1002FAEX-0 Rev: 05.0
      Type:   Direct-Access            ANSI  SCSI revision: 05
```

HDPARM can be used on Linux systems for a simple comparison of disk speed. Below are examples from an SSD disk and a HDD disk. The SSD disk is listed at 480 Mb/sec while the HDD is listed at 123 Mb/sec. There are also other tools (e.g. IOZONE).

```
[Node]# hdparm -t /dev/sda1 # SSD
/dev/sda1:
Timing buffered disk reads: 1442 MB in  3.00 seconds = 480.08 MB/sec
[Node]# hdparm -t /dev/sdb1    # Standard disk
/dev/sdb1:
Timing buffered disk reads: 372 MB in  3.02 seconds = 123.31 MB/sec
```

Creating a RAID 0 device may be done with mdadm:

```
mdadm  - create  - verbose /dev/md0  - level=raid0  - raid-devices=4 /dev/sdb1
/dev/sdc1 /dev/sdd1 /dev/sde1
```

Windows:

Disk type on Windows 7 may be found by clicking on:

```
Start => All Programs => Accessories => System Tools => System Information =>
Components => Storage => Disks
```

Item	Value
Description	Disk drive
Manufacturer	(Standard disk drives)
Model	WDC WD1003FZEX-00MK2A0 ATA Device

CPUs

The MSC Nastran HPC team works directly with hardware vendors to take advantage of the newest hardware options as they are released. While performance improvement is often directly proportional to clock speed, chipset changes like AVX512 provide gains in performance improvement as they allow greater vectorization implying more numerical calculations can be performed in a given clock cycle.

Summary

- Skylake chips with AVX512 may perform about10% faster than Haswell/BroadWell chips with the same clock speed.
- Haswell chips with AVX2 may perform about 10% faster than Sandy Bridge/Ivy Bridge with the same clock speed (incorporated MSC Nastran 2014.0).
- Hyper-threading may degrade MSC Nastran performance. It can be turned it off in the BIOS.
- Some BIOS settings turn down clock settings for minimizing energy use. For maximum performance you may want to adjust this setting.
- AVX2 is not recognized on RedHat 6.4 and below distributions.
- The number of cores desired greatly depends on solution number, number of simultaneous jobs, output requests, etc. Amdahl's Law[3] reminds the user that the increase of performance as a result of additional processes may be minimized if a problem is dominated by non parallel sections (like large output requests). See later chapters for specifics.

Details

CPU selection may affect MSC Nastran Performance. Below is a brief summary of Intel chips as well as instructions used:

Table 2-3 Vector instruction set support and RHEL OS support.

Name	Circa	Comments	OS Support
Sandy Bridge	2011	AVX	
Ivy Bridge	2012	AVX	
Haswell	2013	AVX2	RHEL 6.5
Broadwell	2015	AVX2	RHEL 6.5
Skylake	2015	AVX512	RHEL 7.3

The following figure shows the performance comparison of a combination of several jobs on different CPUs with different vector instruction sets:

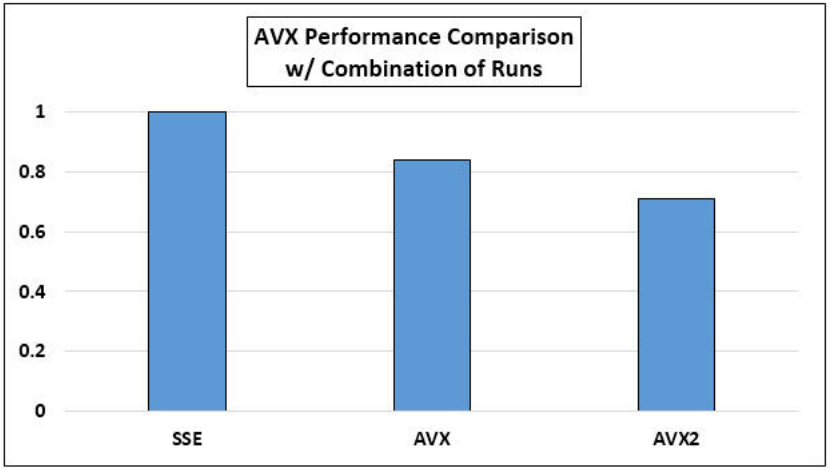


Figure 2-6 Performance comparison of various vector instruction sets.

Determining Your Current System Description

Linux:

Processor type may be determined by "**`cat /proc/cpuinfo`**". For example:

```
processor       : 0
vendor_id       : GenuineIntel
cpu family      : 6
model           : 79
model name      : Intel(R) Xeon(R) CPU E5-2697 v4 @ 2.30GHz
stepping        : 1
microcode       : 0xb00000e
cpu MHz         : 2301.000
cache size      : 46080 KB
physical id     : 0
siblings        : 18
core id         : 0
cpu cores       : 18
apicid          : 0
initial apicid  : 0
fpu             : yes
fpu_exception   : yes
cpuid level     : 20
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat
                  pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx
                  pdpe1gb rdtscp lm constant_tsc arch_perfmon pebs bts rep_good nopl
                  xtopology nonstop_tsc aperfmperf eagerfpu pni pclmulqdq dtes64
                  monitor ds_cpl vmx smx est tm2 ssse3 fma cx16 xtpr pdcm pcid dca
```

```
                sse4_1 sse4_2 x2apic movbe popcnt tsc_deadline_timer aes xsave avx
                f16c rdrand lahf_lm abm 3dnowprefetch ida arat epb xsaveopt pln pts
                dtherm tpr_shadow vnmi flexpriority ept vpid fsgsbase tsc_adjust
                bmi1 hle avx2 smep bmi2 erms invpcid rtm rdseed adx smap
bogomips        : 4589.39
clflush size    : 64
cache_alignment : 64
address sizes   : 46 bits physical, 48 bits virtual
power management:
```

Windows:

Processor type may be determined with the Windows "wmic" command. For example:

```
D:>wmic cpu get /format:list | findstr Name
CreationClassName=Win32_Processor
Name=Intel(R) Xeon(R) CPU E5-2650 v2 @ 2.60GHz
```

Hardware Accelerators

Performance improvements derived from Hardware Accelerators vary a great deal. While in some cases, GPUs may greatly increase performance, in other cases, GPUs may degrade performance. Thus, caution should be used in predicting the amount of potential performance gain.

Summary

- GPUs help with large dense models.
- GPUs degrade performance for ACMS, thus they are disabled for ACMS.
- Static analysis for which GPUs help may benefit more from the CASI solver in cases where the CASI solver is allowed.
- In many cases using a high number of SMP processors may benefit as much as a GPU.
- Multiple GPUs are only used when MSC Nastran is running DMP. The command line option as the form: `gpuid=0,1`

  Where the numbers specified are the minor numbers from the "nvidia-smi –q" command.
- GPU performance will improve the following system cell settings. These settings will happen automatically in 2016 for GPU jobs:

```
system(655)=32 system(656)=1024 system(205)=320 system(219)=384 system(220)=320 system(221)=320
```

- MSC Nastran does not support the Intel PHI.

Details

The following images shows the several jobs comparing SMP/DMP/GPU for different solution sequences with MSC Nastran 2014.1. These jobs are large enough to take advantage of the GPU. Job descriptions are below the graphs. Jobs were run on a machine with 128 GB with "mem=max".

Table 2-4 Descriptions of models for use in testing in figures below.

Name	DOF	SOL	MEM(Serial)	BPOOL(Serial)
XX0RST0	1,296,377	101	64 GB	60 GB
XX0VMD0	2,604,102	103	64 GB	16 GB
XL0DF10	601,656	108	64 GB	16 GB
XX0ZN40	2,077,956	400	64 GB	53 GB

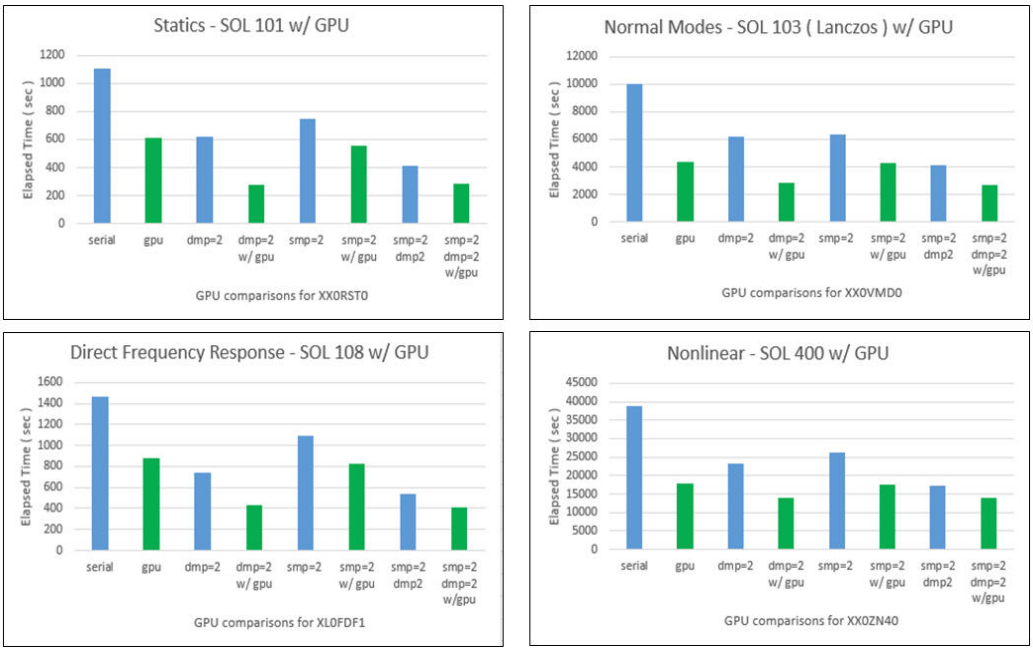


Figure 2-7 Performance comparison of various models using DMP, SMP, and GPU.

Determining Your Current System Description

Linux:

GPU type may be found with the “nvidia-smi” command:

```
node <80> /usr/bin/nvidia-smi
```

```
Tue Oct 21 17:38:36 2014
+-----------------------------------------------------+
| NVIDIA-SMI 396.26 Driver Version: 396.26 |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  Tesla K40m          Off  | 0000:24:00.0     Off |                    0 |
| N/A   45C    P0    70W / 235W |    364MiB / 11519MiB |     65%      Default |
+-------------------------------+----------------------+----------------------+
|   1  Tesla K40m          Off  | 0000:27:00.0     Off |                    0 |
| N/A   27C    P8    19W / 235W |     56MiB / 11519MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Compute processes:                                                GPU Memory |
|  GPU       PID  Process name                                      Usage      |
|=============================================================================|
|    0     41505  ...sc20140_beta2/msc20140/linux64_rhe63i4/analysis   307MiB |
+-----------------------------------------------------------------------------+
```

Windows:

```
C:>"c:\Program Files\NVIDIA Corporation\NVSMI\nvidia-smi.exe"
```

```
Mon Apr 18 12:34:14 2016
+-----------------------------------------------------+
| NVIDIA-SMI 353.90     Driver Version: 353.90        |
|-------------------------------+----------------------+----------------------+
| GPU  Name            TCC/WDDM | Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  Tesla K40m          TCC  | 0000:24:00.0     Off |                    0 |
| N/A   33C    P8    19W / 235W |     23MiB / 11519MiB |    ERR!      Default |
+-------------------------------+----------------------+----------------------+
|   1  Tesla K40m          TCC  | 0000:27:00.0     Off |                    0 |
| N/A   27C    P8    19W / 235W |     23MiB / 11519MiB |    ERR!      Default |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                        GPU Memory |
|  GPU       PID   Type   Process name                              Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

Network Connections

Network connections effects analysis in two ways:

1. If a directory containing files is remotely mounted;
2. If an analysis is running DMP over a network.

Summary

The largest performance gains from MSC Nastran by using multiple nodes is that each process will have an increase of RAM available by dispersing the runs amongst different computers. If an analysis can take advantage of the additional RAM on multiple nodes, then faster networks may help for problem types that have a substantial amount of communication. Refer to CHAPTER 5 on Direct Frequency Response Analysis for the best cases of using multiple nodes to accelerate your MSC Nastran simulations.

Details

Using multiple nodes will require:

- Passwordless “ssh” to be working
- The hosts to be specified on the command line, e.g., “hosts=host1:host2”

Below is a comparison of a large SOL 103 run using multiple nodes comparing standard Ethernet to Infiniband:

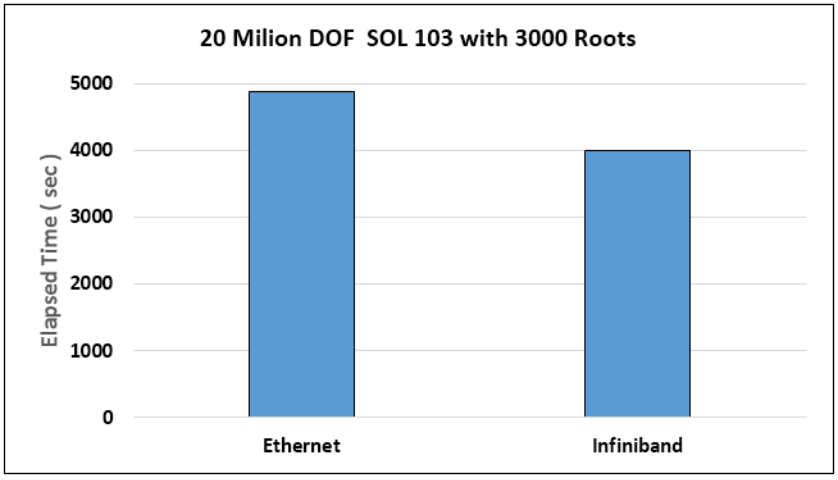


Figure 2-8 Network comparison using DMP across a network

While the above image implies the value of a fast network, it is usually faster to run DMP on the same node unless additional memory of multiple nodes is needed. Also, the amount of communication varies greatly with problem type. The total communication is printed at the bottom of the F04 file. For example:

```
*** MPI STATISTICS FOR DISTRIBUTED NASTRAN ***
```

```
FROM     MESSAGES  TOTAL (MB)  MAX BYTE  AVG BYTE   FROM   MESSAGES  TOTAL (MB)  MAX BYTE  AVG BYTE
---------- --------  ----------  --------  --------  ------  --------  ----------  --------  -----
1 -> 2    87227483   67863.984 100581272       815  2 -> 1      3153     121.999     82040     40572

 TOTAL NUMBER OF MESSAGES SENT =        87230636
 TOTAL AMOUNT OF DATA XFR (MB) =       67985.983
 AVERAGE MESSAGE LENGTH        =             817.
```

Determining Your Current System Description

Linux:

You may use "ibstatus" to see the status of Infiniband:

```
hpccluster <81> /usr/sbin/ibstatus
Infiniband device 'qib0' port 1 status:
        default gid:     fe80:0000:0000:0000:0011:7500:0079:c120
        base lid:        0x1
        sm lid:          0x1
        state:           4: ACTIVE
        phys state:      5: LinkUp
        rate:            40 Gb/sec (4X QDR)
        link_layer:      InfiniBand
```

You may determine your Ethernet status with "ethtool". For example:

```
[root@hpccluster ~]# /sbin/ethtool eth0
Settings for eth0:
        Supported ports: [ TP ]
        Supported link modes:   10baseT/Half 10baseT/Full
                                100baseT/Half 100baseT/Full
                                1000baseT/Full
        Supports auto-negotiation: Yes
        Advertised link modes:  10baseT/Half 10baseT/Full
                                100baseT/Half 100baseT/Full
                                1000baseT/Full
        Advertised pause frame use: No
        Advertised auto-negotiation: Yes
        Speed: 1000Mb/s
        Duplex: Full
        Port: Twisted Pair
        PHYAD: 1
        Transceiver: internal
        Auto-negotiation: on
        MDI-X: Unknown
        Supports Wake-on: d
        Wake-on: d
        Link detected: yes
```

References

1. https://en.wikipedia.org/wiki/CDC_7600
2. http://www.nextplatform.com/2015/03/19/hpc-scales-out-of-balance-with-cpu-heavy-thinking/

3 Optimal Performance of Linear Static Analysis (SOL 101)

- Introduction 38
- Memory Usage 38
- Matrix Solver Options 43
- Parallel Options 49
- Output Request 54
- Additional Topics 58
- Model and Hardware Details 59
- Decision Tree for Linear Static Analysis 60

Introduction

In MSC Nastran, linear static analysis involves the calculation of displacements and other result quantities like stresses in response to applied loads. The performance of the corresponding simulation is dominated by the following two operations:

- Solving the matrix problem
- Writing the requested data to an output file

The inputs to the matrix problem are the stiffness matrix and the loads, and the outputs are the displacements. Stresses and other results are computed based on displacements afterwards. At the end of the simulation, the requested data is written to an output file on the disk.

In Section 3.2, we describe in detail the options for solving the matrix problem, but we mention them here first for references purposes. We consider the two sparse direct symmetric solvers and an iterative symmetric solver:

- MSC Sparse Direct Solver (MSCLDL)
- Pardiso Sparse Direct Solver (PRDLDL)
- CASI Iterative Solver (CASI)

We use a series of models and hardware platforms as described in Section "Model and Hardware Details" on page 59 to illustrate how a user can get best performance based on memory settings, method selection, and parallel settings. We note that in the following sections, unless otherwise stated, we have performed the studies on the HPC Machine described in Section "Model and Hardware Details" on page 59. Finally, in Section "Decision Tree for Linear Static Analysis" on page 60, we provide a decision tree that allows the user to make a choice on the various settings.

Memory Usage

Memory Guidelines

As stated previously, more memory is generally better. We have stated in the previous chapter that users should use 50 to 75% of the physical RAM when specifying memory for MSC Nastran when a single job is being run on the machine; the same applies here to SOL 101.

There are multiple ways that a user can define memory:

```
1. nast20180 mem=40gb
2. nast20180 mem=40gb bpool=20gb
3. nast20180 memorymax=40gb
4. nast20180 memorymax=0.5xPhys
```

In case (1), MSC Nastran gets 40 GB of RAM to perform the simulation with no memory going to Buffer Pool[1]. In case (2), we specifically allocate 20 of the 40 GB to Buffer Pool to cache I/O. Finally, in case (3), we let the estimate program determine how much memory is needed for keeping the solver in-core, then leaving the remainder to Buffer Pool. A larger amount of Buffer Pool will reduce reading from and writing to

[1]Please refer to CHAPTER 2 or the *MSC Nastran 2019 Installation and Operations Guide* for a thorough description of Buffer Pool.

disk, and thus the wall clock time. Finally, in case (4), we set the maximum memory to be 50% of the physical RAM and then let estimate determine the proportions to Open Core and Buffer Pool. Today, we highly recommend the use of (3) or (4) in combination with the default of "mem=max". Note that in the remainder when we refer to "memorymax" that we always assume that the user is employing "mem=max".

The user can query the F04 file to determine the actual amounts given to the various components of the memory system. For a case of "memorymax=16gb", we have the following:

```
        USER OPENCORE (HICORE)          =   588 MB
        EXECUTIVE SYSTEM WORK AREA      =     7 MB
        MASTER(RAM)                     =     2 MB
        SCRATCH(MEM) AREA               =     6 MB (   100 BUFFERS)
        BUFFER POOL AREA (BPOOL4)       = 15779 MB (252434 BUFFERS)

        TOTAL MSC NASTRAN MEMORY LIMIT = 16384 MB
```

Since MSC Nastran only uses 64-bit words, this implies total MSC Nastran memory is 16 GB and the solver (HICORE) gets 0.57 GB and the rest of memory is left for Buffer Pool (BPOOL4) to cache I/O. Scratch mem is not recommended today and gets the default of 100 buffers.

In SOL 101, there are some key requirements with regards to memory to get the best performance. The main requirement is that the direct method or the iterative method fits entirely in-core. The specifics of this requirement are different depending on whether it is the MSC Sparse Direct Solver, the Pardiso Sparse Direct Solver, or the CASI Solver. We describe how to examine memory used in a previous run to see if optimal memory choices were made.

For the MSC Sparse Direct Solver, we can see the details of the memory usage in the F04 file as follows:

```
*** USER INFORMATION MESSAGE 4157 (DFMSYM)
     PARAMETERS FOR PARALLEL SPARSE DECOMPOSITION OF DATA BLOCK KLL     ( TYPE=RSP ) FOLLOW
                      MATRIX SIZE =    685566 ROWS             NUMBER OF NONZEROES =  19553805 TERMS
           NUMBER OF ZERO COLUMNS =         0        NUMBER OF ZERO DIAGONAL TERMS =         0
                      SYSTEM (107) =     32776                      REQUESTED PROC. =         8 CPUS
           ELIMINATION TREE DEPTH =     17772
                CPU TIME ESTIMATE =       650 SEC                I/O TIME ESTIMATE =        10 SEC
       MINIMUM MEMORY REQUIREMENT =       117 MB                  MEMORY AVAILABLE =       588 MB
     MEMORY REQR'D TO AVOID SPILL =       280 MB               MEMORY USED BY BEND  =       117 MB
    EST. INTEGER MEMORY IN FACTOR =       898 MB                 EST. NONZERO TERMS =    239758 K TERMS
     ESTIMATED MAXIMUM FRONT SIZE =      4572 TERMS                    RANK OF UPDATE =        64
 11:32:31     0:13       32.0         0.0        13.4        2.2    SPDC BGN TE=650
 11:32:46     0:28       32.0         0.0       248.1      234.7    SPDC END
 *** USER INFORMATION MESSAGE 6439 (DFMSA)
     ACTUAL MEMORY AND DISK SPACE REQUIREMENTS FOR SPARSE SYM. DECOMPOSITION
        SPARSE DECOMP MEMORY USED =       280 MB                MAXIMUM FRONT SIZE =      4572 TERMS
         INTEGER MEMORY IN FACTOR =        28 MB           NONZERO TERMS IN FACTOR =    239758 K TERMS
   SPARSE DECOMP SUGGESTED MEMORY =       149 MB
```

As can be observed for this case, the entire sparse factor fits in memory. This should always be the target for the default solver and is generally easy to achieve due to the minimal memory usage of the MSC Sparse Direct Solver.

For the Pardiso Sparse Direct Solver, we get the following memory usage details in the F04 file:

```
PARAMETERS FOR INTEL MKL PARDISO PARALLEL SPARSE SOLVE
      MATRIX KLL      on unit 103 ( TYPE=1) FOLLOW
                  MATRIX SIZE =685566 ROWS
          NUMBER OF NONZEROES =19553805 TERMS
        NUMBER OF ZERO COLUMNS =0
               NUMBER OF LOADS =1
                  SYSTEM (107) =1
          MKL REQUESTED PROC. =1 CPUS
```

```
 MEMORY PARAMETERS FOR INTEL MKL PARDISO PARALLEL DECOMPOSITION FOLLOW
                AVAIL. CORE MEMORY =2180 MB
     APPROX. REQUI. IN-CORE MEMORY =2878 MB
 APPROX. REQUI. OUT-OF-CORE MEMORY =1187 MB
 STATISTICS FOR SPARSE DECOMPOSITION USING INTEL MKL PARDISO FOLLOW
   MATRIX NAME KLL      on unit 103
   NUMBER OF NEGATIVE TERMS ON FACTOR DIAGONAL = 0
   MAXIMUM RATIO OF MATRIX DIAGONAL TO FACTOR DIAGONAL = 2036.44 AT ROW NUMBER 550931
```

In this case, the available core memory is much less than the approximate required in-core memory, thus the user will also see the following message in the F06 file

```
*** USER WARNING MESSAGE 4157 (PardisoSolver::checkMemoryNeeds)
     MEMORY PARAMETERS FOR INTEL MKL PARDISO PARALLEL DECOMPOSITION FOLLOW
         AVAIL. WORK MEMORY =2180 MB
         REQUI. IN-CORE MEMORY =2878 MB                REQUI. OUT-OF-CORE MEMORY =1187 MB
```

The user would get better performance by increasing the amount of memory available to the solver by approximately 40 GB such that the available core memory for PRDLDL is greater than 59411 MB. See FAQ #17 in CHAPTER 8 for further details.

For the Pardiso solver with many load cases, we will have an additional information message in the F06 file. Given the following F04 file (corresponding to 1235 loads):

```
PARAMETERS FOR INTEL MKL PARDISO PARALLEL SPARSE SOLVE
      MATRIX KLL      on unit 103 ( TYPE=1) FOLLOW
                  MATRIX SIZE =685566 ROWS
            NUMBER OF NONZEROES =19553805 TERMS
        NUMBER OF ZERO COLUMNS =0
               NUMBER OF LOADS =1235
                  SYSTEM (107) =1
            MKL REQUESTED PROC. =1 CPUS
      MEMORY PARAMETERS FOR INTEL MKL PARDISO PARALLEL DECOMPOSITION FOLLOW
                    AVAIL. CORE MEMORY =5457 MB
          APPROX. REQUI. IN-CORE MEMORY =2878 MB
      APPROX. REQUI. OUT-OF-CORE MEMORY =1187 MB
      STATISTICS FOR SPARSE DECOMPOSITION USING INTEL MKL PARDISO FOLLOW
        MATRIX NAME KLL      on unit 103
        NUMBER OF NEGATIVE TERMS ON FACTOR DIAGONAL = 0
        MAXIMUM RATIO OF MATRIX DIAGONAL TO FACTOR DIAGONAL = 2036.44 AT ROW NUMBER 550931
```

we would have the additional user information message in the corresponding F06 file:

```
*** USER INFORMATION MESSAGE 11363 (PardisoSolver::directSolve)
      INTEL MKL PARDISO SOLVER MULTIPLE LOAD INFORMATION:
      NUMBER OF LOAD CASES   = 1235
      NUMBER OF LOAD CASES FIT IN MEMORY = 150
      NUMBER OF PASSES = 9
      INCREASING MEMORY BY 1219 MB MAY IMPROVE PERFORMANCE BY FITTING MORE LOADS IN MEMORY.
```

This implies that there is only enough memory for 150 loads to be solved at once. Hence, we have to make 9 passes over the load vector to complete the calculations. We will explore the implications of this on performance in the later section where we also discuss the selection of the method based on available memory.

For the CASI iterative solver, the amount of memory needed by CASI can be found in the F06 file with the following system information message:

```
*** SYSTEM INFORMATION MESSAGE 3008 (CAPMEM)MEMORY AVAILABLE TO NASTRAN IS SUFFICIENT FOR CASI
    ITERATIVE SOLVER. TOTAL HIWATER MARK FOR NASTRAN WITH CASI CAN BE CALCULATED BY ADDING THE
    BELOW MAXIMUM ESTIMATED MEMORY NEEDED TO NASTRAN HIWATER
       NASTRAN MEMORY AVAILABLE = 58568MB,
       MINIMUM ESTIMATED MEMORY NEEDED FOR CASI ITERATIVE SOLVER = 5358MB,
       MAXIMUM ESTIMATED MEMORY NEEDED FOR CASI ITERATIVE SOLVER = 8126MB
```

Examples

Now, we will look at a few examples based on the models in Section "Model and Hardware Details" on page 59 that illustrate the effect of memory on performance in SOL 101 for each method. We focus first on MSC

Sparse Direct Solver for Model 3 described in Section 3.6 on the Small Memory Machine described in Section "Model and Hardware Details" on page 59

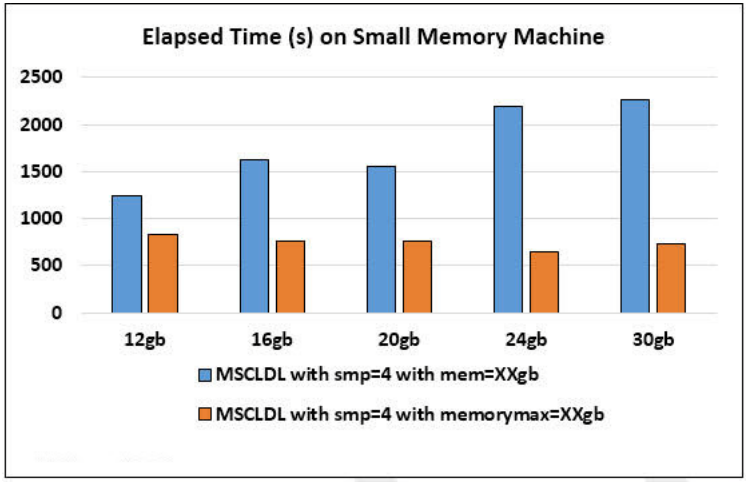


Figure 3-1 Elapsed time on Small Memory Machine for various memory amounts for Model 3

In Figure 3-1, we consider two instances of memory usage with different memory amounts. The case of "mem=XXgb" implies that all memory goes to the solver and no memory goes to Buffer Pool. The case of "memorymax=XXgb" implies that MSC Nastran and the estimate program give optimal memory to the solver and Buffer Pool leading to a significant reduction in I/O and elapsed time.

We can see from Figure 3-1 that it is critical to take advantage of Buffer Pool to cache I/O and that the use of memorymax is an efficient way to do this for MSC Nastran. We also note that I/O is reduced as we go from the left at "memorymax=12gb" to the right at "memorymax=24gb" leading to a reduction of 80 GB of I/O to 27 GB of I/O.

We see that "memorymax=24gb" is the fastest option, which corresponds to 75% of physical RAM. We also see that at "memorymax=30gb" we are slowing down as compared to "memorymax=24gb". This corresponds to our recommendation to not increase memory amounts beyond 75% of physical RAM. We did not include results for the Pardiso Solver for this case, since based on the memory estimates described previously it was not appropriate to use Pardiso. In fact, during testing, Pardiso only ran for "mem=26gb" but no memory was given to Buffer Pool and I/O costs caused the simulation to be > 2x longer than MSCLDL which was able to effectively use Buffer Pool.

In the next example, we consider the Pardiso Solver for Model 2 on the HPC Machine, where there is plenty of memory, and we force Pardiso to run out-of-core and compare performance to the MSC Sparse Direct Solver for the same memory amount.

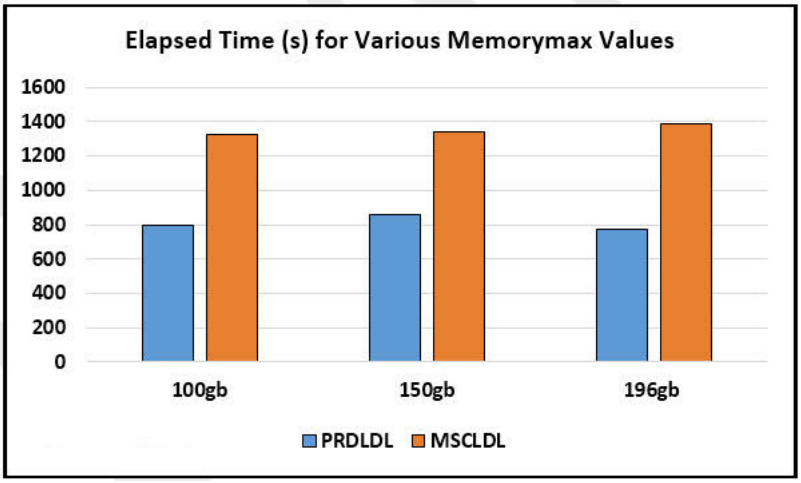


Figure 3-2 Elapsed time for various memorymax amounts using PRDLDL or MSCLDL for Model 2 on the HPC Machine with smp=16 and setting bpool=50gb

In Figure 3-2, for "memorymax=196gb", we have more memory available to Open Core implying that PRDLDL gets more memory and runs entirely in-core. Hence, it has the optimal elapsed time. For "memorymax=150gb" and "memorymax=100gb", PRDLDL is running out-of-core but it is still faster than MSCLDL. Note that we are directly specifying the amount of memory given to Buffer Pool instead of allowing the estimate program to specify. This is to observe the performance of PRDLDL as it has different memory amounts available for keeping the factorization in-core. As stated previously, we recommend however for the user to merely set memorymax and let MSC Nastran divide up memory to the solver and Buffer Pool optimally.

In the above discussion, we have focused on the performance of the two Sparse Direct Solvers with regards to memory but have not mentioned the iterative solver. The iterative CASI solver is less dependent on memory and generally works in most memory contexts, and therefore, we will not explore its dependency on memory here.

Matrix Solver Options

The method selection involves specifying the approach to solving the matrix problem that returns the displacements given the applied loads. In linear algebra terminology, we want to select the method that finds "x" in the linear algebra equation "$Ax=b$". There are two types of methods for solving $Ax=b$: direct methods and iterative methods.

Direct Methods

Direct methods are the default in MSC Nastran and require factorizing the stiffness matrix A and then performing forward and backward solves, often referred to as the FBS part of the solution algorithm. The factorization generally takes 80-90% of the total solve time and the FBS generally takes the remaining 10-20% of the total solve time.

There are two main direct methods in MSC Nastran for SOL 101: the MSC Sparse Direct Solver (MSCLDL) and the Pardiso solver (PRDLDL). The MSC Sparse Direct Solver is the original sparse direct solver that has been in MSC Nastran for over 20 years. It can run in very limited memory settings but has limited parallel scalability. The Pardiso solver was added in MSC Nastran 2016.0 for SOL 101. It consumes 5-12x as much memory as MSCLDL depending on the model type, but it can exhibit much greater performance with shared memory parallelism, i.e., SMP. See CHAPTER 8 and FAQ #19 for a description of the memory usage difference between MSCLDL and PRDLDL.

No changes have to be made to the input file to choose the MSC sparse direct solver since it is the default option. To use the Pardiso solver, the user needs to add the following to the Executive Control Section (as described in the *MSC Nastran Quick Reference Guide*):

```
SOL 101
SPARSESOLVER DCMP(FACTMETH=PRDLDL)
CEND
```

If the model has linear contact, then the user needs to add the following since linear contact depends on the NLSOLV module:

```
SOL 101
SPARSESOLVER NLSOLV(FACTMETH=PRDLDL)
CEND
```

Other options with respect to the Pardiso solver can be found in the MSC Nastran Quick Reference Guide.

Recommendations for use of MSCLDL and PRDLDL:

1. If the number of loads is greater than 64 and there is sufficient memory, then use PRDLDL regardless of recommendation #2.
2. If number of solid elements is less than 80% of total number of elements, then use one of the two Sparse Direct Solvers, else use iterative methods (described next).
3. If the above is true and memory is sufficient to use PRDLDL, then use it; otherwise, use MSCLDL. To determine if memory is sufficient for PRDLDL, we provide the rough guide based on model details.

To determine if memory is sufficient for PRDLDL, we provide the rough guide based on model details. We break up the types of models into shell-element models and solid-element models described in Table 3-1.

Table 3-1 Breakdown of Model Type based on Types of Elements.

Model Type Name	Model Description
Shell-Element models	Models with ≤ 80% Solid Elements
Solid-Element models	Models with > 80% Solid Elements

Next, we discuss memory requirements for models of these types using PRDLDL.

For shell-element models, we have tested PRDLDL on a wide variety of models of sizes ranging from 500K DOFs to 12M DOFs. We investigate the memory requirement for PRDLDL to run in-core and then perform analysis to estimate the memory requirements for various problem sizes. This data is seen in Table 3-2, where we provide a suggested memorymax and buffer pool amount for various model sizes.

Table 3-2 Estimates of Memorymax for shell-element models using the Pardiso Solver.

DOF=6*ngrids	Suggested memorymax	Suggested bpool
< 1 Million	8 GB	20x
1-2 Million	16 GB	20x
2-4 Million	24 GB	20x
4-8 Million	48 GB	20x
8-12 Million	64 GB	20x
12-16 Million	96 GB	20x

Note that these are rough estimates, and may at times overestimate or underestimate, but it is a good initial guide. We will continuously improve on these estimates and build them into the solver in future releases. The user will want to run using these estimates in the following way:

```
nast20180 shellmodel.dat memorymax=XXgb bpool=20x
```

We have performed the same analysis for solid-element models using a wider variety of model types. The data is in Table 3-3, where we provide suggestions for memorymax and buffer pool.

Table 3-3 Estimates of Memorymax for Solid-Element Models using the Pardiso Solver.

DOF=3*ngrids	Suggested memorymax	Suggested bpool
< 1 Million	16 GB	20x
1-2 Million	32 GB	20x
2-4 Million	64 GB	20x
4-8 Million	96 GB	20x
8-12 Million	112 GB	20x
12-16 Million	148 GB	20x

Note that in neither of these cases (shell-element or solid-element) have we considered the case of multiple loads. If there are greater than 64 loads, then additional memory needs to be added to memorymax to account for the additional loads. It is best to add the following to memorymax:

additional memory = nloads*ndof*8

Where nloads is optimally min (num_loads,256) but should be adjusted based on memory of hardware. Also, note that this value is computed in bytes, so it must be converted to MB or GB to add to given memorymax.

Based on the value of memorymax for shell-element models and solid-element models, we have the following decision table Table 3-4:

Table 3-4 Decision table on when to use the Pardiso Solver in SOL 101.

Memory Comparison	Solver Choice
memorymax ≤ 0.75*RAM	Use PRDLDL
memorymax > 0.75*RAM	Use MSCLDL

where RAM is the physical memory available on the machine. This does not need to be followed rigorously but is a good first start to determining if your model can run in-core and fast enough to use the Pardiso Solver.ol

Note that similar memory estimates are used in MSC Nastran in SOL 101 when a user employs the Pardiso Solver in SOL 101. However, the above estimate, can be used by a user a priori to decide whether or not to use Pardiso Solver and to more carefully control the memory amounts. Note that the MSC Nastran HPC team is constantly working on improving these estimates.

Iterative Methods

Iterative methods are the other option to solve the linear equations. They must be selected by the user via a Case Control statement. An iterative method does not rely solely on a factorization and an FBS, but instead performs several iterations, $\{x_1, x_2, x_3, \ldots, x_n\}$, where x_n is a converged approximation to the exact answer. Iterative methods are generally based on the Conjugate Gradient method or the GMRES method. These methods can be much faster and consume much less memory for certain problems, mainly solid-element models.

While there are a few iterative method options in MSC Nastran, we focus only on one: the CASI Solver. The CASI solver is selected by adding the following in the Case Control section:

```
SMETHOD=element
```

See the *MSC Nastran Quick Reference Guide* as well. Note that the CASI solver is a Conjugate Gradient solver that uses a preconditioner based on element matrices. As stated in the previous section, we have the following recommendations for using the CASI Solver:

- If the number of solid elements is greater than 80% of total number of elements, then use the CASI iterative method unless there are a large number of loads (> 64 loads), for which one should use PRDLDL.

Examples

The first example that we consider in Model 2, is a solid-element engine-block model where the CASI Solver is most effective. On the HPC Machine, we can completely eliminate I/O due to large amounts of memory given to Buffer Pool so we strictly focus on the performance differences of the various methods. The results of the CASI solver as compared to the MSC Sparse Direct Solver (MSCLDL) and the Pardiso solver (PRDLDL) are in Figure 3-3 for SMP=1-16.

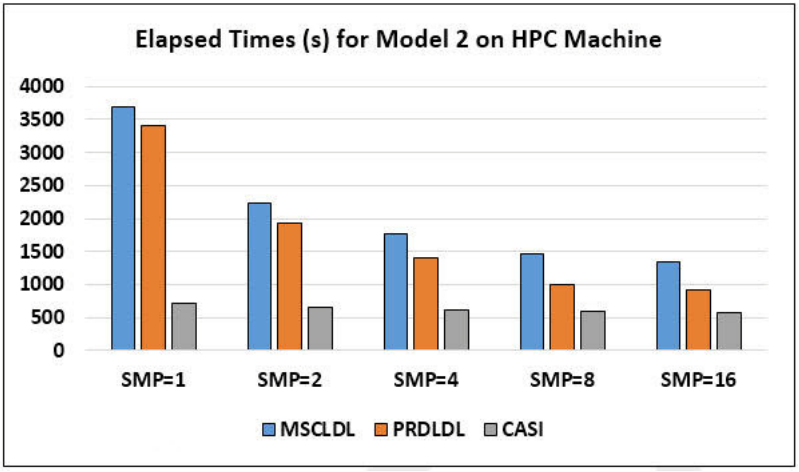


Figure 3-3 Elapsed times using MSCLDL, PRDLDL, and CASI for Model 2, which is a solid-element engine-block model, for various SMP values.

The results in Figure 3-3 illustrate the significant performance advantage of CASI over both the MSC Sparse Direct Solver and the Pardiso Solver. CASI for a single thread (SMP=1) is faster than both the MSC solver and Pardiso for 16 threads (SMP=16). CASI with SMP=16 was 20% faster than CASI with SMP=1, or in this context, was 2 minutes faster on an 11-minute simulation.

The second example is Model 3, which is a shell-element model. We again use the same three solvers but see different results in Figure 3-4 below for the various solvers.

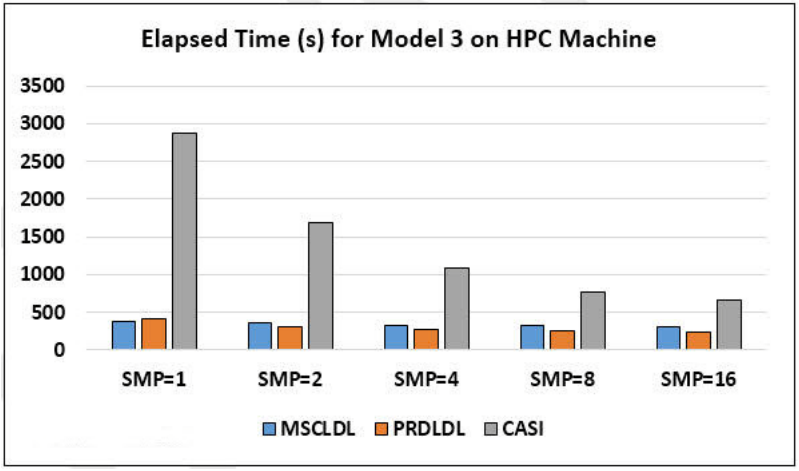


Figure 3-4 Elapsed times using MSCLDL, PRDLDL, and CASI solver for Model 3, which is a shell-element model, for various SMP values.

The results illustrate that CASI is not an effective method for a model dominated by shell elements. CASI does show good scalability due to the fact that it requires many iterations to find a solution. The iterations in CASI (3776 in this case) are where the SMP parallelization is employed so we expect to see this behavior.

The third example is for Model 4, which is a solid-element model with linear contact. The problem is nonlinear and requires 10 iterations to converge. Furthermore, for this nonlinear problem, we have a different default convergence criterion for the iterative method. In SOL 101 linear statics, the problem is converged when its error tolerance is 1e-4, whereas for SOL 101 linear contact, the problem is converged when its error tolerance is 1e-8. Refer to the *MSC Nastran Quick Reference Guide* for details on changing the tolerance.

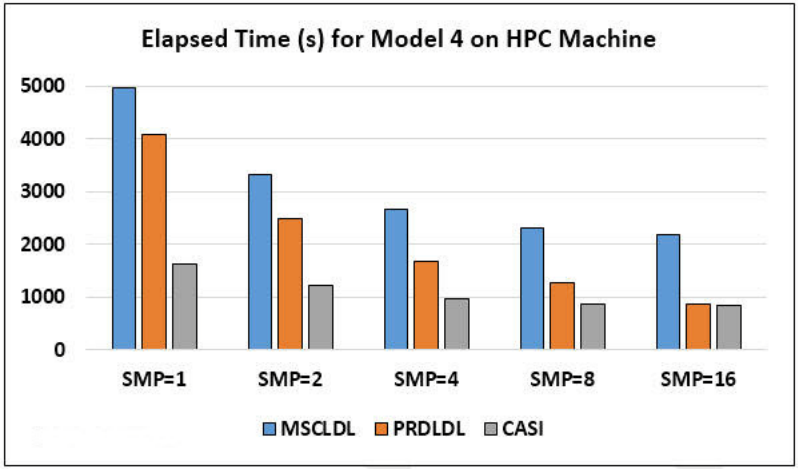


Figure 3-5 Elapsed times using MSCLDL, PRDLDL, and CASI for Model 4 for various SMP values.

We see in Figure 3-3 that CASI is still much faster than MSCLDL and PRDLDL except for SMP=16 where the Pardiso solver has equivalent performance. In the case of linear contact, the problem is different than the typical linear statics model of SOL 101. The contact is handled by Lagrange multipliers implying the matrix solve in each iterations is highly indefinite. The CASI solver solves this problem by converting the Lagrange multiplier representation to multi-point constraints. This implies the resulting matrix will be positive definite.

For the MSC Solver and Pardiso, the matrix is not converted and instead the highly indefinite problem is solved. The performance differences between the MSC Solver and Pardiso for Model 5 are due to the fact that they are solving a highly indefinite matrix due to the existence of contact. The Pardiso solver is much more effective at solving highly indefinite problems as compared to the MSC Sparse Direct Solver.

The fourth example is Model 1, which is a typical wing model from the aerospace industry with 1234 load cases. We only test the default MSC Solver and the Pardiso solver. We described in Section "Memory Usage" on page 38, and saw in the previous example, that the CASI solver is not suitable for models with lots of 2D elements. It is also not suitable for models with a large number of loads, and thus we do not test it here. The tests for this example are performed on the HPC Machine and are presented in Figure 3-6.

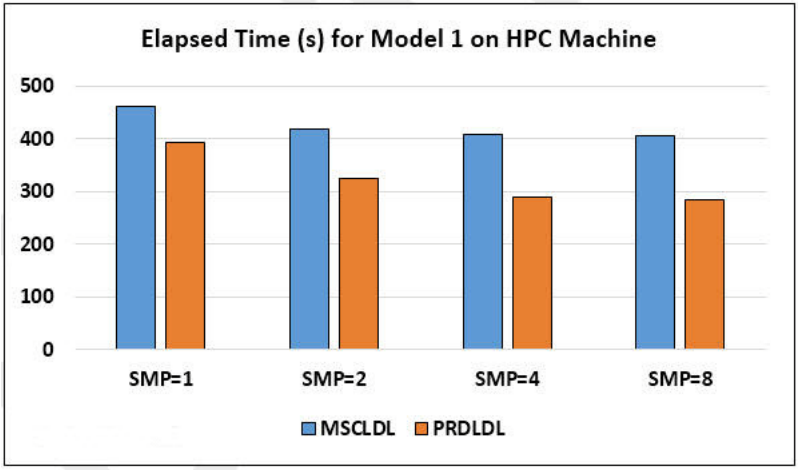


Figure 3-6 Elapsed time using MSCLDL and PRDLDL for Model 1 for various SMP values

The results in Figure 3-6 illustrate that the Pardiso Solver is capable of greater performance at higher SMP values due to its ability to parallelize over a large number of load cases. Thus, for the case of a larger number of loads with sufficient memory, PRDLDL is the best option when there is sufficient memory for Pardiso to run in-core and for enough memory to be left for Buffer Pool to be useful.

Note that in the above case for PRDLDL, we were able to fit 195 loads into memory at once for serial; 145 loads into memory at once for SMP=2; 95 loads into memory for SMP=4; and 55 loads into memory for SMP=8. The solver had 8 GB of memory available. The aim is to fit approximately 256 loads into memory at a time for optimal performance.

We have covered several cases and have described the proper time to use the various solvers. In addition to the information presented on selecting options based on memory amount, we have a good idea now based on how much memory is available and what type of model is it in order to choose the correct method. Next, we move to improving the performance of the selected method by utilizing parallel options.

Parallel Options

There are three parallel options available to users: shared memory parallelism (SMP), distributed memory parallelism (DMP), and general purpose graphics processing units (GPGPU/GPU). We describe the benefit of each, and later provide the suggestions to the users for each of these options. The conclusions will be that users will get the most benefit out of just using SMP.

In MSC Nastran, independent of method selection, a SOL 101 job will likely encounter an MPYAD call (performs general purpose matrix multiplication) and this will automatically get benefit from SMP. In future releases, this benefit will be even greater than today.

For MSCLDL, SMP is used to accelerate the mathematical kernels (like DGEMM) to reduce the time to perform the factorization. When the matrices are solid-element models, then the DGEMM operations are more significant and thus SMP can be more beneficial. For a model with over 5M DOFs and strictly solid elements, then the simulation will receive benefit up to SMP=8, but not likely beyond. If the same sized

model has mostly shell elements, then benefit may only go up to SMP=4 or 6. For smaller models, e.g., less than 2M DOFs, the benefit for solid or shell element models will likely only be up to SMP=4. The critical issue is the models have to be large enough for multiple process to have work to do.

For PRDLDL, SMP is used to accelerate not just the mathematical kernels but also to divide up the work on a much larger scale. Thus, to the user, this means one can expect greater parallel performance for a given SMP value as well as greater benefit out of using higher SMP values. Just as with MSCLDL, the greatest benefit is for the larger models and for those with solid elements. For example, a user has reported on good parallel efficiency to SMP=32 for a solid-element model with 30M DOFs.

For CASI, the details of the SMP implementation are unknown to MSC Software as it is a proprietary solver. Users, as well as the HPC team, have not seen the benefit of higher SMP values, as with PRDLDL, but still have observed very good scaling up to SMP=4 for larger models. We have seen a 2x reduction in the time to solve the matrix problem with the iterative method using SMP=4. Note that better scalability is observed for CASI at higher SMP values if the model requires many iterations due to ill-conditioning but generally then CASI is not the appropriate solver.

A general set of guidelines of SMP usage for the various solvers is found below. First, we defined what a small, medium, large job and very large job is and then we define the best SMP options for these cases.

Table 3-5 Definition of Problem Size dependent on Model Type and Number of DOFs.

Problem Size	Model Types	# DOFs
Small	Solid-Element Models	#DOFs < 2M
	All Other Model Types	
Medium	Solid-Element Models	2M < #DOFs < 4M
	All Other Model Types	2M < #DOFs < 6M
Large	Solid-Element Models	4M < #DOFs < 20M
	All Other Model Types	6M < #DOFs < 30M
Very Large	Solid-Element Models	20M < #DOFs
	All Other Model Types	30M < #DOFs

Table 3-6 Recommendation for SMP values for different problem sizes and method types.

Method Type	Small	Medium	Large	Very Large
MSCLDL	SMP=1-2	SMP=3-5	SMP=6-8	SMP=8-16
PRDLDL	SMP=1-4	SMP=5-8	SMP=9-16	SMP=17-32
CASI	SMP=1	SMP=1-2	SMP=3-6	SMP=6-8

For DMP, the only method that receives benefit is MSCLDL, and the benefit is marginal as compared to just SMP. In the examples below, we show a case where there is no benefit in using DMP=2 with SMP=4 (a total of 8 CPU cores where 4 CPU cores are used in separate DMP processes) as compared to just SMP=8 (a total of 8 CPU cores used strictly for SMP). Thus, with current hardware and usage, DMP is not recommended in SOL 101. This may change in the future as we continue to investigate new technologies.

For GPU, we show in the examples below that the only benefit is if it is used with SMP=1 on newer hardware. Furthermore, the use of GPU should be reserved for solid-element models. Once you increase the value of SMP, it is better to just let SMP alone do the work; this is even more the case with today's CPUs than with older CPUs. With older CPU hardware like Intel Sandy Bridge architecture or Westmere architecture, there can be a greater benefit with using a combination of SMP and GPU.

Examples

In the first example, we compare the use of DMP and GPU. We again run Model 2 on the same HPC Machine but we now test for DMP=2 and then for gpuid=0, i.e., they are not combined. The gpuid option uses the GPGPU attached to the HPC machine to accelerate the MSCLDL solver. The GPGPU on the HPC Machine is a Tesla K40m with 12 GB GDDR5 memory. The results are in Figure 3-7 comparing standard SMP with the DMP option and the GPU option.

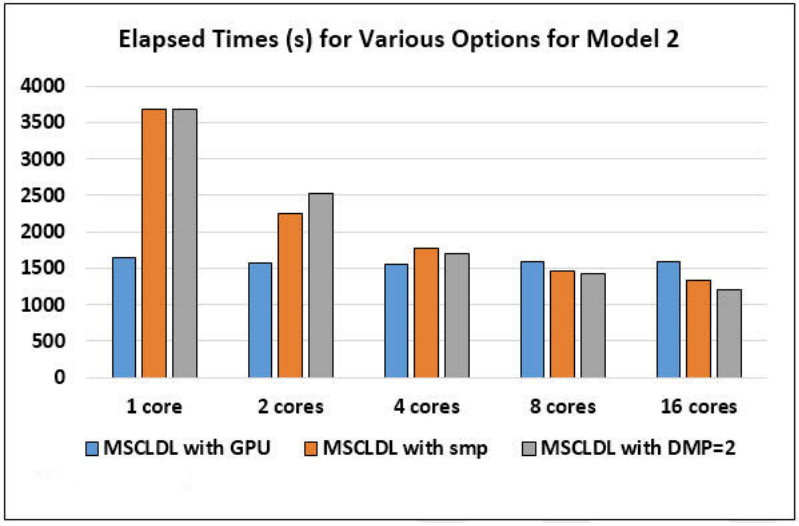


Figure 3-7 Elapsed time of various options with MSCLDL using DMP or GPUs for Model 2 for performance acceleration

In these results, we have specified the number of CPU cores used instead of the SMP value. This is because for DMP=2 with SMP=N we are actually using 2*N cores. Thus, for MSCLDL with DMP, the actual smp value for given number of cores is SMP=(#cores)/2. However, for MSCLDL with SMP and MSCLDL with GPU, it just implies that SMP=#cores. As we can see from these results, for higher core count, there is little benefit to using "dmp=2" or "gpuid=0". These two are most beneficial when a small number of cores are used.

The results presented in Figure 3-7 are for a machine with an Intel Haswell CPU, which has the newest AVX2 technology and thus shows less benefit to offloading computation to the GPU. Next, we consider only the GPU results on a different machine with an older CPU, which only has AVX technology. For the Medium Memory Machine, we get the following set of results as shown in Figure 3-8. As observed in Figure 3-8, we have a greater speed-up when using the GPU with serial (2.61x speed-up verses 2.25x speed-up). This is again due

to the AVX2 technology in the Intel Haswell CPU (found in the HPC Machine) that gives better serial performance.

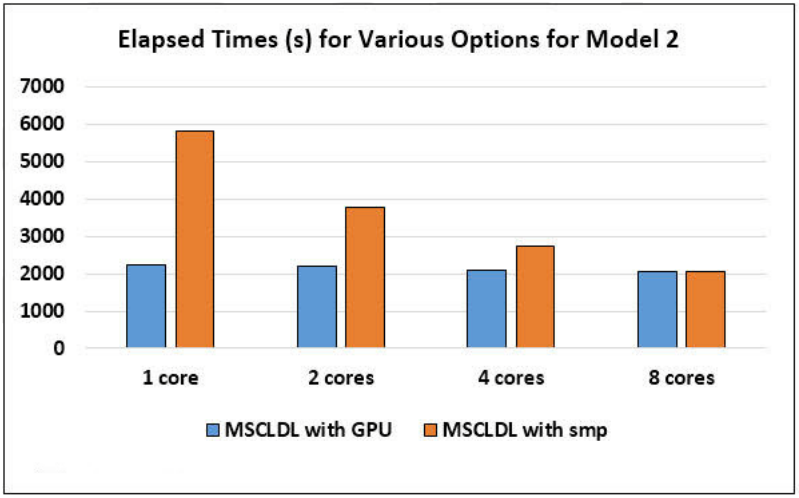


Figure 3-8 Elapsed time of various options with MSCLDL using SMP or GPUs for Model 2 on the Medium Memory Machine with Intel Sandy Bridge processors

In summary, the GPU is most useful when a user only has a few cores available or wants to reduce token cost and can get good acceleration by just turning on the GPU. Today, we do not recommend the use of DMP for performance acceleration of SOL 101 in MSC Nastran 2018.0 or before.

In the second example, we consider the CASI solver and the impact of using SMP. We again refer to results from Model 4, which is a SOL 101 model with linear contact for which we showed in Section "Matrix Solver Options" on page 43 that CASI was a very effective solver. We investigate it here again in more detail.

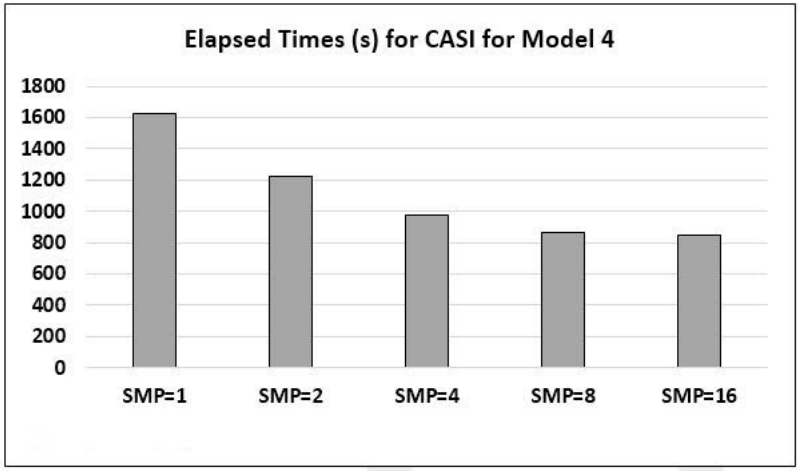


Figure 3-9 Elapsed time for CASI Solver for Model 4 for various SMP values

As we can see, the scalability is very good up to SMP=4 but beyond that we get less benefit out of SMP=8 or SMP=16. If we look at the various MSC Nastran modules and their impact on the overall time, we can see that just accelerating the CASI Solver yields this type of behavior. Thus, we investigate results in Figure 3-10 but split up the times into SETUP, TRANSF, CASI, and SDR2.

The time spent in SETUP is everything before the nonlinear iterations begin. The time spent in TRANSF is the time spent in converting the matrix to a form suitable for CASI. The time spent in CASI is in the matrix solve and the time spent in SDR2 is computing the stresses. This job had no output.

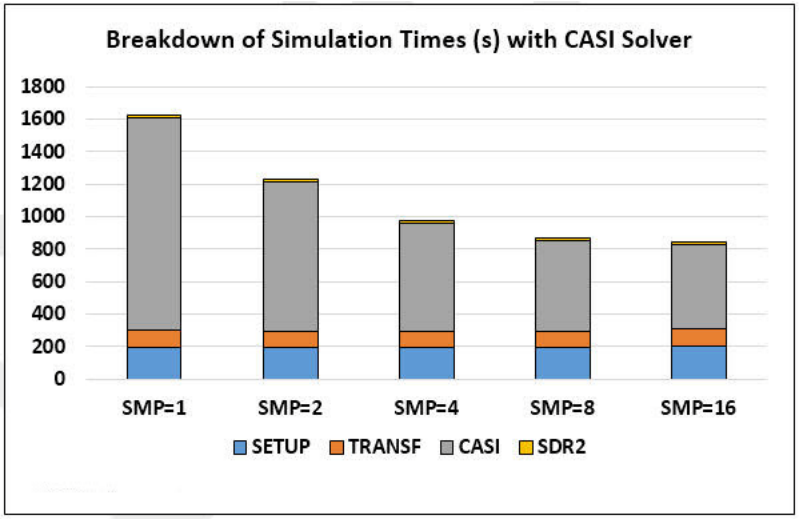


Figure 3-10 Breakdown of time for CASI Solver for Model 4 into SETUP, TRANSF, CASI, and SDR2 for various SMP values

The results in Figure 3-10 show that approximately 300 s receives no benefit from SMP and that only CASI in this case sees reduction as SMP is increases. It also further illustrates that the benefit of SMP to CASI stops at SMP=4 for this model. Note that this model has 4.9M DOFs and is mainly solid elements. We mentioned before that the benefit of SMP for CASI is found in the iterations, thus the more iterations the greater benefit. For this model, it only took 312 iterations regardless of SMP value. Note that the overall scalability would be less if we had a large output request, which we examine in the next section.

In the third example, we consider Model 1 with many load cases and perform a similar analysis to Model 4 above. We compare MSCLDL and PRDLDL and examine where the impact of SMP is for these two solvers.

For this model, we use SETUP, FACTOR, FBS, and OUTPUT corresponding again to the setup of the problem, the matrix factorization phase, the forward-backward solve phase, and the output time.

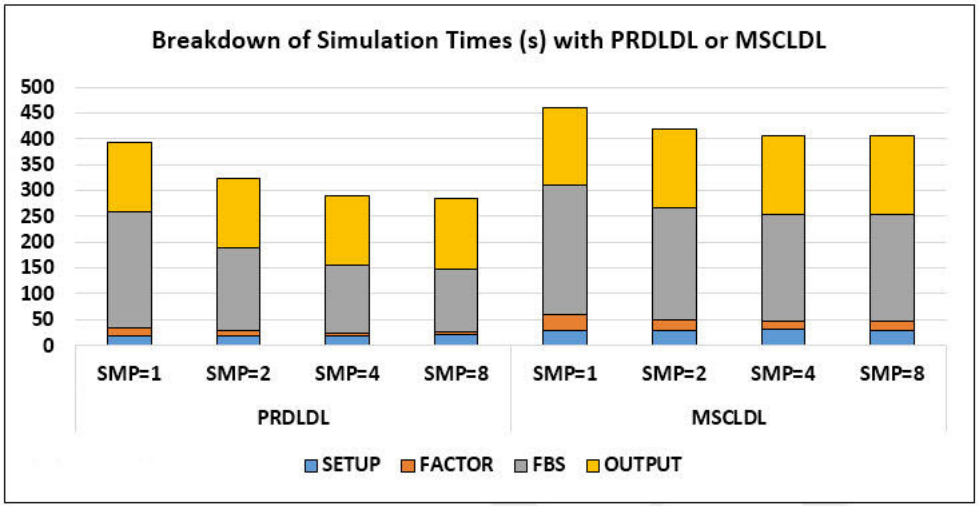


Figure 3-11 Breakdown of time for MSCLDL and PRDLDL for Model 1 into SETUP, FACTOR, FBS, and OUTPUT for various SMP values

The results in Figure 3-11 illustrates the better scalability of PRDLDL for the factorization and the forward-backward solve. Note that the FBS time is large for this model, as compared to other models, because there are over 1000 load cases. These results also illustrate that the output is a significant part of the overall cost and reduces the overall scalability of the various solvers. Next, we consider that output request and how the various choices impact overall time and overall output file size.

Output Request

Solving a linear statics problem invariably requires some output to an OP2 file, an HDF5 file, or a PCH file. For an HDF5 file, we can write uncompressed data (HDF5,0) or compressed data (HDF5,1). The amount of output in the output request can have a dramatic effect on the wall clock time, which is due to the time spent in recovering the results and the time spent in writing the data to the results file. A small output request such as

```
SET 1 = 1 THRU 100 $ limited output set
DISP = 1
STRESS = 1
```

will have a minimal impact on the wall clock time, but a large output request such as

```
DISP(PLOT) = ALL
STRESS(PLOT) = ALL
```

with

```
PARAM,POST,-1
```

or

```
MDLPRM,HDF5,0
```

can have a noticeable impact on the wall clock time. The choice of whether to use an OP2 or HDF5 file will have a minimal impact on the wall clock time.

Note that a punch file can be generated with

```
DISP(PUNCH,PLOT) = ALL
STRESS(PUNCH,PLOT) = ALL
```

but the use of a PCH file can have a very large impact on performance, as we will see below, and it will generate a very large output file. Finally, the specification of "PLOT" means that the output quantities are not printed in the F06 file. This will save both time and disk space in the F06 file and should be always used for large output requests.

To get the best performance for the case of a large output request (either use of ALL or a set with a very large number of grid point or element ids), it is imperative that the user following some basic standards:

1. The user should avoid directing output to a PCH file, if at all possible.
2. The user should employ the "PLOT" option to eliminate writing to the F06 file.
3. The user should direct output to a fast local disk such as an SSD or a fast spinning HHD.
4. The user should not use more than 75% of physical memory for the simulation to leave some memory to the operating system to buffer I/O.

Example

In this example, we first consider Model 2, which has an output request given by STRESS(PLOT)=ALL. We examine how different output types affect performance. We run the model on the HPC Machine, but we vary the output request. We consider no output at all as well as an output in OP2 format (PARAM, POST,

-1), an output in HDF5 format (MDLPRM, HDF5, 0), and output to a PCH file. In all results, we use SMP=8.

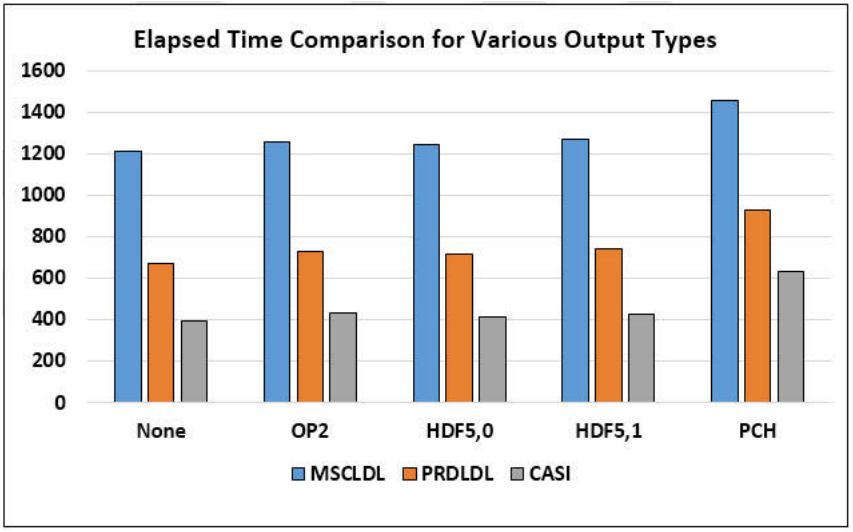


Figure 3-12 Elapsed time for the three solvers with various output requests with SMP=8 for Model 2

The results show that the output request is a 5-10% cost for STRESS=ALL depending on the type of solver. The results clearly show the cost to writing a PCH file and thus we highly recommend against using PCH files. The results also show that the HDF5 format is slightly faster than OP2 for the case of HDF5 with no compression. The size of the various output files is given in Figure 3-13 below.

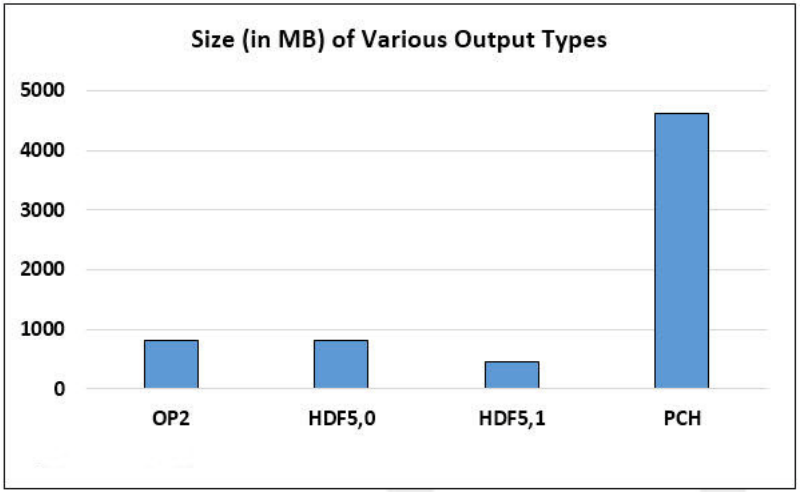


Figure 3-13 Sizes (in MB) of the various output types

Next, for Model 3, we consider the effect of various output types on an even larger output request given by output DISPLACEMENT, ESE, MPCFORCES, SPCFORCES, and STRESS. We compare OP2 files,

HDF5 files with no compression, and HDF5 files with compression. The results are as given in the following table where we have left out PCH results due to our earlier recommendation to avoid using.

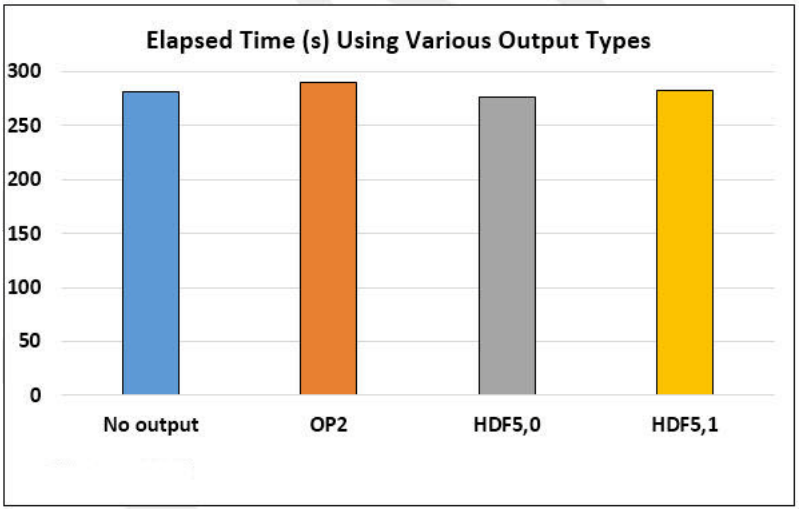


Figure 3-14 Performance differences for the three output types using PRDLDL with SMP=8 for Model 3

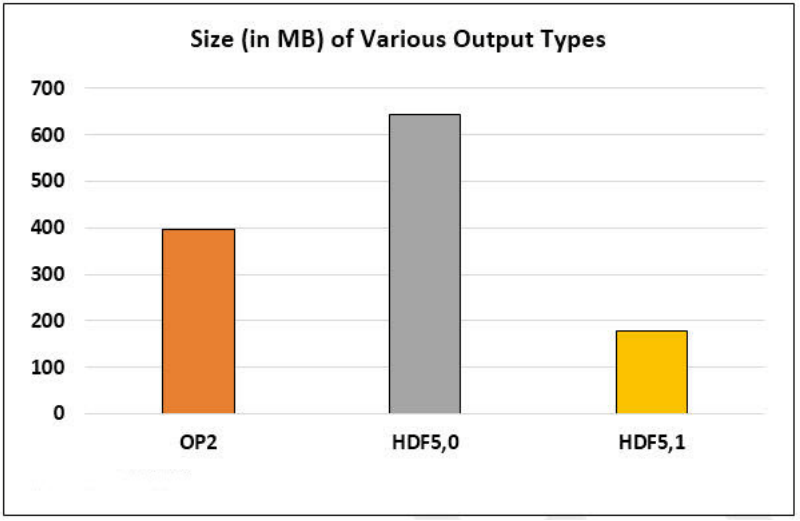


Figure 3-15 Sizes (in MB) of the various output types for Model 3

We can see here that the sizes vary by 2-4x. We note that HDF5 with compression is particularly effective for this case since computed stress contains a number of zeros allowing for greater compression. We can also see from the performance numbers in Figure 3-14, a user would gain from reduced output size at no real cost by using HDF5 with compression. The results in Figure 3-14 show very little difference in cost for using no output, OP2, or the HDF5 file, but these results were generated on the HPC machine with very fast disks.

A machine with slower disks would of course have a higher overhead over writing to an OP2 file or an HDF5 file.

Finally, we note that the amount of data written to the output file, even for an “ALL” event, is relatively small compared to writing all 500 eigenvectors to disk for the case of Normal Modes Analysis. Thus, we refer the reader to the next chapter to observe a different set of behaviors for larger output files.

Additional Topics

High Matrix Connectivity

High matrix connectivity is a feature of the matrix where there are regions of the matrix with lots of nonzero values connect many degrees of freedom together. These could be fully dense matrix sub-blocks in the matrix or just very non-sparse regions. This usually occurs with special types of RBE2 or RBE3s or with large Direct Matrix Input at Grid (DMIGs) from a previous superelement reduction. These can be especially challenging for sparse matrix solvers like MSCLDL or PRDLDL. Also, for the CASI solver, we have the following limitation from the *MSC Nastran Quick Reference Guide*:

- x2GG/x2PP direct input matrix selection is allowed; however, the matrix size is limited to 100 grid points and must be symmetric.

The recommendation is to avoid this if possible and find alternative modeling approach. However, in many cases, this is not possible and so we make the following recommendation:

- If memory is sufficient, we recommend to use the Pardiso Solver, PRDLDL.
- If memory is not sufficient, then use the default MSC Sparse Direct Solver, MSCLDL.

Inertia Relief

Inertial relief in SOL 101 is used for analysis of unsupported systems such as vehicles in flight. It is a way to perform the analysis without grounding a body. The process of performing inertia relief analysis leads to a matrix problem, which is slightly indefinite. Thus, to use the Pardiso solver with inertia relief, the user should add the following to the bulk data section:

```
MDLPRM,PRDMTYPE,-2
```

This forces the Pardiso Solver to treat the matrix as indefinite and thus properly handle the pivoting needed for the inertial relief problem.

Model and Hardware Details

The following models are used in this section to illustrate our points about obtaining best performance in a linear statics run.

Table 3-7 Summary of Model Types for Performance Comparison

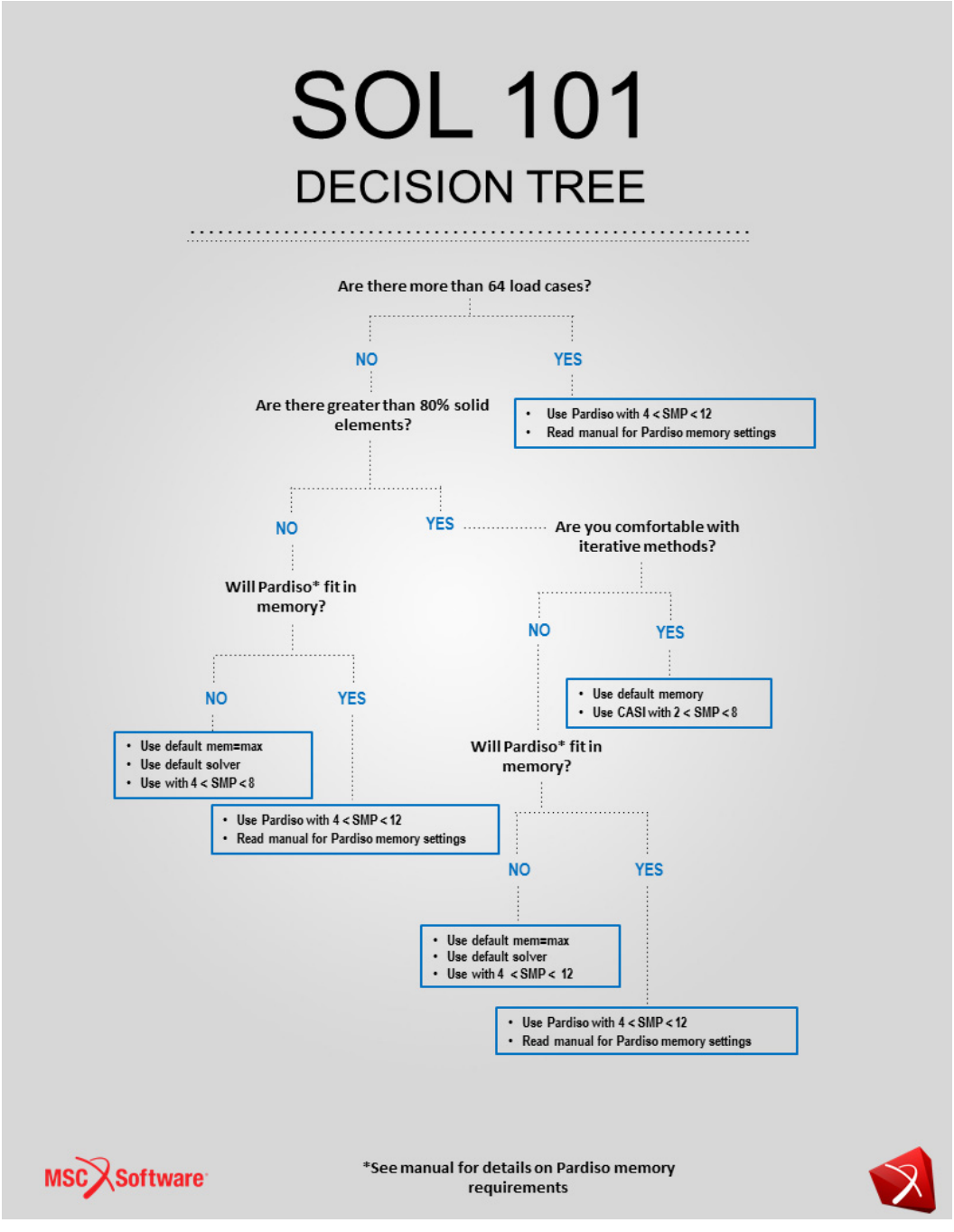
Model	Description	DOFs	2D Elements	3D Elements	Force
1	Aerospace model with many loads	688200	111099	0	1234
2	Large Solid Element Engine Block	14739330	0	1551620	1
3	Large Shell Element Body-In-White Model	7318782	1273695	193134	1
4	Solid Element Model with Linear Contact	4906336	0	525741	1

These machines are a large HPC machine, a Medium Memory machine, and a Small Memory. Details follow:

Table 3-8 Summary of Machine Types for Performance Comparison

Memory	Memory	CPUs	HDD	OS
HPC Machine	256 Gb	2x10 Intel® Xeon® CPU E5-2550 v3 @2.60GHz	4 x 600 GB 15K RPM in RAID0	RH 7.1
Medium Memory Machine	128 Gb	2x8 Intel(R) Xeon(R) CPU E5-2650 v2 @2.60GHz	1 x 550 GB 10K RPM Hard Drive	SuSE SP2
Small Memory Machine	32 Gb	2x10 Intel® CPU @2.8GHz	1 x 1 TB 7200 RPM Hard Drive	RH 6.3

Decision Tree for Linear Static Analysis



Main Index

4 Optimal Performance of Normal Mode Analysis (SOL 103)

- Introduction 62
- Memory Usage 62
- EigenSolver Options 66
- Parallel Options 69
- Output Request 72
- Additional Topics 73
- Models and Hardware Details 75
- Decision Tree for Normal Modes Analysis 76

Main Index

Introduction

In MSC Nastran, normal modes analysis is performed in the following scenarios:

- To determine the lowest elastic vibration modes
- To check the magnitude of the “rigid body” (zero frequency) vibration modes
- To compute a modal basis for a subsequent dynamic analysis

To satisfy requirements for the first two scenarios, the recommend solution strategy is to employ the block-shifted Lanczos eigenvalue extraction method. Assuming the requirement is to extract approximately the lowest 20 eigenvalues, the solution characteristics and performance issues are the same as for linear static analysis (Chapter 3). The focus of this section is on eigenvalue extraction techniques of real matrices (excluding damping). For guidelines on solving the damped eigenvalue problem, refer to the “FAQ” on page 145. A specific chapter will be devoted to this problem in a later release of this manual.

The other scenario above, computing a modal basis for dynamic analysis, is more optimally accomplished by using the MSC Nastran Automated Component Modal Synthesis technique (ACMS). ACMS is a multi-level substructure, modal reduction technique that produces a close approximation to a normal modes solution. It is optimal for analyses requiring a significant number of vibration modes, and also for jobs requiring relatively few vibration modes of very large models.

For any of the three possible scenarios, the best performance is obtained by a combination of method selection, memory usage, parallel settings, and hardware. Furthermore, the model features can have a significant impact on the overall performance. In this chapter, we discuss topics ranging from the effects of method selection, memory settings, and parallel options, output requests and hardware configurations on performance.

Section “Models and Hardware Details” on page 75, describes the models and hardware platforms that are used throughout to show performance. Unless otherwise stated, the default machine is the HPC Machine described in Section “Models and Hardware Details” on page 75.

Refer to the Section “Decision Tree for Normal Modes Analysis” on page 76 which allows the user to make a decision on method selection, memory settings, and parallel options. The details in Sections “Memory Usage” on page 62 through “Additional Topics” on page 73 support the decision tree in Section “Decision Tree for Normal Modes Analysis” on page 76.

Memory Usage

Prudent memory use is perhaps the simplest way to improve performance. Optimal MSC Nastran performance is achieved by minimizing disk I/O, and by providing adequate memory for numeric calculations. Memory must be reserved for “caching” I/O operations, both by MSC Nastran internally, and by the underlying operating system.

MSC Nastran I/O operations are minimized by using the command line keyword in order to set the optimal MSC Nastran memory use. Using “memorymax” will automatically set the MSC Nastran buffer cache (“Buffer Pool” or “bpool”) for an optimal balance between memory reserved for calculations, and memory used for I/O caching (75% for the solver and 25% for Buffer Pool). In addition, memory used by a MSC Nastran job should not exceed 75% of the available memory on the host computer. This amount is so that the host operating system can perform its own caching of I/O data. For example, on a computer with 256

GB memory, setting “memorymax=128gb” or “memorymax=192gb” would ensure optimal memory use for large models in the 20 to 50 million DOFs range. For smaller models, less memory will be required. The default installation value of memorymax is 50% of the total memory on the computer.

It is also possible to tune individual MSC Nastran jobs using “mem=” and “bpool=” to set MSC Nastran Open Core memory and Buffer Pool memory, instead of using “memorymax”. See Section “Memory” on page 21 in CHAPTER 2 on Linear Statics for more examples. Note that use of this technique is beyond the scope of this document and is generally not recommended.

For Distributed Memory Parallel (DMP) operations, use of “memorymax” defines the total memory used by all DMP processes. For example, a MSC Nastran job run with dmp=4 and “memorymax=100gb” (on a single host) will divide 100 GB across the four DMP processes. Setting “mem=val” for a DMP job will result in each DMP process allocating memory of length “val” and should be used with caution.

As of MSC Nastran 2018.0, there are two versions of the ACMS method. The Accelerated ACMS method was introduced in MSC Nastran 2016.0 and updated in MSC Nastran 2018.0, whereas the Original ACMS method was first added in 2008 and will eventually be deprecated. We refer to the two versions here as the **Accelerated ACMS method** and the **Original ACMS method**. We also note that the memory requirements were reduced for the Accelerated ACMS method in MSC Nastran 2018.

Table 4-1 Accelerated ACMS Memory Recommendations for MSC Nastran 2018

Model Size (DOF)	No. of Threads	Suggested MEMORYMAX=
Up to 10 Million	1-8	16 GB
	9-16	32 GB
10 – 20 Million	1-8	32 GB
	9-16	64 GB
20-40 Million	1-8	100 GB
	9-16	128 GB

Note that for Lanczos or the Original ACMS method, suggested memory requirements are roughly half of those listed above for Accelerated ACMS method.

In Figure 4-1 below, we consider various models with varying memorymax amounts used in the simulation for the Lanczos solver.

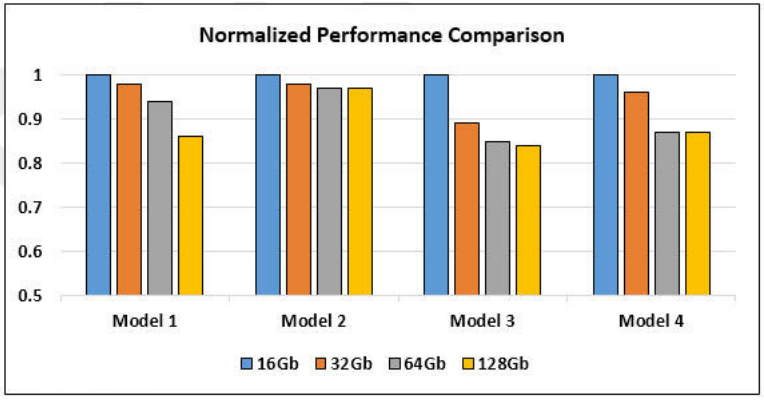


Figure 4-1 Normalized Performance Comparison of Various Models with Varying memory for the default Lanczos Solver

The performance gain from increased memory depends on a variety of factors. Model 4 illustrates a large reduction in elapsed time from the 32 GB to 64 GB run. This reduction is best explained in the F04 file. Consider:

32 GB run prints:

```
*** SYSTEM INFORMATION MESSAGE 4199 (PREFACT)
    A PORTION OF THE SPARSE FACTOR HAS BEEN CACHED IN MEMORY FOR THE FBS.
    59453 FRONTAL MATRICES OUT OF A TOTAL OF 110946 ARE STORED IN MEMORY.
    MEMORY AVAILABLE: 11077 MB
    ESTIMATED ADDITIONAL MEMORY NEEDED TO STORE THE ENTIRE FACTOR: 35942 MB
```

64 GB run prints:

```
*** SYSTEM INFORMATION MESSAGE 4199 (PREFACT)
    A PORTION OF THE SPARSE FACTOR HAS BEEN CACHED IN MEMORY FOR THE FBS.
    110942 FRONTAL MATRICES OUT OF A TOTAL OF 110946 ARE STORED IN MEMORY.
    MEMORY AVAILABLE: 23170 MB
    ESTIMATED ADDITIONAL MEMORY NEEDED TO STORE THE ENTIRE FACTOR: 23842 MB
```

128 GB run prints:

```
*** SYSTEM INFORMATION MESSAGE 4199 (PREFACT)
    THE ENTIRE SPARSE FACTOR HAS BEEN CACHED IN MEMORY FOR THE FBS.
    ALL 110946 FRONTAL MATRICES ARE STORED IN MEMORY.
    MEMORY AVAILABLE: 23170 MB
    MEMORY USED TO STORE THE ENTIRE FACTOR: 42986 MB
```

Most of the frontal matrices are stored in memory for the 64 GB run. This amount is almost twice that of the 32 GB run.

In Figure 4-2 below, we compare the performance on several hardware platforms with differing memory amounts for the Original ACMS method. As is clear, the elapsed times is highly dependent on memory. It is noteworthy that the "Medium" Linux machine is identical hardware to the "Windows" systems.

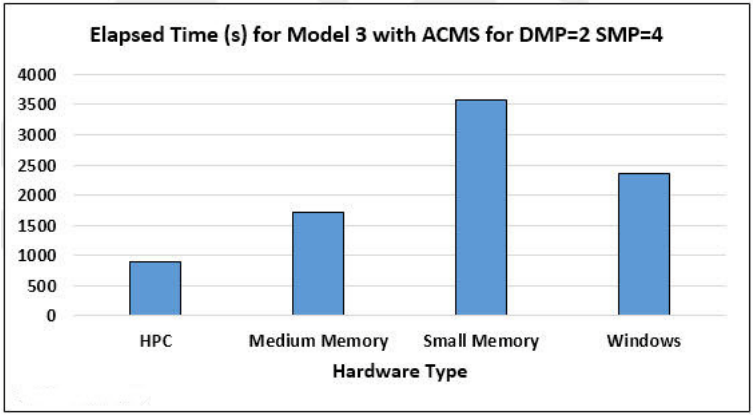


Figure 4-2 Performance Comparison of the Original ACMS method for Model 3 (1129 roots) with DMP=2 SMP=4 for Different Hardware Types

The additional memory leads to reduced elapsed time due to significant reduction in I/O. This reduction allowed the HPC system to use more of the CPU. Consider the percent CPU used by the Master process for each run (printed in the LOG file):

Small Memory

```
6869 user    985 system    59:42 elapsed    219%CPU
```

HPC Machine:

```
2852 user    237 system    14:53 elapsed    345%CPU
```

The additional memory allowed the SMP=4 on the Master to scale well.

In Figure 4-3, we compare the performance for a Lanczos analysis on a solid-element model with a few roots on the same set of hardware. The elapsed times again is highly dependent on memory. It is noteworthy that the “Medium” Linux machine is identical hardware to the “Windows” systems.

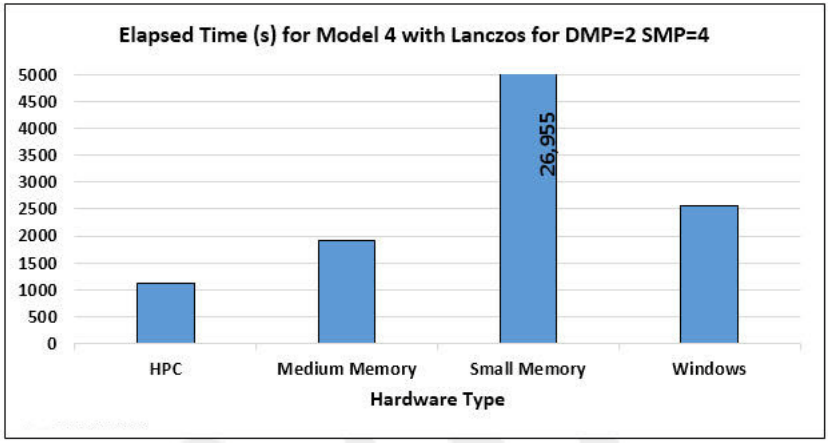


Figure 4-3 Elapsed Time (s) for Model 4 (49 roots) with Lanczos for DMP=2 SMP=4 for Various Hardware Platforms

The performance increase above on Linux systems is because of the increase in memory. The HPC system does allow for AVX2 which may improve performance by roughly 15%, but most of the improvement is because of the memory. Comparisons of the times spent in specific modules between the HPC and Small Memory Machine are below:

Small Memory Machine:

```
*** USER INFORMATION MESSAGE 5403 (LNNRIGL)
    BREAKDOWN OF CPU TIME (SEC) DURING LANCZOS ITERATIONS:
    OPERATION                 REPETITIONS    AVERAGE       TOTAL
    FBS (BLOCK SIZE=  7)              25    1498.41    37460.29
    MATRIX-VECTOR MULTIPLY            62       0.12        7.30
    SHIFT AND FACTOR                   2    1259.87     2519.74
    LANCZOS RUN                        1    2254.05     2254.05
```

HPC Machine:

```
*** USER INFORMATION MESSAGE 5403 (LNNRIGL)
    BREAKDOWN OF CPU TIME (SEC) DURING LANCZOS ITERATIONS:
    OPERATION                 REPETITIONS    AVERAGE       TOTAL
    FBS (BLOCK SIZE=  7)              26     44.55     1158.38
    MATRIX-VECTOR MULTIPLY            64      0.06        3.72
    SHIFT AND FACTOR                   2    915.05     1830.09
    LANCZOS RUN                        1    223.55      223.55
```

EigenSolver Options

There are multiple methods to determine eigenvalues: Accelerated ACMS, Original ACMS, Lanczos, Householder, Givens, etc. This section will compare the performance of the three most common methods (Accelerated ACMS, Original ACMS, and Lanczos). Input parameters of interest are:

Executive section:

```
DOMAINSOLVER ACMS (version=new)  $ Select Accelerated ACMS Method

DOMAINSOLVER ACMS (version=old)  $ Select Original ACMS Method
```

Bulk data section:

```
EIGRL   SID     F1      F2      ND  MSGLVL  MAXSET  SHFSCL  NORM
EIGRL   300     -0.1    300.0
```

Lanczos is the standard eigenvalue extraction method, and is generally faster when few modes (< 50) are required. Accelerated and Original ACMS require a DOMAINSOLVER entry (Executive system) and is generally faster when a significant number of modes is required, or if the model size is very large (20 million DOFs or more). The desired frequency range information for both ACMS versions is taken from the selected EIGRL entry in the bulk data section.

ACMS

ACMS (Automated Component Mode Synthesis) replaces the full eigensolution on all A-set DOFs with a modal synthesis of automatically generated components. ACMS is an approximation to the full Lanczos eigensolution. In order to decrease the level of approximation, users can increase the maximum frequency used internally by ACMS. The default is two times your EIGRL F-max. You can increase this by modifying your DOMAINSOLVER entry with UPFACT. The Accelerated ACMS method will generally be faster than Original ACMS method, but may require more memory. Eventually, the Original ACMS method will be deprecated.

Lanczos

The Lanczos method only makes the calculations necessary to find the roots requested. It uses Sturm sequence logic to ensure that all modes are found. When required to extract a relatively small number of eigenvalues, the bulk of the computation time for Lanczos is spent during symmetric factorization, so that optimal compute considerations are virtually the same as for linear static analysis in MSC Nastran.

Method Selection

The decisions on method selection for SOL 103 are simple:

- Few roots: Use Lanczos
- Many roots: Use Accelerated ACMS
- Few roots with very large model (20+ million DOFs): Use Accelerated ACMS

The following example compares the elapsed times for calculating various number of modes using Accelerated ACMS, Original ACMS, and Lanczos. In this example, we consider both 2D and 3D dominant models. Eigenvector output requests were turned off to exemplify the time used in calculating the modes.

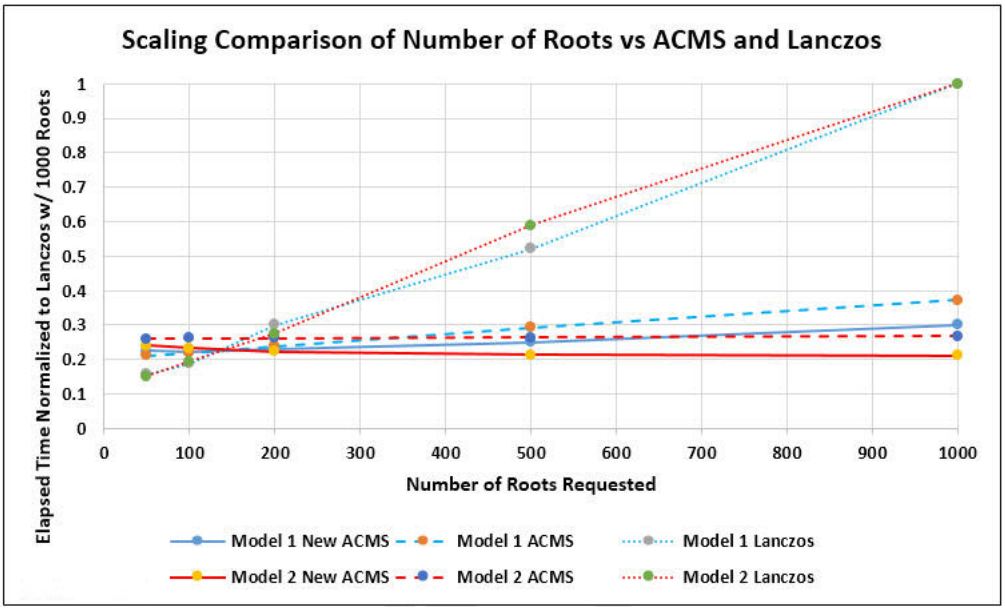


Figure 4-4 Comparison of Lanczos versus Accelerated and Original ACMS methods for a variety of Models with different number of roots

The results in Figure 4-4 illustrate that the cost of calculating additional modes with Lanczos is almost proportional to the number of roots, but the cost of calculating modes with ACMS is generally independent of the number of modes. Therefore, Lanczos may be faster with fewer modes and ACMS may be faster with more modes.

The time taken for various tasks are printed in the F04 file.

Accelerated ACMS times:

```
*** TIMING INFORMATION
                              TIP-------------------------    COLLECTOR-------------------    TOTAL-----------------------
                              CPU Seconds   Elapsed Seconds    CPU Seconds   Elapsed Seconds    CPU Seconds   Elapsed Seconds
   Matrix Assembly                1020.10            121.80        2442.69            327.52        3462.79            449.32
   Component Eigensolution        2052.82            251.80         700.63             93.58        2753.45            345.38
   Auto-SUPORT                     100.38             12.97          11.71              1.05         112.09             14.01
   Residual vectors                326.35             41.93          53.12              7.87         379.47             49.81
   [Koo](-1)              333.51              40.21          36.84              4.06         370.35              44.27
   FBS -> [GOT]           567.11              87.26         380.59             50.74         947.70             138.00
   Static Reduction of [K]        1081.09            148.64        2063.52            252.98        3144.61            401.62
   Static Reduction of [M]         640.83             81.04        5071.08            634.19        5711.91            715.22
   Static Reduction of [P]          40.05              3.19          33.90              4.93          73.95              8.13
   Q-set Assembly                 3679.21            471.51        2414.62            299.89        6093.83            771.40
   Re-orthogonalization              0.00              0.00        2746.26            925.43        2746.26            925.43
   Stack Management                 55.57              6.10         100.76             12.10         156.33             18.20
   Total Time                     9370.95           1184.41       16425.27           2686.87       25796.22           3871.28
```

Lanczos times:

```
*** USER INFORMATION MESSAGE 5403 (LNNRIGL)
    BREAKDOWN OF CPU TIME (SEC) DURING LANCZOS ITERATIONS:
    OPERATION                 REPETITIONS    AVERAGE       TOTAL
    FBS (BLOCK SIZE= 15)               95      27.35     2597.80
    MATRIX-VECTOR MULTIPLY            227       0.22       50.22
    SHIFT AND FACTOR                    4     525.83     2103.33
    LANCZOS RUN                         3    1186.94     3560.82
```

Parallel Options

The use of multiple cores can significantly improve performance for an eigenvalue analysis. Accelerated ACMS, Original ACMS and Lanczos all support Shared Memory Parallel (SMP) and Distributed Memory Parallel (DMP): Accelerated ACMS scales best with SMP, Original ACMS scales well with DMP and Lanczos scales with both SMP and DMP.

SMP may be specified either on the command line as “smp=n” or in the input file:

```
NASTRAN SMP=n
```

DMP may be specified either on the command line as “dmp=n” or (starting with MSC Nastran 2018.0) in the input file:

```
NASTRAN DMP=n
```

SMP and DMP may both be specified simultaneously. The Accelerated ACMS method will use all available cores (DMP times SMP) in a shared memory parallel multi-threading solution. The recommendation for the Original ACMS method and Lanczos is to specify SMP as the number of cores per socket and DMP as the number of sockets.

The DMP communication information is printed at the bottom of the F04 file. For example:

```
*** MPI STATISTICS FOR DISTRIBUTED NASTRAN ***

 FROM   MESSAGES  TOTAL (MB)  MAX BYTE  MIN BYTE  AVG BYTE    FROM   MESSAGES  TOTAL (MB)  MAX BYTE  MIN BYTE  AVG BYTE
------  --------  ----------  --------  --------  --------  ------  --------  ----------  --------  --------  --------
1 -> 2     18049   10281.1791909614896         8    597296   2 -> 1   1171709    5571.697    134000         8      4986
1 -> 3     18049   10281.1791909614896         8    597296   3 -> 1    736333    3048.618    134000         8      4341
1 -> 4      3421      36.469     34248         8     11178   4 -> 1    828751    4941.332    134000         8      6252
2 -> 3      3354      42.368     39792         8     13245   3 -> 2    318992    3114.114     55872         8     10236
2 -> 4     17982   10281.1781909614896         8    599521   4 -> 2         0       0.000         0         0         0

TOTAL NUMBER OF MESSAGES SENT =         3116640
TOTAL AMOUNT OF DATA XFR (MB) =       47598.135
AVERAGE MESSAGE LENGTH        =           16014.
```

ACMS

In Figure 4-5, we have a graph of elapsed times from a typical car body model from the automotive industry (Model 3). The model is an NVH job attempting to find many roots. Both Accelerated ACMS and Original ACMS options were used to compare performance of parallel options.

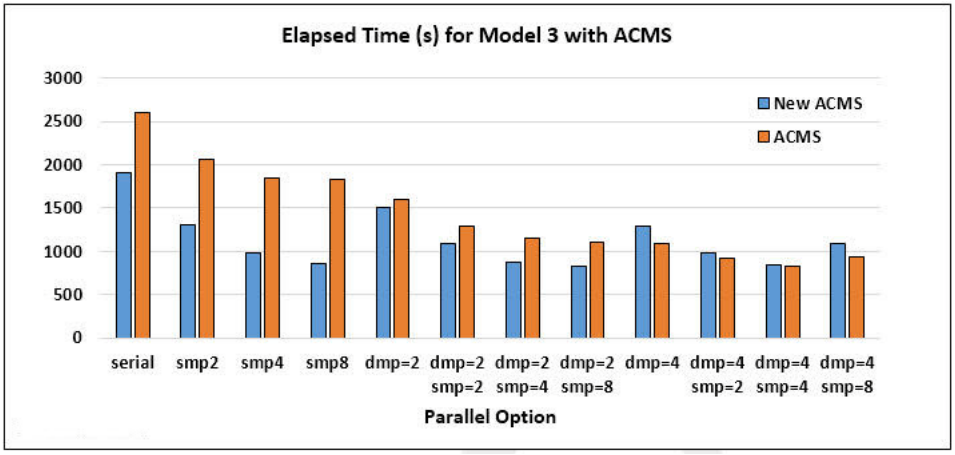


Figure 4-5 ACMS parallel time comparison for Model 3 with 1129 roots

Accelerated ACMS will use all available cores (DMP times SMP) for SMP parallel computation. Original ACMS requires DMP for optimal parallel scalability.

It is noteworthy to mention how memory is processed for DMP when “mem=max” is specified in Accelerated ACMS. The estimate program for Accelerated ACMS partitions 80% of the memory specified by “memorymax” to the Master process and the remaining 20% divided among the slave processes. Original ACMS partitions that the memory evenly amongst all the processes.

Lanczos

In Figure 4-6, we consider Model 4, which is a typical model from the automotive industry. Note that Lanczos DMP requires a frequency range (F1/F2) rather than specifying the number of roots.

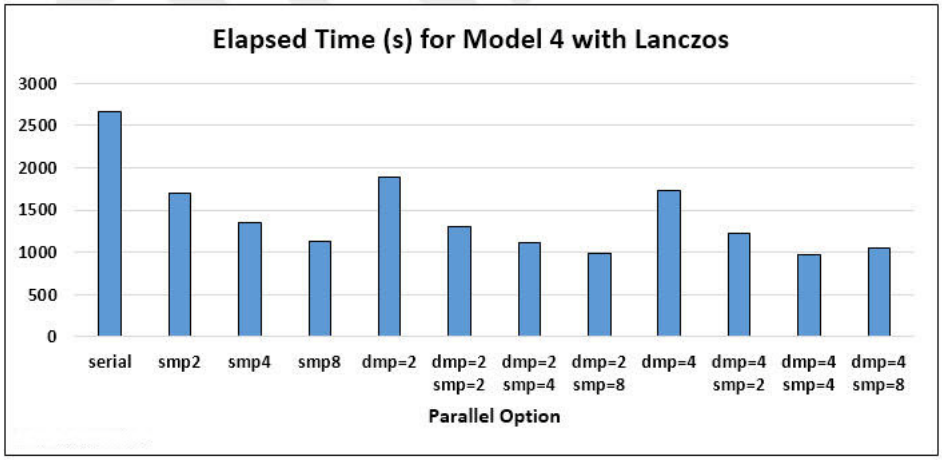


Figure 4-6 Elapsed Time (s) for Model 4 (49 roots) using Lanczos method with various parallel options

Dense models scale well with SMP. DMP may be used to get slightly better performance, or in the case of extremely large models, DMP may be used to split the work between nodes so the memory of each node may be used.

GPUs may help for models that are very dense (3D dominated), with relatively few modes, using the Lanczos solver. GPUs may hurt performance with the ACMS solver and so are turned off. Multiple GPUs may be used in DMP analysis. In below is a comparison with and without a Tesla K40M GPU for one of the solid element models with a small number of roots. Note that if FBS dominates in the case of Lanczos, then the GPU will be less effective.

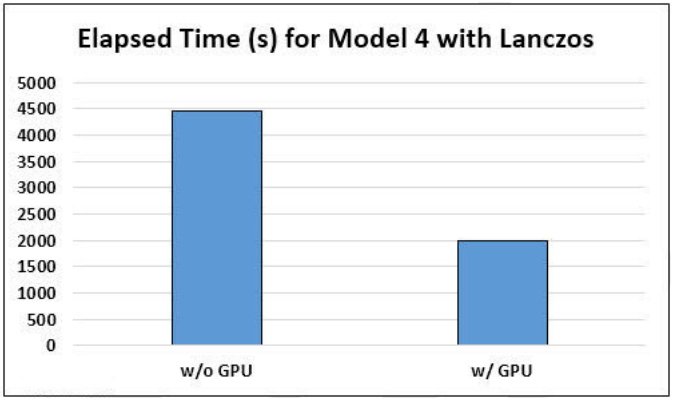


Figure 4-7 Elapsed Time (s) for Model 4 (49 roots) using Lanczos Solver with a GPU

Output Request

Output requests may result in a significant increase in time for an analysis. The uncompressed HDF5 option is faster when outputting lots of eigenvectors. If a user only needs the frequencies of interest and not eigenvectors then avoid printing the eigenvectors as the cost of printing may exceed the cost of the calculation. Input parameters of interest are:

```
DISP(PLOT) = ALL   $ Use to calculate the eigenvectors.

PARAM,POST,-1      $ Use to create OP2 files

MDLPRM,HDF5,[0,1]  $ Use to create HDF5 files
```

Note that the specification of "PLOT" means that the output quantities are not printed in the F06 file. This will save both time and disk space in the F06 file.

In the following example, we consider a typical model from the automotive industry (Model 1). The model has been modified to see the effect of performance based on output requests.

The New ACMS solver was used for this study because the compute time varies less with the number of frequencies than Lanczos to demonstrate the effect of output requests. When few roots are requested the dominant time is in the calculation of the roots. However, as the number of requested roots increases the time to output the eigenvectors becomes larger. If eigenvectors are required, then for performance reasons, it is best to write in the uncompressed HDF5 format. However, a large file size should be expected.

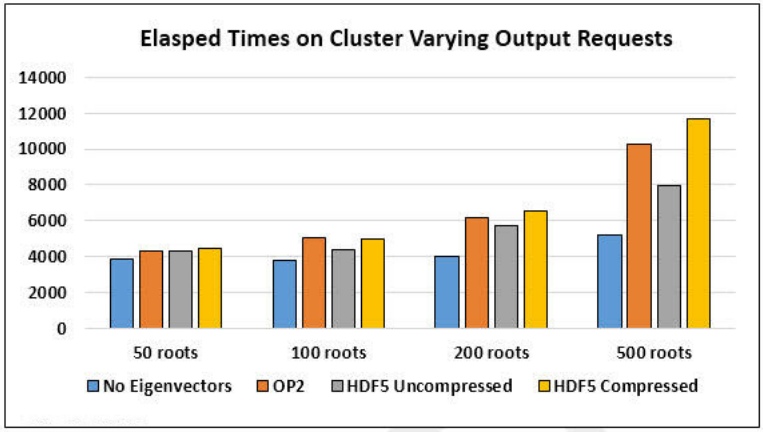


Figure 4-8 Elapsed Time for Model 1 for Various Number of Roots and Output Types

The results in Figure 4-8 illustrate the need to either eliminate extra printing of eigenvectors if they are not needed or to use uncompressed HDF5 output for performance. The size of the output file may increase for uncompressed HDF5 output as shown below:

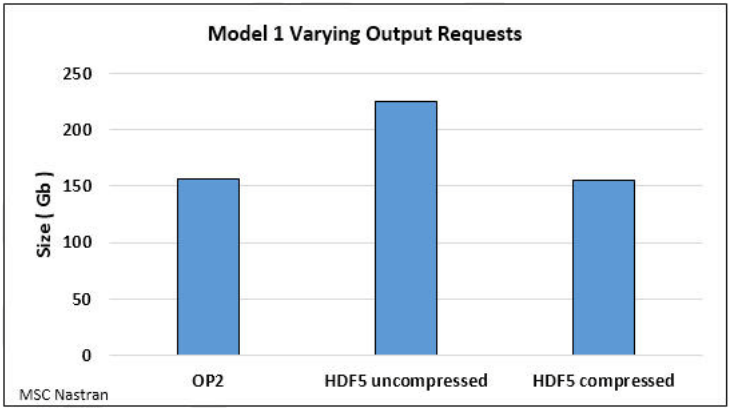


Figure 4-9 Comparison of varying output requests for Model 1 with 500 roots

Additional Topics

Sturm Count

A common output in a Normal Mode calculation in the F06 file is UIM 5010, which gives information regarding the Sturm Sequence Data for eigenvalue extraction. For example, a user may encounter the following:

```
*** USER INFORMATION MESSAGE 5010 (LNCILS)
    STURM SEQUENCE DATA FOR EIGENVALUE EXTRACTION.
    TRIAL EIGENVALUE = 8.902122D+01, CYCLES = 1.501644D+00 THE STURM COUNT = 0
*** USER INFORMATION MESSAGE 5010 (LNCILS)
    STURM SEQUENCE DATA FOR EIGENVALUE EXTRACTION.
    TRIAL EIGENVALUE = 1.602323D+05, CYCLES = 6.370818D+01 THE STURM COUNT = 11
```

before they see the "E I G E N V A L U E A N A L Y S I S S U M M A R Y". We summarize below the meaning of this information for the curious user.

The classic real eigenvalue problem for the mechanical vibrations is expressed as:

$$[K - \lambda_i M]\{\phi_i\} = 0 \qquad \text{i} = 1, 2, 3 \ldots$$

where K is the symmetric positive semi-definite stiffness matrix, M is the positive definite mass matrix, $\lambda=2\pi f^2$ is the scalar eigenvalue and ϕ is an eigenvector. The Sturm sequence check states that for any positive frequency shift $K_{shift} = K - \lambda M$ that the number of eigenvalues below the shift frequency is the number of negative terms in the diagonal matrix, D, resulting from the symmetric factorization of the shifted matrix, K_{shift}:

$$L * D * L^t = K_{shift}$$

The Sturm sequence check is used in the Lanczos algorithm to determine how many eigenvalues to calculate within the frequency range of interest and to help control the shift strategy to insure that all the required eigenvalues and eigenvectors are calculated.

Number of Roots Found by ACMS

ACMS, being a component mode synthesis, is an approximation of a full eigenvalue analysis. The eigenvalues produced by ACMS are computed using reduced stiffness and mass matrices. While the stiffness reduction employed during static reduction is exact, the mass matrix reduction will introduce some level of approximation. This approximation is due in large part to the frequency cutoff used for obtaining the eigenvectors for the interior degrees of freedom for each component. By default, the frequency cutoff is two times larger than the user's maximum frequency for the eigenvalues (Hz). Users may increase this frequency cutoff by using the UPFACT parameter, which is set on the DOMAINSOLVER executive control entry. For example, to set the frequency cutoff to three times the eigenvalue maximum frequency set UPFACT to 3.0:

```
DOMAINSOLVER ACMS (UPFACT=3.0,…)
```

Note that these details apply both to the Accelerated ACMS method and the Original ACMS method. Theoretical details for component modal synthesis may be found in Chapter 18 of MSC Nastran Dynamic Analysis User's Guide (see the "Dynamic Reduction" on page 667)

Additional Acceleration Options for the Accelerated ACMS Method

The ACMS method employs a domain decomposition to generate matrix components based on the matrix connectivity of the global stiffness and mass matrices of the model. The domain decomposition employs an algorithm based on nested dissection which recursively divides the global matrices into smaller and smaller sub-components, until a minimum size is reached for a component. These final components are known as "leaf" or "tip" components. For very large models, the number of atomically generated components may exceed 100,000. In some cases, better performance may be obtained by reducing the number of components generated by the domain decomposition. This may be accomplished by increasing the final "tip" domain size using the TIPSIZE parameter, which is set on the DOMAINSOLVER executive control entry. The default value of TIPSIZE is 200 (compressed) DOFs. Users may set TIPSIZE to 300 for very large models:

```
DOMAINSOLVER ACMS (VERSION=NEW, TIPSIZE=300,…)
```

Note, setting TIPSIZE greater than 300 for VERSION=NEW is not recommended.TIPSIZE is applicable to the METISO partition method only ("PARTMETH=METISO"), which is the default partition method for the Accelerated ACMS method.

Models and Hardware Details

A variety of models and hardware will be used in the examples. Unless otherwise stated, all examples will have been run on our HPC system. Cases where other systems are used will be explicitly noted. The models used are given below:

Table 4-2 Summary of Model Types for Performance Comparison

Model #	DOF	N Grid	2D elements	3D elements	
Model 1	19,149,108	3,191,518	3,201,068	0	Floor pan
Model 2	8,627,970	2,867,337	31,838	1,526,072	Differential
Model 3	3,799,278	821,589	704,415	211,596	Car Body
Model 4	2,604,102	881,748	0	561,545	Piston

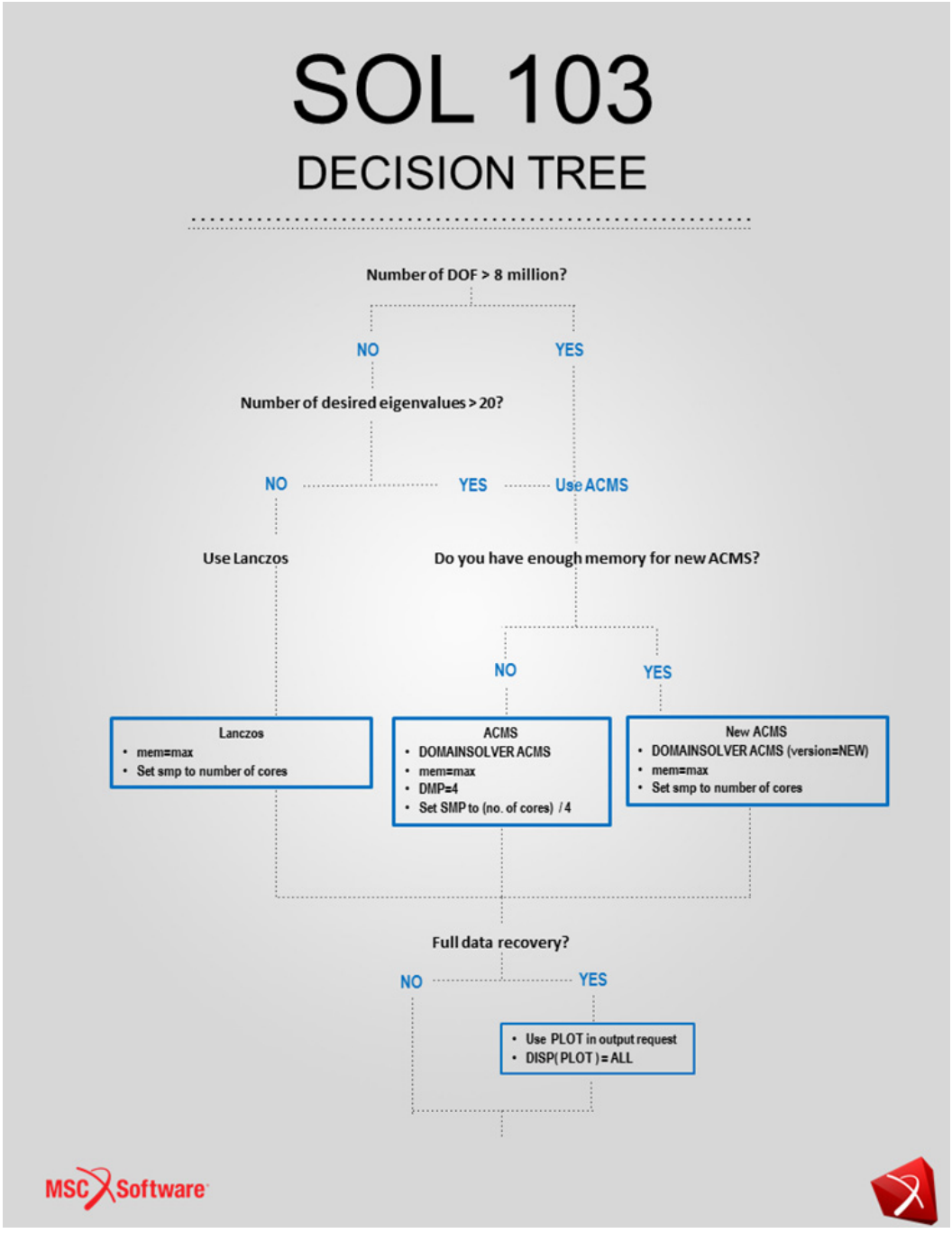
Car bodies tend to request many modes, but solid models tend to request fewer modes. In our examples we will consider both possibilities, but the general usage is as stated.

These machines are a large HPC machine, a medium memory machine, a small memory machine, and a Windows machine

Table 4-3

Memory	Memory	CPUs	HDD	OS
HPC System	256 GB	2x10 Intel® Xeon® CPU E5-2550 v3 @2.60GHz	4 x 600 GB 15K RPM in RAID0	RH 7.1
Medium Memory	128 GB	2x10 Intel(R) Xeon(R) CPU E5-2650 v2 @ 2.60GHz	300 GB 15000 RPM	SuSE 11 SP3
Small Memory	32 GB	2x10 Intel® CPU @2.8GHz	1 x 1 TB 7200 RPM Hard Drive	RH 6.3
Windows	128 GB	2x10 Intel(R) Xeon(R) CPU E5-2650 v2 @ 2.60GHz	300 GB 15000 RPM	Windows 10

Decision Tree for Normal Modes Analysis



Main Index

5 Optimal Performance of Direct Frequency Response Analysis (SOL 108)

- Introduction 78
- Memory Usage 79
- Matrix Solver Options 83
- Parallel Options 85
- Output Request 89
- Additional Topics 90
- Model and Hardware Details 91
- Decision Tree for Direct Frequency Response Analysis 93

Introduction

Direct Frequency Response Analysis is one of the more expensive solution sequences to employ in a simulation study, but one of the best for taking advantage of parallelism. It requires solving the full frequency response problem, given by

$$[-\omega^2 M + i\omega B + K]u(\omega) = P(\omega)$$

For a few or several frequencies, given by ω, where M is the mass matrix, B is the damping matrix, and K is the stiffness matrix. The matrix problem is typically complex, as compared to linear statics, and thus the direct solvers require 2-4x as much memory to run in-core. Typically, users employ direct frequency response analysis for the following simulation studies:

- simulate many frequencies with a limited output request, for example, DISP/VELO/ACCE for a small set of GRIDs.
- simulate a few frequencies (< 10) with full data recovery, e.g., DISP(PLOT)=ALL and STRESS(PLOT)=ALL.

Any other frequency response studies should be performed using Modal Frequency Response Analysis (SOL 111), which we describe in Chapter 6: Optimal Performance of Modal Frequency Response Analysis (SOL 111).

The performance of the direct frequency response analysis is dominated by the following two operations:

- Solving the matrix problem for many frequencies
- Writing the requested data to an output file

The relative cost difference of these two operations in the simulation is dependent on the simulation study and the model size. In most cases, though, the solution of the matrix problem dominates.

In Section "Matrix Solver Options" on page 83, the options are described for solving the matrix problem, but we mention them here for references purposes. We consider the following Sparse Direct Solvers:

- MSC Sparse Direct Solver (MSCLDL)
- MSC Unsymmetric Sparse Direct Solver (MSCLU)
- UMFPACK Sparse Direct Solver (UMFLU)
- Pardiso Sparse Direct Solver (PRDLU)

In Section "Additional Topics" on page 90, we also mention the Krylov solver and our recommendations against its use in SOL 108 except for very special cases.

In this chapter, we use a series of models and hardware platforms as described in Section "Model and Hardware Details" on page 91 to illustrate performance. Also, unless otherwise stated, we generate the results on the HPC Machine as described in Section "Model and Hardware Details" on page 91. Finally, in Section "Decision Tree for Direct Frequency Response Analysis" on page 93, we provide a decision tree that allows the user to make a choice on the various settings.

Memory Usage

Understanding Memory

In previous chapters, we have described how a user can set memory in an MSC Nastran simulation. In this chapter, we focus only on the use of memorymax and the default of "mem=max". In SOL 108, when a user runs a simulation with the following command

```
nast20180 memorymax=192gb model_108.dat
```

On a machine with 256 GB of RAM, the amount of memory given to the simulation will be 192 GB. In contrast to linear statics, the estimate program will not be used here to determine how much memory goes to Open Core. Instead, we give 75% of the available memory to Open Core and the remainder to Buffer Pool. Thus, we see in the F04 file the following memory breakdown for the various subsystems:

```
        USER OPENCORE (HICORE)          = 150466 MB
        EXECUTIVE SYSTEM WORK AREA      =      9 MB
        MASTER(RAM)                     =      2 MB
        SCRATCH(MEM) AREA               =     50 MB (     100 BUFFERS)
        BUFFER POOL AREA (BPOOL4)       =  50176 MB (  100351 BUFFERS)


        TOTAL MSC NASTRAN MEMORY LIMIT = 200704 MB
```

Like in SOL 101, there are similar requirements with regards to memory to get the best performance. The main requirement is that the solver runs in-core. The specifics of this requirement are different depending on the actual solver and the performance degradation from not having the solver run in-core can vary.

For MSCDL, we refer the reader to Chapter 3 on Linear Statics for details of the memory consumption of that solver.

For MSCLU, we can see the details of the memory usage in User Information Message 4216 as we see below in a text block taken from F04 file. Note that we have used bold to highlight the unsymmetric descriptor of the decomposition for reference.

```
*** USER INFORMATION MESSAGE 4216 (UDSFA)
                    PARAMETERS FOR PARALLEL SPARSE UNSYM. DECOMP OF DATA BLOCK  SCRATCH ( TYPE=CSP ) FOLLOW
                    MATRIX SIZE = 1442144 ROWS             NUMBER OF NONZEROES =95574706 TERMS
          NUMBER OF ZERO COLUMNS =       0        NUMBER OF ZERO DIAGONAL TERMS =       0
                   SYSTEM (107) =   32777                      REQUESTED PROC. =       9 CPUS
              CPU TIME ESTIMATE =   18895 SEC                 I/O TIME ESTIMATE =     456 SEC
    ESTIMATED MEMORY REQUIREMENT =    3094 MB                  MEMORY AVAILABLE =   49090 MB
   EST. INTEGER MEMORY IN FACTOR =     875 MB                EST. NONZERO TERMS = 1092538 K TERMS
    ESTIMATED MAXIMUM FRONT SIZE =   11379 TERMS                 RANK OF UPDATE =      64
 15:22:48     1:19      289.0         0.0        86.0       10.3    USPD BGN TE=18895
 15:29:06     7:37    36018.0     35729.0     12223.0    12137.0    USPD END
 *** USER INFORMATION MESSAGE 6439 (UDSFA)
     ACTUAL MEMORY AND DISK SPACE REQUIREMENTS FOR SPARSE UNSYM. DECOMPOSITION
     SPARSE DECOMP MEMORY REQUIRED =    2423 MB               MAXIMUM FRONT SIZE =   11379 TERMS
          INTEGER MEMORY IN FACTOR =     152 MB           NONZERO TERMS IN FACTOR = 1092538 K TERMS
    SPARSE DECOMP SUGGESTED MEMORY =    3634 MB
```

We can see in the above User Information Message in the F04 file that the sparse decomposition suggested memory is 3634 MB, or 3.6 GB, and that the minimum should be at least 2423 MB, or 2.4 GB. Even for this, we still perform a large amount of I/O in this case – 532.3 GB. We ran this model with plenty of memory beyond the suggested amount.

For UMFLU, we can see the details of the memory usage in the F04 file for the UIM 4220. An example of this follows:

```
*** USER INFORMATION MESSAGE 4220 (UMFDRV)
     PARAMETERS FOR SPARSE UNSYMMETRIC DECOMPOSITION OF DATA BLOCK SCRATCH  (TYPE=CSP ) FOLLOW
     MATRIX SIZE =      30456 ROWS   NUMBER OF NONZEROS =          774027 TERMS
     INITIAL MEMORY REQUIREMENT    =         2413455 WORDS (      19 MB)
     MEMORY AVAILABLE FOR UMFPACK =      3216613216 WORDS (   24541 MB)
     TOTAL MEMORY AVAILABLE       =      3219026670 WORDS (   24560 MB)
 18:01:04     0:06       33.0         0.0          9.5         0.1    UMFFACZ BEGN
 18:01:06     0:08       33.0         0.0         18.5         9.0    UMFFACZ END
 *** USER INFORMATION MESSAGE 4226 (UMFDRV)
     ACTUAL MEMORY AND DISK SPACE REQUIREMENTS FOR SPARSE UNSYMMETRIC DECOMPOSITION
     NUMBER OF NONZERO TERMS IN UPPER FACTOR =         5621615
     NUMBER OF NONZERO TERMS IN LOWER FACTOR =         5623003
     MINIMUM MEMORY REQUIREMENT              =       40015495. WORDS (     306 MB)
 *** USER INFORMATION MESSAGE 4234 (UFBS)
     UFBS TIME ESTIMATE TO FORM SCRATCH( TYPE=CSP ) CPU=2, I/O=0,TOTAL=2,PASSES=1
 18:01:06     0:08       33.0         0.0        20.6         2.1    UFBSBEGN P= 1
 18:01:06     0:08       33.0         0.0        20.6         0.0    UFBSEND
```

As can be seen from the UIM 4220, the minimum memory required for this particular model is 306 MB. Thus, a user should look at the solver memory defined by open core in the F04 file to determine if the memory is sufficient or furthermore if there is too much memory and some of it can be distributed to Buffer Pool. For this case, we have the following in the F04 file for memory decomposition where we used "memorymax=32gb":

```
        USER OPENCORE (HICORE)          = 24559 MB
        EXECUTIVE SYSTEM WORK AREA      =     7 MB
        MASTER(RAM)                     =     2 MB
        SCRATCH(MEM) AREA               =     6 MB(     100 BUFFERS)
        BUFFER POOL AREA (BPOOL4)       =  8192 MB(  131057 BUFFERS)

        TOTAL MSC NASTRAN MEMORY LIMIT = 32768 MB
```

Thus, we have 3219057681 WORDS, or approximately 24 GB available to the solver. We could have significantly reduced the memory for this model to still have it run in-core or passed more memory to Buffer Pool.

For the unsymmetric version of the Pardiso Solver, PRDLU, we have the same details as for the symmetric version, PRDLDL, for SOL 101. The same example used for UMFLU is used in the following to generate this output in the F04 file:

```
PARAMETERS FOR INTEL MKL PARDISO PARALLEL SPARSE SOLVE
      MATRIX SCRATCH  on unit 305 ( TYPE=2) FOLLOW
                  MATRIX SIZE =30456 ROWS
              NUMBER OF NONZEROES =774026 TERMS
            NUMBER OF ZERO COLUMNS =0
                   NUMBER OF LOADS =1
                      SYSTEM (107) =8
              MKL REQUESTED PROC. =8 CPUS
      MEMORY PARAMETERS FOR INTEL MKL PARDISO PARALLEL DECOMPOSITION FOLLOW
                     AVAIL. CORE MEMORY =19631 MB
          APPROX. REQUI. IN-CORE MEMORY =224 MB
      APPROX. REQUI. OUT-OF-CORE MEMORY =81 MB
```

In this case, PRDLU only needed 224 MB to run in-core, and avoid spill. Thus, we could have again reduced the memory. Rerunning this model with "memorymax=600mb" yielded the following:

```
PARAMETERS FOR INTEL MKL PARDISO PARALLEL SPARSE SOLVE
```

```
MATRIX SCRATCH  on unit 305 ( TYPE=2) FOLLOW
             MATRIX SIZE =30456 ROWS
     NUMBER OF NONZEROES =774026 TERMS
  NUMBER OF ZERO COLUMNS =0
         NUMBER OF LOADS =1
            SYSTEM (107) =8
     MKL REQUESTED PROC. =8 CPUS
MEMORY PARAMETERS FOR INTEL MKL PARDISO PARALLEL DECOMPOSITION FOLLOW
               AVAIL. CORE MEMORY =330 MB
    APPROX. REQUI. IN-CORE MEMORY =223 MB
APPROX. REQUI. OUT-OF-CORE MEMORY =81 MB
```

where the solver (or Open Core) had 450 MB of memory (or 75% of 600 MB) and the other 150 MB was given to Buffer Pool. Of the 450 MB of memory, only 75% was left to PRDLU as the other memory was required for storing matrices and other vectors. With this knowledge, we could reduce the memorymax available to 586 MB and still have PRDLU be in-core.

Examples

In the first example, we consider Model 1 on the Small Memory Machine. For this model, we could not use PRDLU due to memory restrictions and UMFLU was automatically switched to MSCLDL since it was a complex symmetric matrix. Thus, we see in Figure 1 the results for Model 1 where we vary the value of memorymax, thus generating different I/O and different elapsed times.

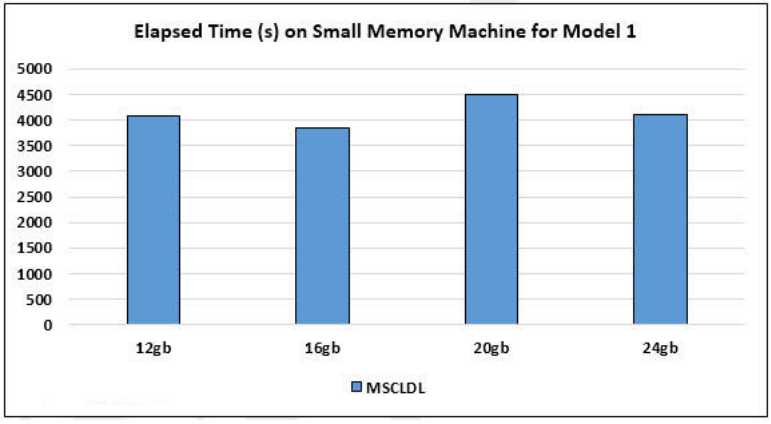


Figure 5-1 Elapsed time on Small Memory Machine for Various Memorymax Amounts with “mem=max” for Model 1 using the Default MSCLDL Solver.

The default value of memorymax for “mem=max” option is 50% of physical RAM, which for this machine is 16 GB. We have also performed this simulation using “mem=16gb” and saw a slight performance degradation. In all cases above, and the case of “mem=16gb”, the amount the I/O is hundreds of GB thus implying that the problem is I/O limited. Therefore, we don’t see a significant performance difference between the various cases.

In Figure 5-2, we consider Model 3 on the HPC Machine, where we compare the default solver to PRDLU and we vary the memorymax amount so that PRDLU is forced to go out-of-core.

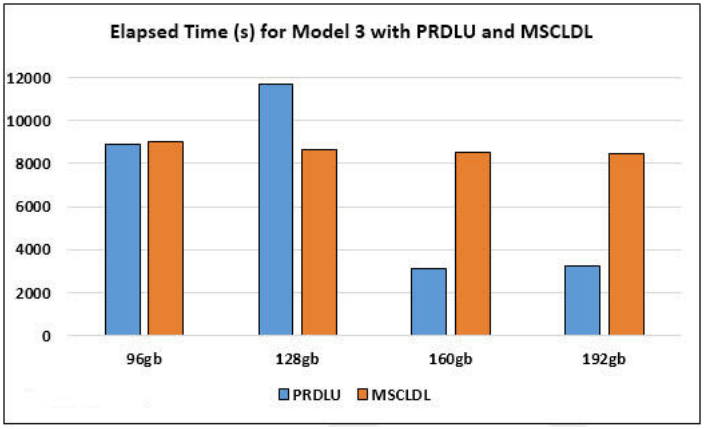


Figure 5-2 Elapsed time for Various Memorymax Amounts with “mem=max” for Model 3 using PRDLU and MSCLDL.

The first observation from Figure 5-2 is that PRDLU can be 2.5x faster than the default MSCLDL solver with sufficient memory. The second observation is that PRDLU can be slower than the default MSCLDL solver with reduced memory. For “memorymax=96gb” and “memorymax=128gb”, PRDLU is running out-of-core and thus has reduced performance. For example, we see the following in the F04 file for the case of “memorymax=128gb”:

```
PARAMETERS FOR INTEL MKL PARDISO PARALLEL SPARSE SOLVE
      MATRIX SCRATCH  on unit 305 ( TYPE=1) FOLLOW
                  MATRIX SIZE =8654733 ROWS
           NUMBER OF NONZEROES =315553279 TERMS
        NUMBER OF ZERO COLUMNS =0
    NUMBER OF RIGHT-HAND SIDES =1
                  SYSTEM (107) =16
           MKL REQUESTED PROC. =16 CPUS
     MEMORY PARAMETERS FOR INTEL MKL PARDISO PARALLEL DECOMPOSITION FOLLOW
                    AVAIL. CORE MEMORY =72445 MB
         APPROX. REQUI. IN-CORE MEMORY =87490 MB
      APPROX. REQUI. OUT-OF-CORE MEMORY =23095 MB
```

We also see the surprising result that PRDLU is faster at “memorymax=128gb” than “memorymax=96gb”. While we cannot make a definite statement about this result, the result follows the same observation that PRDLU is faster when it is running in-core as compared to the default solver when the matrix is complex symmetric. Note that in Section 5.2 we will see that the story is different when the matrix is unsymmetric.

While we do not have access to the source code to understand why Pardiso out-of-core is so much slower in SOL 108 than in SOL 101, we can speculate. The first reason is we are solving a symmetric positive definite system in SOL 101, which has minimal pivoting needs. In SOL 108, we are solving a complex matrix with potential indefinite features, i.e., negative eigenvalues. The second reason is that it is likely a very different code path in the case of complex symmetric matrices in the Pardiso Solver. In any case, the results in Figure 5-2 can be used as a guide to understand the memory needs for PRDLU.

Matrix Solver Options

The method selection involves specifying the approach to solving the matrix problem that returns the complex displacements given the applied loads. In linear algebra terminology, we want to select the method that finds “x” in the linear algebra equation “Ax=b”. There are two types of methods for solving Ax=b: direct methods and iterative methods. We only focus here on direct methods; we remark in Section “Additional Topics” on page 90 on an iterative method that is referred to in MSC Nastran as the Krylov method.

The default solver option in SOL 108 automatically selects MSCLDL or MSCLU depending on the matrix type. The matrix solution is complex, due the presence of damping effects. We select the default solver based on whether the resulting complex matrix is complex symmetric or not. The following cases generally lead to complex unsymmetric matrices, and thus, use MSCLU as the default solver:

- Exterior acoustic analysis (infinite elements)
- Interior/exterior acoustic coupling analysis
- Unsymmetric direct matrix input (x2PP matrices)
- Rotordynamics

Understanding the type of matrices generated by the user model is useful in selecting the solver option if the default wants to be overridden. The two options for overriding the defaults are UMFLU and PRDLU. These can be chosen with SPARSESOLVER options. For UMFLU or PRDLU, it is selected in the Executive Control section via:

```
SPARSESOLVER FRRD1(FACTMETH=UMFLU)
```

or

```
SPARSESOLVER FRRD1(FACTMETH=PRDLU)
```

We describe these specific options below and divide up the remaining discussion by symmetric and unsymmetric.

Symmetric

Two models, Model 1 and Model 3, lead to complex symmetric matrices. We presented in Figure 1 and Figure 2 performance results for these two models on different machines. The result of that study was that PRDLU was faster than MSCLDL when PRDLU was running in-core. In Figure 5-3 below, we consider Model 1 again on the HPC Machine and observe the results for when there is sufficient memory.

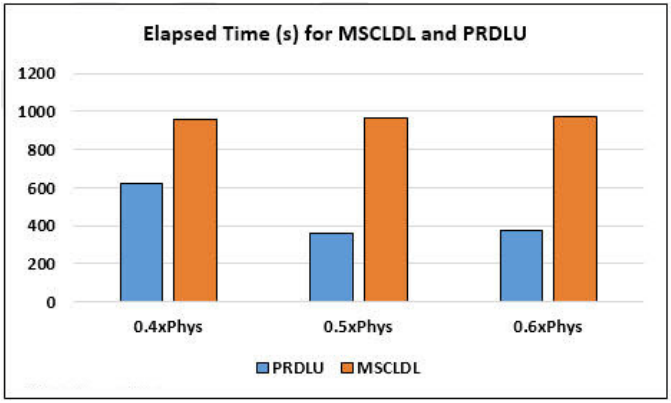


Figure 5-3 Elapsed Time for MSCLDL and PRDLU for Model 1 on the HPC Machine for Various Memorymax Values with SMP=16

As we can see in Figure 5-3, it is faster to use PRDLU for Model 1 when it can run in the given memory limitations. We ran on the HPC Machine down to "memorymax=0.3xPhys" and PRDLU was still faster. However, at "memorymax=0.2xPhys", PRDLU could not run but the default MSCLDL solver could run, thus again illustrating the case of having enough memory to use PRDLU.

Unsymmetric

Model 2 and 4 are the two exterior acoustic cases for which the matrix is not complex symmetric. The default solver for this case is MSCLU. Unfortunately, MSCLU is not as efficient as the symmetric solver MSCLDL, and thus the performance difference between it and PRDLU and UMFLU is significant. We see this first for Model 4 below in Figure 5-4 where UMFLU and PRDLU are superior to MSCLU.

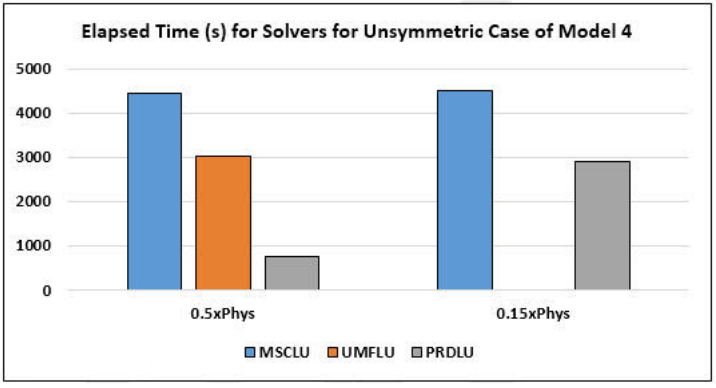


Figure 5-4 Elapsed time for MSCLU, UMFLU, and PRDLU for Unsymmetric Case of Model 4 for Various Memorymax Amounts with SMP=8

As we can see from Figure 5-4, PRDLU is a very powerful solver for this case and this it is highly recommended for a SOL 108 model where the matrix results in being unsymmetric. We also note that UMFLU failed for "memorymax=0.15xPhys" due to insufficient memory.

In the next example for Model 2, we see that the performance difference is even greater between MSCLU and PRDLU. In the case of Model 2, UMFLU was unable to solve it due to robustness issues. See the results in Figure 5-5.

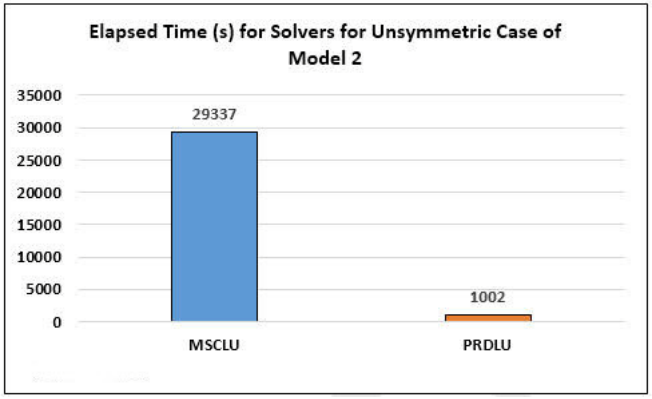


Figure 5-5 Elapsed time for MSCLU and PRDLU for Unsymmetric Case of Model 2 with SMP=8

Parallel Options

Direct Frequency Response is the epitome of parallel processing in MSC Nastran. There are three parallel options available to users: shared memory parallelism (SMP), distributed memory parallelism (DMP), and general purpose graphics processing units (GPGPU). We describe the benefit of each. The conclusions will be:

- DMP scales almost linearly up to the number of frequencies with sufficient memory.
- SMP scales well and should be added once memory for DMP is exceeded.
- DMP should be max of 4 or number of frequencies on single node.
- Use PRDLU if sufficient memory exists.
- Using multiple nodes may help because each DMP process will be split amongst more hardware and the extra RAM/CPUs will most likely be able to be used efficiently.
- GPUs may be used to enhance an already parallel solution if not using PRDLU.
- More SMP may be more beneficial than GPUs

Number of Frequencies and Parallel Performance

In the first example, we investigate how the number of frequencies affect parallel performance for two different machines with different memory amounts. The calculations from the response of each frequency is independent. Therefore, if there is sufficient memory, DMP will scale almost linearly to the number of frequencies. This is illustrated in Figure 5-6 below:

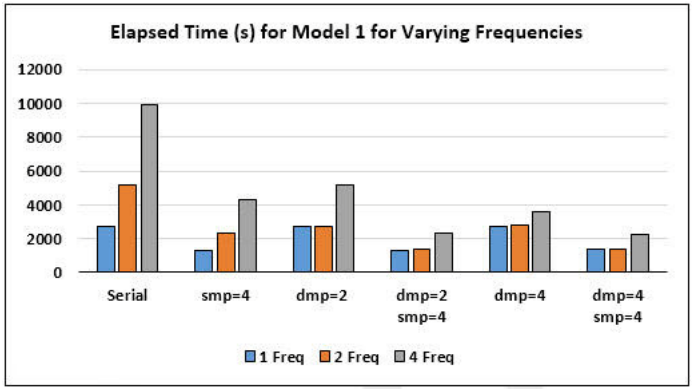


Figure 5-6 Scaling of MSCLDL for DMP and SMP as the Number of Frequencies Varies

We reran this job on the Medium Memory Machine with less memory and obtained the results in Figure 5-7.

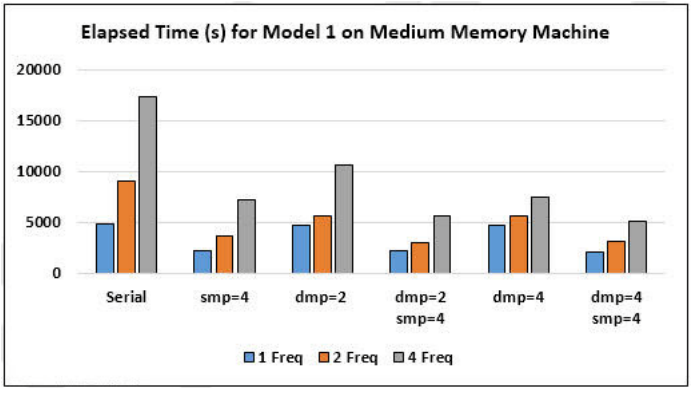


Figure 5-7 Comparison with MSCLDL for how the Number of Frequencies Affects Parallel Performance on the Medium Memory Machine

The above two figures illustrate:

- The serial times confirm that the time spent is linearly proportional to the number of frequencies.
- DMP does not scale beyond the number of frequencies.
- DMP on a machine with less memory can affect the overall performance due to higher I/O costs.
- SMP scales in all cases but at reduced scalability as SMP grows.

Solver Options and Parallel Performance

As described in the previous section, there are several possible methods to use in Direct Frequency Response depending on the matrix type: MSCLDL, MSCLU, PRDLU, and UMFLU. PRDLU is the fastest option, but may encounter issues in cases of insufficient memory. For the case below, the matrix is complex symmetric so only MSCLDL or PRDLU is applicable. Results are in Figure 5-8 below.

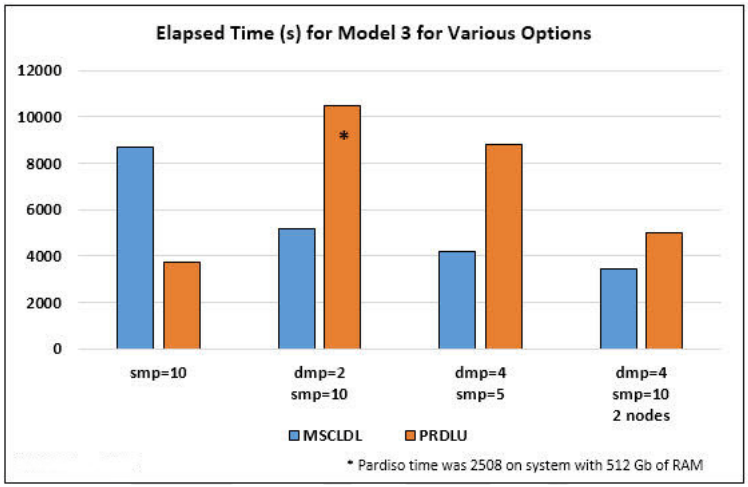


Figure 5-8 SOL 108 with Various SPARSESOLVER Options and 10 Frequencies in Frequency Response Comparing how Parallel Options Scale

The chart in Figure 5-8 illustrates:

- MSCLDL scales consistently with DMP/SMP
- PRDLU is the fastest option when there is sufficient memory (always use with "memorymax=0.75xPhys" so that it is more likely to run in-core).
- PRDLU with DMP=2 and DMP=4 both went out-of-core.
- PRDLU with DMP=4 has the following F04 file output for PRDDSV:

```
MEMORY PARAMETERS FOR INTEL MKL PARDISO PARALLEL DECOMPOSITION FOLLOW
                AVAIL. CORE MEMORY =23696 MB
     APPROX. REQUI. IN-CORE MEMORY =81979 MB
 APPROX. REQUI. OUT-OF-CORE MEMORY =19860 MB
```

- PRDLU with DMP=2 was ran on a node of the HPC Machine with 512 GB of RAM and it completed in 2508 s – fastest result of all cases.
- PRDLU with DMP=4 on two nodes with default "memorymax=0.5xPhys" ran out-of-core but paid less of a performance penalty.

Unsymmetric Case

The two unsymmetric cases include exterior acoustics. As will be seen below in Figure 5-9 for Model 4, the recommended solver is PRDLU.

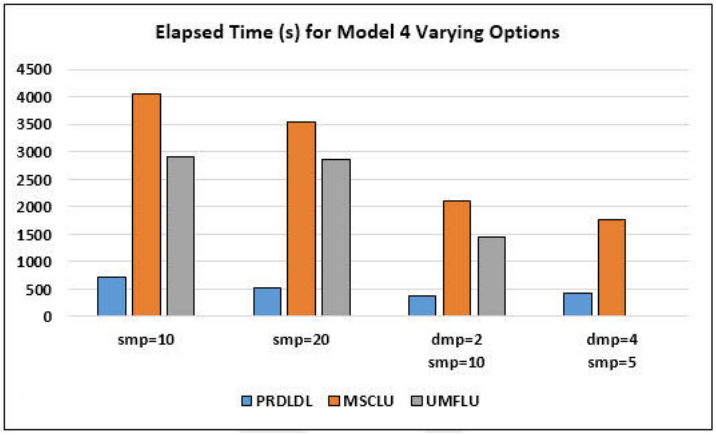


Figure 5-9 External Acoustics model comparing various solvers and parallel options

Additionally, PRDLU is more stable than the UMFLU option in addition to being faster. Also, for these types of models, PRDLU may run with slightly less memory than MSCLU. Thus, the below case in Figure 5-10 will focus on the PRDLU solver for this external acoustics model.

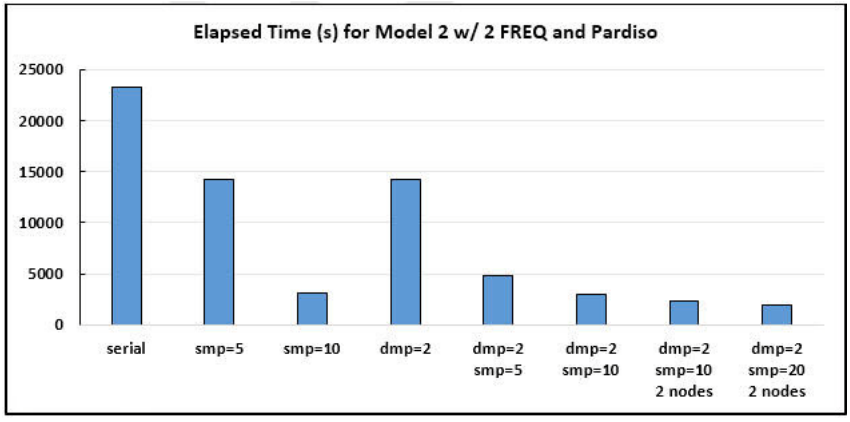


Figure 5-10 External Acoustics model comparing parallel options for the PARDISO Solver for 2 frequencies.

Exterior acoustics require more resources, but scale in the same way as a case without exterior acoustics. The computation is still independent for each frequency response.

GPU scaling

GPUs may be used to enhance a run. The below chart was made on the Medium Memory Machine with a Tesla K40m with 12GB GDDR5 memory. DMP may also be used with GPUs if you have multiple GPUs but we do not provide any specific results for this case here

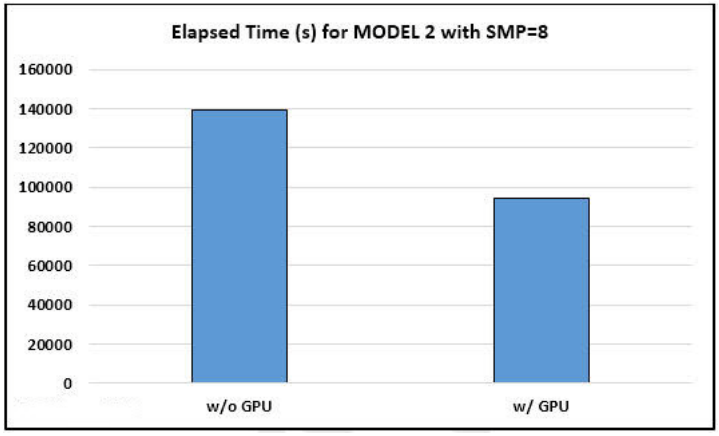


Figure 5-11 Comparing various GPU options for Model 2

The above chart shows a 30% improvement with the GPU. The actual performance gain can be very dependent on the model and furthermore additional SMP may help more than a GPU. Finally, the biggest gain from the use of a GPU is to use in combination with serial, i.e., the gain from just adding a GPU to a serial run can be significant.

Output Request

As stated in the beginning of this chapter, the output requests in SOL 108 are typically of a very specific type. For example, they are generally of the form:

- Simulate many frequencies with a limited output request, e.g., DISP/VELO/ACCE for a small set of GRIDs.
- Simulate a few frequencies (< 10) with full data recovery, e.g., DISP(PLOT)=ALL and STRESS(PLOT)=ALL.

In the first case for a limited output request, the typical user often employs a PCH file for post-processing and the output request is a very small part of the overall elapsed time. In the second case of full data recovery, the situation is very similar to what was described in Chapter 3 on linear statics. We briefly review it here.

A large output request can write to an OP2 file, an HDF5 file, or a PCH file. Additionally, the user may request output directly to the F06 file. A large output request of the following form:

```
DISP(PLOT) = ALL
STRESS(PLOT) = ALL
```

with

```
PARAM,POST,-1
```

or

```
MDLPRM,HDF5,0
```

can have a noticeable impact on the wall clock time. The choice of whether to use an OP2 or HDF5 file will have a minimal impact on the wall clock time.

Note that a punch file can be generated with

```
DISP(PUNCH,PLOT) = ALL
STRESS(PUNCH,PLOT) = ALL
```

but the use of a PCH file can have a noticeable impact on performance as we will and it will generate a very large output file. Finally, note that the specification of "PLOT" means that the output quantities are not printed in the F06 file. This will save both time and disk space in the F06 file and should be always used for large output requests.

To get the best performance for the case of a large output request (either use of ALL or a set with a very large number of grid point or element ids), it is imperative that the user following some basic standards:

1. The user should avoid directing output to a PCH file if possible
2. The user should employ the "PLOT" option to eliminate writing to the F06 file.
3. The user should direct output to a fast local disk such as an SSD or a fast spinning HHD.
4. The user should not use more than 70% of physical memory for the simulation to leave some memory to the operating system to buffer I/O.
5. The user should use (MDLPRM, HDF5, 0), which will not compress the data if they are employing HDF5 and have plenty of disk space.

Example

As the output requests are typically small for SOL 108, we do not provide any examples here. For the rare case that there is a large output request, the reader can refer to Section 3.4 and Section 4.4 on output request performance for Linear Statics and Normal Modes and in addition to Section 6.4 for Modal Frequency Response.

Additional Topics

Krylov Iterative Method

The Krylov iterative method is an option often tested by users for SOL 108 or SOL 111. It is turned on via the following Executive System Parameters:

```
NASTRAN KRYLOV1=-1
```

To get the default options there are several other options that a user can find in the Quick Reference Guide. The Krylov method will be described in greater detail in Chapter 6 but for this chapter we merely mention that it uses Krylov subspaces and a preconditioner to reduce the overall wall clock. However, **we do not recommend it to be used for SOL 108 except in the case of an exterior acoustics only model.** In particular, for the case of a body-in-white model with structural coupled to interior acoustics (and potentially also

exterior acoustics), the method will likely have convergence issues. It is best reserved for SOL 111 to solve the exterior acoustics part of the weakly coupled model, i.e., the ACOWEAK option.

Matrix Iterative Method

The matrix iterative method can be defined in Case Control via

```
SMETHOD=matrix
```

or with greater control via options defined in the Quick Reference Guide. It is not a highly recommended option today and in fact for Model 4 showed significantly worse performance than PRDLU with the added possibility of convergence or accuracy issues. Thus, again, we do not recommend it today as an option for solving the matrix problem in SOL 108. In the future, we may introduce new matrix iterative methods with better performance and convergence.

Model and Hardware Details

The following models are used in this section to illustrate our points about obtaining best performance in a direct frequency response run. None of these are available to share with users but we will consider new models in the future that are shareable.

Table 5-1 Summary of Model Types for Performance Comparison

Model #	Description	DOFs	2D Elements	3D Elements	NFREQ
1	Large Shell Element Model with Small Output Request	8266806	1376256	0	1
2	Large Exterior Acoustics Model with Small Output Request	6020862	66801	4563707	1
3	Large Solid Element Model with Large Output Request	17204022	31760	1526072	11
4	Medium Engine Block with Exterior Acoustics	2801772	42464	397680	9

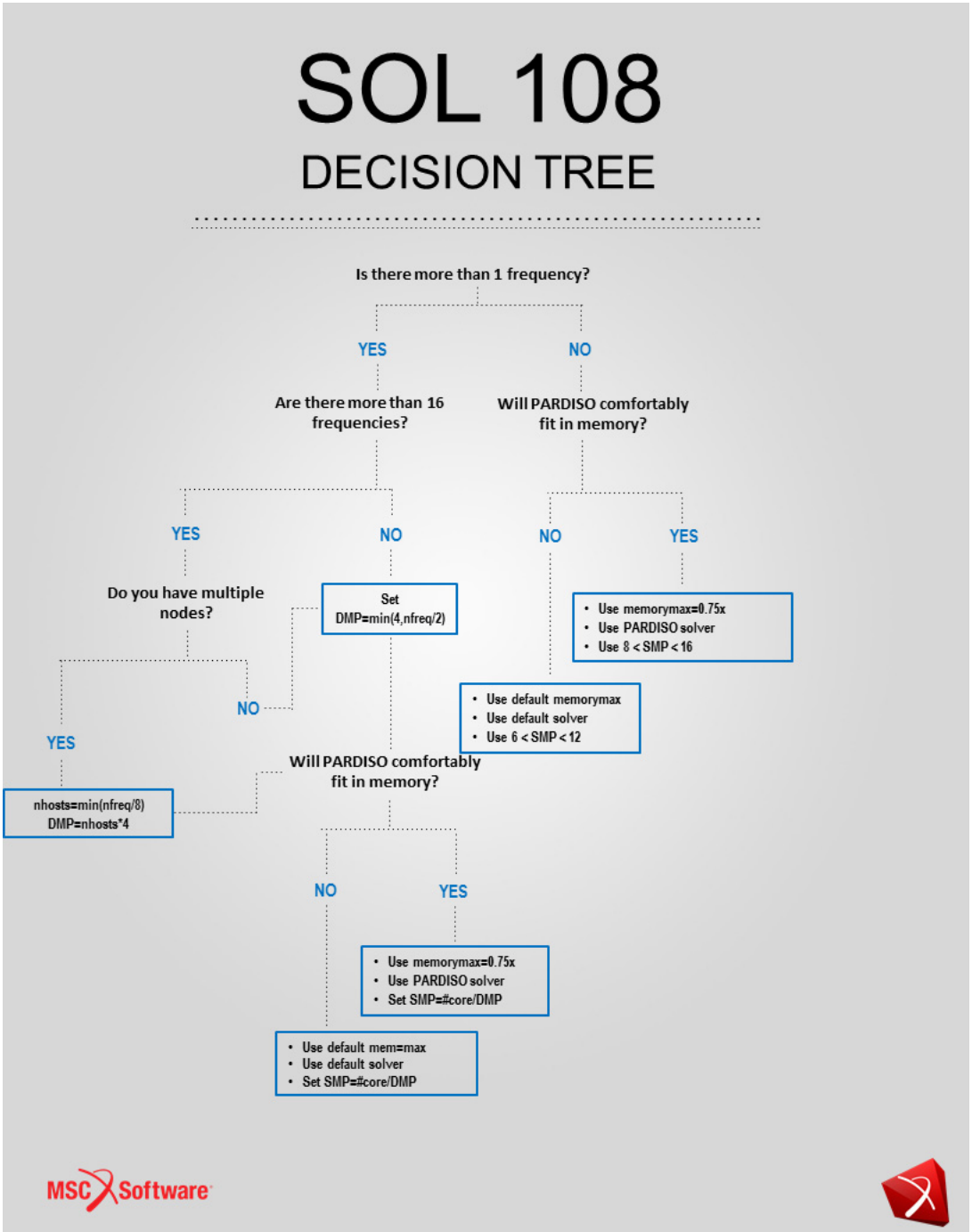
Note that many of the 3D elements correspond to fluid elements for the cavity for Model 2 and 4.

These machines are a large HPC Machine, a Medium Memory Machine, and a Small Memory Machine. Details follow:

Table 5-2 Summary of Machine Types for Performance Comparison

Memory	Memory	CPUs	HDD	OS
HPC Machine	256 GB	2x10 Intel® Xeon® CPU E5-2550 v3 @2.60GHz	4 x 600 GB 15K RPM in RAID0	RH 7.1
Medium Memory Machine	128 GB	2x8 Intel(R) Xeon(R) CPU E5-2650 v2 @2.60GHz	1 x 550 GB 10K RPM Hard Drive	SuSE SP2
Small Memory Machine	32 GB	2x10 Intel® CPU @2.8GHz	1 x 1 TB 7200 RPM Hard Drive	RH 6.3

Decision Tree for Direct Frequency Response Analysis



6 Optimal Performance of Modal Frequency Response Analysis (SOL 111)

- Introduction 96
- Memory Usage 96
- Solver Options 100
- Parallel Options 103
- Output Request 104
- Additional Topics 106
- Definition of Hardware/Models used in Examples 107
- Decision Tree for Modal Frequency Analysis 108

Main Index

Introduction

In MSC Nastran, modal frequency response analysis is an alternative to the direct approach of Chapter 5 to computing the frequency response of a structure. The modal approach uses the mode shapes of the structure to reduce the problem size and make the overall simulation more efficient. This chapter focuses on optimizing performance for the most common applications of SOL 111. For more information on the modal frequency analysis approach, refer to Chapter 4 of the MSC Nastran Dynamic Analysis User's Guide.

Since the mode shapes are typically computed as part of the structure characteristics, modal frequency response is a natural extension of normal modes analysis. If all modes are used in a modal frequency response analysis, the results will be exact. However, it is not necessary to compute all of the modes in order to provide sufficient accuracy in the frequency range of interest. Furthermore, computing the normal modes of a structure, for a given frequency range, is but one way to produce a basis for an ensuing frequency response analysis.

One popular technique for producing a good approximate modal basis is to compute a component modal synthesis on a collection of automatically generated sub-structures, or components. In MSC Nastran, this technique is known as Automatic Component Modal Synthesis (ACMS). As stated in other parts of this manual, there is an Accelerated ACMS method introduced in 2016.0 (and often referred to as the New ACMS method) and the Original ACMS method that has been in MSC Nastran since 2004. These were described in Chapter 4 and further results comparing the two will be presented here. In the future, the Original ACMS method will be deprecated.

This chapter adds information relevant to SOL 111 to material presented in Chapter 4 of this manual on Normal Modes Analysis. In addition to the modal solution portion of a modal frequency response analysis, there are other analysis requirements that should be considered when pursuing optimal performance:

- The nature of the frequency response analysis (uncoupled, coupled, unsymmetric effects)
- Presence of an interior fluid model (interior acoustic analysis)
- Presence of infinite elements (exterior acoustic analysis)
- Special analysis situations

These are in addition to typical analysis considerations:

- Size and composition of the structural model (plate dominated vs. solid element dominated)
- Output requirements

Material in this chapter is used to develop a decision tree for specifying memory and parallel options, presented in Section 6.7.

Memory Usage

As in any MSC Nastran solution, optimal performance is achieved by minimizing disk I/O, and by providing adequate memory for numeric calculations. Memory must be reserved for caching I/O operations, both by MSC Nastran internally, and by the underlying operating system. The principal considerations for specifying memory for MSC Nastran in SOL 111 are:

- The method for computing the modal subspace (Lanczos, Accelerated ACMS, or Original ACMS).
- The size of the model

- The output request

The use of the “memorymax” keyword is recommended so that optimal settings are used for MSC Nastran internal I/O caching operations. In this way, the MSC Nastran Buffer Pool length is set automatically. Also, note that keeping memorymax below 75% of machine physical memory leaves enough memory for the OS to cache I/O.

In the remainder, we discuss best memory practices using Lanczos, Original ACMS, and Accelerated ACMS. We start the discussion for the Lanczos solver but we note that later we will see the significant performance advantage using the ACMS methods.

SOL 111 using Lanczos

If Accelerated ACMS or Original ACMS are not used, there is likely to be a significant disk I/O requirement for the Lanczos method to compute the modal subspace. In this case, it is recommend to limit memory to half of the real memory on the host computer. In this way, memory is reserved for the underlying operating system to use unsubscribed memory to buffer the physical I/O requests.

Subsequent to the modal solution, MSC Nastran memory requirements are fairly modest during typical stages of the modal frequency response analysis; these include computing reduced quantities for damping and acoustic coupling, reduced loads computation, and fluid model solution (for interior acoustics). Memory requirements for the actual frequency response computations are also quite modest, due to having reduced the problem size to modal coordinates.

After the responses are computed in modal coordinates, they must be transformed back to physical space. For these operations, the size of the output request is important. If ALL is requested for any output quantity, or if the size of the selected SET is significant, memory requirements are fairly significant in order to carry out the required operations. In addition, a significant disk I/O requirement may be incurred. For outputs at only a few degrees of freedom from a small collection of grid points, MSC Nastran will sense the “sparse” output request, reducing memory and disk I/O substantially.

Suggested memory amounts are presented in the Table 6-1.

Table 6-1 Memory Length Suggested for SOL 111 with Lanczos

Table 6-2

Model Size (DOFs)	Output Request	Suggested MEMORYMAX=
Up to 10 Million	sparse	max(16GB, 0.5xphysical)
	full	max(32GB, 0.5xphysical)
10 - 20 Million	sparse	max(48GB, 0.5xphysical)
	full	max(64GB, 0.5xphysical)
20 - 40 Million	sparse	max(128GB, 0.5xphysical)
	full	max(200GB, 0.5xphysical)

Note that “0.5xPhysical” is a valid specification for the MEMORYMAX keyword (e.g. “memorymax=0.5xPhysical”). It says to allocate 50% of the computer’s memory to the MSC Nastran job.

SOL 111 using Original ACMS

When using Original ACMS in SOL 111, the memory requirements are about the same as those for Lanczos. In terms of resource requirements, the principal difference between Original ACMS and Lanczos is that the Original ACMS solution will require far less memory and disk I/O to compute the modal subspace. Subsequent to modal subspace computation, SOL 111 operations are similar for the different solution strategies. Suggested memory amounts are presented in Table 6-3.

Table 6-3 Memory Length Suggested for SOL 111 with Original ACMS

Table 6-4

Model Size (DOF)	Output Request	Suggested MEMORYMAX=
Up to 10 Million	sparse	max(16GB, 0.5xphysical)
	full	max(32GB, 0.5xphysical)
10 - 20 Million	sparse	max(48GB, 0.5xphysical)
	full	max(64GB, 0.5xphysical)
20 - 40 Million	sparse	max(128GB, 0.5xphysical)
	full	max(200GB, 0.5xphysical)

The use of Distributed Memory Parallel processing (DMP) is highly recommended when using SOL 111 with Original ACMS. Note that when using memorymax to specify memory for MSC Nastran, the amount of memory allocated by each DMP process is memorymax divided by NDMP, where NDMP is the number of DMP processes specified. For example, specifying “memorymax=200gb dmp=4” results in each DMP process allocating 50 GB of memory. See Section 6.2 for more information on specifying parallelism for SOL 111.

SOL 111 using Accelerated ACMS

The memory requirements for Accelerated ACMS in SOL 111 are higher in the modal calculation part than for Original ACMS but are the same for the remaining part of the solution sequence. Suggested memory amounts are presented in Table 6-5.

Table 6-5 Memory Length Suggested for SOL 111 with Accelerated ACMS

Table 6-6

Model Size (DOF)	Output Request	Suggested MEMORYMAX=
Up to 10 Million	sparse	max(32GB, 0.5xphysical)
	full	max(48GB, 0.5xphysical)
10 - 20 Million	sparse	max(48GB, 0.5xphysical)
	full	max(64GB, 0.5xphysical)
20 - 40 Million	sparse	max(128GB, 0.5xphysical)
	full	max(200GB, 0.5xphysical)

Just as with Original ACMS, the use of Distributed Memory Parallel processing (DMP) is highly recommended when using SOL 111 with Accelerated ACMS. For Accelerated ACMS, memory is divided by the DMP processes in an unequal fashion: the modal reduction module (ACMS1) is executed on the Master DMP process only, using all available cores. Since memory requirements are relatively high for the ACMS1 module, default memory allocation when using memorymax provides that 80% of the specified memory will be allocated to the Master DMP process. The remaining 20% of specified memory will be evenly divided among the N-1 “slave” DMP processes.

Example memory allocations are listed in Table 6-7.

Table 6-7 SOL 111 memory allocation for Accelerated ACMS using DMP

Table 6-8

MEMORYMAX=	DMP=	Memory Allocation			
		DMP Process 1 (Master)	Process 2	Process 3	Process 4
64 GB	2	51 GB	13 GB		
128 GB	2	102 GB	26 GB		
64 GB	4	51 GB	4 GB	4 GB	4 GB
128 GB	4	102 GB	8 GB	8 GB	8 GB
200 GB	4	160 GB	13 GB	13 GB	13 GB

When running SOL 111 with Accelerated ACMS in master-slave mode (i.e. single host execution), memory requirements for Slave DMP processes are modest. The percentage of memory allocated to the Master DMP process is controlled by the “dmpmem” command line keyword. The default value for “dmpmem” is 80, or 80% of “val” where “memorymax=val”.

Solver Options

In SOL 111, we have to both find the modes of the system using an eigensolver like ACMS as well as solver the frequency response problem for the modal system. Furthermore, if the problem has exterior acoustics, then we also have to solve the exterior acoustics part in physical space. Consequently, we divide the rest of the section into EigenSolver Options and Frequency Response Solver Options.

EigenSolver Options

Due to the dramatic performance advantages of the Original and Accelerated ACMS methods, Lanczos is not recommended for MSC Nastran SOL 111 for large (approximately 8 million DOFs or more) models as an eigensolver. Using the Lanczos method for the class of problem under discussion would result in jobs running for several days or more. For a model with approximately 7 million DOFs (Model 4), we see the performance differences in Figure 6-1 below.

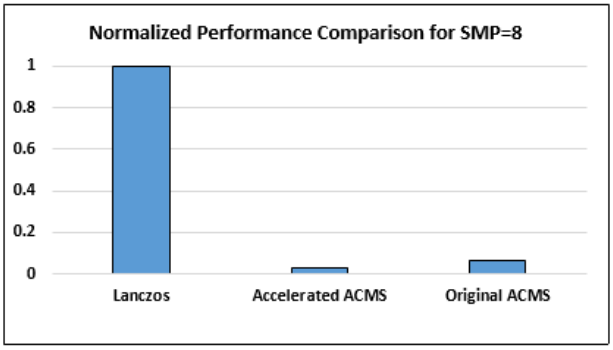


Figure 6-1 Performance Comparison of Lanczos, Accelerated ACMS, and Original ACMS for Model 4

For this model, the performance difference is hours versus minutes. For larger models, it can be days versus hours.

The Accelerated ACMS method is the recommended ACMS method. The Original ACMS method should only be used when there is insufficient memory to use the Accelerated version. Results in Figure 6-2 illustrate the performance difference that one can observe using SMP=8 for three different models

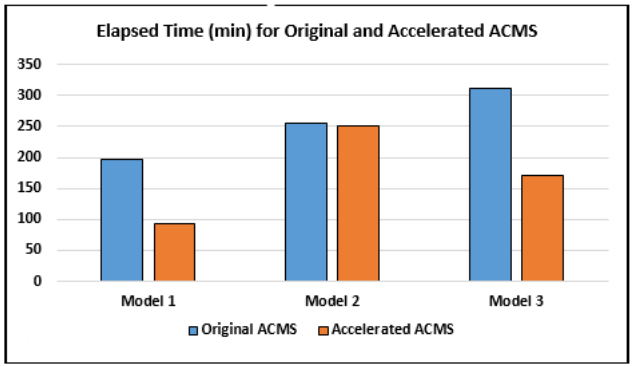


Figure 6-2 Performance comparison of Original and Accelerated ACMS methods. See Table 6 for model details

Frequency Response Solver Options

With the modal frequency response approach, the actual frequency response problem is computed using the modal subspace resulting from a global eigensolution, or some other modal basis such as ACMS. The general idea is that the problem size is reduced by several orders of magnitude and so is more efficient. With the advent of reduction methods such as ACMS, users are taking advantage of the resulting increase in compute capability, and frequency range requirements are growing steadily. As a result, the reduced problem size can itself become a computational challenge. For this reason, alternate algorithms may be used in place of the simple series of matrix solutions (one for each desired forcing frequency), e.g., the Fast Frequency Response method, or FASTFR, that we describe next.

FASTFR

The FASTFR solution is selected automatically when it is appropriate. The criteria for selection include the reduced problem size, and the rank of the structural damping ("K4") matrix. If the problem size (i.e. the number of mode shapes in the modal subspace) is small, there is not likely to be much benefit in choosing FASTFR over the default series of matrix solutions. Determining the rank of the K4 matrix is a means for determining the level of coupling present in the overall dynamic matrices. If the amount of coupling is high, FASTFR is not optimal and will be de-selected. However, most common modeling scenarios result in a relatively low level of coupling, which is then exploited by the FASTFR solution process. The user can predict the amount of coupling by examining the structural damping application in the model. If a single PARAM,G entry is present, or if a small number of well-defined GE entries are specified via material bulk data entries, then the resulting coupling is probably minor and thus, FASTFR will be applicable.

The FASTFR solution is possible for both symmetric and unsymmetric matrix solutions.

The overall FASTFR method is set using bulk data parameter PARAM,FASTFR. The default is PARAM,FASTFR,AUTO. The valid values for FASTFR are AUTO, YES, and NO. The problem size threshold for FASTFR is controlled via PARAM,FFRHMAX (default=2500).

For the effective execution, the iterative solver might be turned off when the problem size is too small under the default FASTFR=auto. To force iterative solver execution under FASTFR, PARAM, FASTFR, YES or small problem threshold PARAM,FFRHMAX are required.

FASTFR is not available for the following cases:

- Frequency dependent solution
- Unsymmetric acoustic solution
- The model has Extra Points (except exterior acoustic modeling)

User Information Message 5222 is printed in the F06 output file, signifying the use of FASTFR. The REAL Fast Frequency Response solution is used for symmetric matrices, and COMPLEX is used for unsymmetric matrices.

User Information Message 9157 is printed in the F06 output file if the FASTFR solution is not selected, including the reason why it was not selected.

Note that in some cases, excluding exterior acoustics described below, it can be advantageous in the frequency response solve to use the Pardiso Solver, i.e., add the following to the Executive Control Section:

```
SPRARSESOLVER FRRD1(FACTMETH=PRDLU)
```

Exterior acoustics and Weakly Coupled (interior/exterior) acoustics

Weakly coupled acoustics is an increasingly common solution technique in the automotive industry that is an alternative to fully coupled interior and exterior acoustics. This solution features analysis of both interior and exterior acoustic response, including interaction or “coupling” effects between the two. This solution is activated by the presence of PARAM, ACOWEAK, YES in the bulk data input section. For such an analysis, the exterior solution is performed in a direct approach for the acoustic model. The matrices used for the exterior acoustic model are unsymmetric, which adds a degree of complexity to its solution.

In addition to the default MSCLU solver, MSC Nastran features the Pardiso Solver, as well as an iterative method known as the “Krylov” solver. In most cases, the Krylov solver proves most efficient for this class of problem. For weakly coupled acoustics (PARAM, ACOWEAK, YES), the Krylov solver may be specified by including the following in the bulk data input section.

```
PARAM, ACEXTMTD, KRYLOV
```

Note that the Krylov solver requires a special license to run, and many users may not have this. Thus, if a user does not have the Krylov Solver license, then it is recommended to use the Pardiso Solver for exterior acoustics. We refer the reader to Chapter 5 on Direct Frequency Response for more details and remind the reader that the Pardiso Solver is enabled in the Executive Control Section via

```
SPRARSESOLVER FRRD1(FACTMETH=PRDLU)
```

This is particularly the case for frequency dependent material where FASTFR is not applicable.

The default MSCLU solver, as observed in Chapter 5, has poor performance for many cases and thus the default will soon be changed to the Pardiso Solver.

Parallel Options

The use of multiple cores is required for optimal performance in SOL 111. Accelerated ACMS, Original ACMS and Lanczos all support Shared Memory Parallel (SMP) and Distributed Memory Parallel (DMP). Both SMP and DMP must be used for best performance.

SMP may be specified either on the command line as “smp=n” or in the input file:

```
NASTRAN SMP=n
```

DMP may be specified either on the command line as “dmp=n” or (starting with MSC Nastran 2018 for Linux only) in the input file:

```
NASTRAN DMP=n
```

Accelerated ACMS and Original ACMS

When using Accelerated ACMS or Original ACMS in MSC Nastran SOL 111, four distinct stages typically dominate the run time. While we have described hem previously, we want to consider them in the context of parallel performance. Using multiple cores is highly recommended in order to reduce the time required in these stages. These stages are:

1. Modal synthesis using automatically generated components
2. Computation of system modes
3. Computation of reduced damping and acoustic coupling matrices (if applicable)
4. Frequency response solution

Note that stages (2) and (4) are identical when running Accelerated ACMS or Original ACMS. Stages (1) and (3) operate differently: Original ACMS uses a combination of DMP and SMP for parallel performance; Accelerated ACMS uses SMP exclusively for these stages. However, these differences are transparent to the user, so that identical parallel settings should be used for both ACMS versions. Guidelines are presented in Table 6-9.

Table 6-9 SOL 111 parallel guidelines for Accelerated ACMS and Original ACMS

Table 6-10

No. of Cores Available	DMP	SMP
2	2	
4	4	
8	4	2
16	4	4
>16	(Ncore/4)	4

The table presented in Figure 1 shows normalized speedup that is a result of various parallel options for a large SOL 111 interior acoustic analysis of a fully trimmed automobile model using Accelerated ACMS.

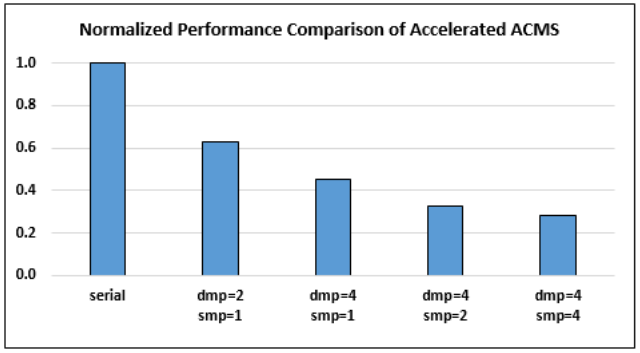


Figure 6-3 Normalized parallel speedup for large interior acoustic automobile analysis (Model 3). All jobs were run with Accelerated ACMS.

Lanczos

As stated previously, Lanczos is not recommended for MSC Nastran SOL 111 for large (>8 million DOFs) models. However, if Lanzcos is desired, the recommendation is to set DMP to 4 and SMP to 4 (maximum) for a total of 16 cores. The Lanczos solution will not scale beyond these settings. Figure 6-4 shows SMP scalability for a job run with Lanczos.

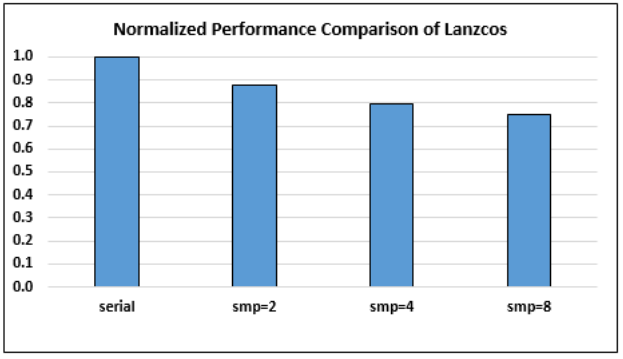


Figure 6-4 Normalized parallel performance for a moderate size model (Model 4) using Lanzcos. See Table 6 for model details.

Output Request

For most use cases, the output request for a modal frequency response analysis consists of grid point oriented quantities (displacement, velocity, and acceleration) for only a few points. This type of output request is known as “sparse” and the data recovery operations required are tuned for the “sparse data recovery” solution path. Below is a description of various controlling parameters for data recovery. Generally, changing the default values for these options is not recommended.

If there is a large SET of grid points required, or if ALL is specified for any output quantity, this “full” output request is processed in the most efficient manner possible. Due to the nature of any full data recovery request, there are no special parameters.

MDOTM

The MDOTM parameter is a character parameter (default=’AUTO’) which indicates whether or not the OTM method will be used for data recovery. (OTM stands for Output Transformation Matrix.) The OTM method is available for the ACMS methods only. By default, MSC Nastran determines the suitability of the OTM approach by examining the number of eigenvalues and the number of points in the output request. Generally, the OTM method is optimal for sparse output requests, and will be chosen accordingly. The OTM method only those parts of the eigenvector matrix that are needed for data recovery by utilizing the ACMS reduction tree.

The OTM method is automatically set to NO if the subsequent data recovery operations truly require the recovery of the full eigenvector matrix. This may be true even if the particular output request appears to be sparse. Examples where MDOTM is automatically set to NO include the use of Modal Effective Mass, and the use of the MODESELECT case control command. For more information about PARAM,MDOTM see the MSC Nastran Quick Reference Guide.

Setting PARAM,MDOTM to anything other than AUTO (the default) is not recommended.

SPARSEPH

In the event an eigenvector matrix will be computed, the SPARSEPH parameter controls whether or not all rows of each eigenvector will be computed. By default (SPARSEPH=YES), MSC Nastran creates a partition vector corresponding to the desired DOFs to be output. For sparse data recovery, use of SPARSEPH=YES results in a reduced amount of work required to produce the output DOFs. If appropriate however, the partition vector is set to full, resulting in all rows being computed.

For more information about PARAM,SPARSEPH see the MSC Nastran Quick Reference Guide. Setting PARAM,SPARSEPH to anything other than YES (the default) is not recommended.

SPARSEDR

The SPARSEDR parameter limits data recovery operations to only those grid point on SET commands in the Case Control input section. For more information about PARAM,SPARSEDR see the MSC Nastran Quick Reference Guide. Setting PARAM,SPARSEDR to anything other than YES (the default) is not recommended.

SPEQEXIN

The SPEQEXIN parameter is used to partition only those DOFs that are required from the existing output request. Setting PARAM,SPEQEXIN to anything other than YES (the default) is not recommended.

Additional Topics

Below we provide a discussion on a few other relevant topics, we also suggest the reader refer to Chapter 8 on general guidelines and some additional FAQs on the ACMS method.

Fluid Eigensolution in 111 (DMP)

In MSC Nastran fluid-structure interaction, the fluid is modeled separately from the structure. In most interesting cases, the fluid model is typically small compared to the structure. Analysis requirements for most fluid-structure applications are such that a fairly coarse mesh is sufficient. For these reasons, the fluid eigensolution in the past runs in serial execution mode using the Lanczos eigenvalue extraction method, extracting fluid eigenvalues up to the frequency maximum set by the user.

Contemporary analysis requirements have made the fluid eigensolution a more significant computational task. Namely, the typical maximum frequency for the fluid eigenvalues is now commonly 1000 Hertz to 1500 Hertz for vibro-acoustic analysis of certain automotive model scenarios. For this reason, the fluid eigensolution in SOL 111 is now automatically executed with Distributed Memory Parallelism (DMP). The number of DMP processes is taken from the user's input specification of the number of DMP processes. The fluid eigenvalue frequency range is split according to the MSC Nastran Frequency Domain Parallel domain decomposition technique. This technique is known to be both straightforward and robust. For typical fluid models seen in automotive modeling, parallel speedup is nearly linear with the number of DMP processes used. Since a full Lanczos eigensolution is performed, there is no loss of accuracy compared to previous MSC Nastran versions.

MDK4OPT

Modeling capabilities for damping elements and materials have become quite complex. The specification of structural damping in particular has significant performance implications. The reason for this is that the way structural damping is specified in contemporary modeling techniques contributes to the degree of "coupling" present in the dynamic equations solved for frequency response analysis. Sophisticated solution algorithms used in MSC Nastran can exploit "loosely" coupled systems, resulting in a significant efficiency improvement compared to the solution of a "fully" coupled system. (This is the basis for the FASTFR algorithms.)

Depending on the engineering application, structural damping specifications may be simple or complicated. For the simplest case, a single damping value is input; for slightly more complicated use, a small number of distinct structural damping values are applied. In many such cases, the overall structural damping effect is dominated by a single damping coefficient; in effect, much of the resulting K4 structural damping matrix contains "redundant" information. It is possible to simplify the K4 matrix in this scenario by removing the "dominant" K4 value from the individual K4 matrix entries, to be considered subsequently as a scale factor. In this way, matrix solutions involving K4 structural damping is simplified and the degree of coupling reduced, with no loss of accuracy.

In MSC Nastran, this K4 matrix simplification is controlled by bulk data parameter MDK4OPT. When MDK4OPT=1 (default), the K4 entries are examined and modification is possible under special circumstances. K4 matrix modification is not possible in several cases, including multiple superelements, the presence of C-set DOFs, PARAM,SHLDAMP, and several other cases. Users may de-select K4 matrix simplification by setting MDK4OPT to -1 (not recommended).

DMP parallelization of PFCALC module

For any nontrivial frequency response analysis, time required for the calculation of modal participation factors (PFMODE, PFGRID, PFPANEL) can easily dwarf all other calculation time for a particular analysis. The PFCALC module, where these computations take place, is enabled for DMP parallelization over the frequency range of interest. Specifically, the set of excitation frequencies is divided among DMP processes in a naïve frequency domain decomposition identical to frequency domain parallelization in other parts of MSC Nastran. A distinct domain decomposition and solution is done for each load case, so that parallel load balance is maintained even if the frequency list might change from one load case to another. Note that support for PEAKOUT is not available at this time.

Definition of Hardware/Models used in Examples

The following models are used in this section to illustrate our points about obtaining best performance in a modal frequency response analysis. None of these are available to share with users but we will consider new models in the future that are shareable.

Table 6-11 Summary of Models for Performance Comparison

Table 6-12

Model#	Description	DOFs (G-size)	No. of Structure modes	No. of Fluid modes	NFREQ
1	Automotive interior acoustic analysis	33 million	12,900	1,570	500
2	Coupled interior/exterior acoustic analysis (ACOWEAK)	23 million	13,365	2,009	500
3	Automotive interior acoustic analysis	44 million	12,300	1,700	500
4	Automotive body-in-white analysis	7 million	3,700	n/a	320

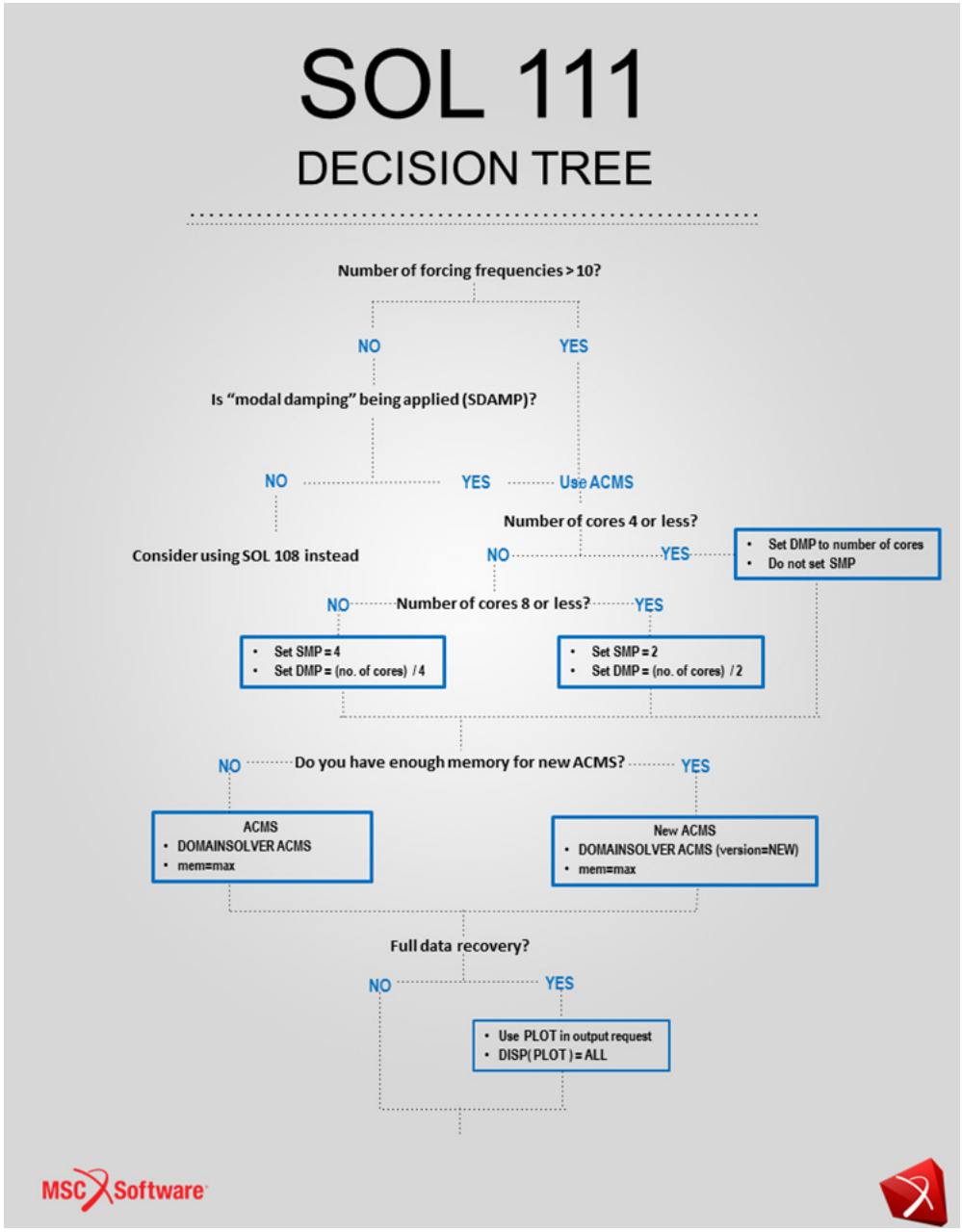
These jobs were tested on our HPC Machine with details below:

Table 6-13 HPC Machine Details

Table 6-14

Machine	Memory	CPUs	HDD	OS
HPC Machine	256 GB	2x10 Intel® Xeon® CPU E5-2550 v3 @2.60GHz	4 x 600 GB 15K RPM in RAID0	RH 7.1

Decision Tree for Modal Frequency Analysis



Main Index

Main Index

7 Optimal Performance of Nonlinear Analysis (SOL 400)

- Introduction 111
- Memory Usage 111
- Matrix Solver Options 114
- Parallel Options 120
- Output Request 126
- Additional Topics 128
- Model and Hardware Details 130
- Decision Tree for Nonlinear Static Analysis 131

Introduction

In MSC Nastran, nonlinear analysis typically involves the solution of a nonlinear structural analysis problem with the goal of understanding the displacements, and other result quantities like element stresses, in response to loads, contact, and nonlinear material laws. Behind the scene, MSC Nastran solves a nonlinear problem, R(x)=0, using a nonlinear solution algorithm that needs many iterations to be converged.

The wall clock time of the corresponding simulation is determined by the following:

- Number of iterations to achieve convergence
- Calculating nonlinear element matrices and stresses
- Solving the matrix problem
- Writing the requested data to an output file

SOL 400 is actually a very rich solution sequence in that you can do a wide variety of analyses defined by ANALYSIS=*XTYPE* where *XTYPE*=NLSTAT, NLTRAN, MODES, MFREQ and DFREQ. We focus the discussion in this chapter on the case of NLSTAT (i.e., nonlinear static analysis). See the MSC Nastran Nonlinear User's Guide for details about the other analyses types.

We also refer the reader to the Nonlinear User's Guide for information on settings in the nonlinear solver that can affect the number of iterations to achieve convergence. The goal, of course, is to minimize the number of iterations, but to still maintain accuracy. We do not discuss the details needed to address the first bullet point, but leave it to the reader to refer to the Nonlinear User's Guide. We focus on the last three bullet points in this chapter.

In "Memory Usage" on page 111, we describe in detail the options for solving the matrix problem, but we mention them here first for references purposes. We consider the two sparse direct solvers and an iterative solver:

- MSC Sparse Direct Solver (MSCLDL)
- Pardiso Sparse Direct Solver (PRDLDL)
- CASI Iterative Solver (CASI)

We use a series of models and hardware platforms as described in "Model and Hardware Details" on page 130 to illustrate how a user can get best performance based on memory settings, method selection, and parallel settings. Also, in "Decision Tree for Nonlinear Static Analysis" on page 131, we provide a decision tree that allows the user to make a choice on the various settings.

Memory Usage

Understanding Memory

The memory requirements for SOL 400 are very similar to SOL 101, and in fact, both use the estimate program to determine the amount of memory required to keep the solver in-core. The reader may want to read "Memory Usage" on page 38 in Chapter 3 for additional information on memory usage. If a user selects memorymax according to "memorymax=16gb" and uses "mem=max", they would see the following memory breakdown in the F04 file:

```
      USER OPENCORE (HICORE)          =    77132272  WORDS
      EXECUTIVE SYSTEM WORK AREA      =      940215  WORDS
      MASTER(RAM)                     =      200000  WORDS
      SCRATCH(MEM) AREA               =      819300  WORDS (100 BUFFERS)
      BUFFER POOL AREA (BPOOL4)       =  2068191862  WORDS (252434 BUFFERS)

      TOTAL MSC NASTRAN MEMORY LIMIT =  2147483648  WORDS
```

Since MSC Nastran uses only 64-bit words, this implies total MSC Nastran memory is 16 GB and the solver (HICORE) gets 0.57 GB and the rest of memory is left for buffer pool (BPOOL4) to cache I/O[1]. Scratch mem is not recommended today and gets the default of 100 buffers.

Just like in SOL 101, one of the main requirements in SOL 400 to get optimal performance is to ensure that the direct method or the iterative method fits in-core. The MSC Sparse Direct Solver, the Pardiso Sparse Direct Solver, and the CASI Solver have different requirements to achieve this in-core characteristic. Below, we show how to examine a preexisting run to determine if optimal memory choices were made and then how to adjust the memory choices for a future run.

For the MSC Sparse Direct Solver, we can see the details of the memory usage in the F04 file as follows:

```
*** USER INFORMATION MESSAGE 4157 (DFMSYM)
     PARAMETERS FOR PARALLEL SPARSE DECOMPOSITION OF DATA BLOCK KLL     (TYPE=RSP) FOLLOW
                      MATRIX SIZE =    685566 ROWS            NUMBER OF NONZEROES =  19553805 TERMS
            NUMBER OF ZERO COLUMNS =         0        NUMBER OF ZERO DIAGONAL TERMS =         0
                    SYSTEM (107) =     32776                     REQUESTED PROC. =         8 CPUS
            ELIMINATION TREE DEPTH =     17772
                 CPU TIME ESTIMATE =       650 SEC               I/O TIME ESTIMATE =        10 SEC
       MINIMUM MEMORY REQUIREMENT =     15289 K WORDS            MEMORY AVAILABLE =    278405 K WORDS
     MEMORY REQR'D TO AVOID SPILL =     36688 K WORDS         MEMORY USED BY BEND  =     15290 K WORDS
     EST. INTEGER WORDS IN FACTOR =    117645 K WORDS           EST. NONZERO TERMS =    239758 K TERMS
     ESTIMATED MAXIMUM FRONT SIZE =      4572 TERMS                RANK OF UPDATE =        64

*** SYSTEM INFORMATION MESSAGE 4199 (PREFAC1)
    THE ENTIRE SPARSE FACTOR HAS BEEN CACHED IN MEMORY FOR THE FBS.
    ALL 19991 FRONTAL MATRICES ARE STORED IN MEMORY.
    MEMORY AVAILABLE: 256 M WORDS
    MEMORY USED TO STORE THE ENTIRE FACTOR: 479 M WORDS
```

As can be observed for this case, the entire sparse factor and its frontal matrices fit in memory. This should always be the target for the default solver and is generally easy to achieve due to the minimal memory usage of the MSC Sparse Direct Solver.

[1]Note that in a future release MSC Nastran will use MB, GB, etc instead of WORDS.

For the Pardiso Sparse Direct Solver, we get the following memory usage details in the F04 file:

```
*** USER WARNING MESSAGE 4157 (PardisoSolver::checkMemoryNeeds)
     MEMORY PARAMETERS FOR INTEL MKL PARDISO PARALLEL DECOMPOSITION FOLLOW
         AVAIL. CORE MEMORY =24322 MB
         REQUI. IN-CORE MEMORY =59411 MB                 REQUI. OUT-OF-CORE MEMORY =10101 MB
```

The user would get better performance by increasing the amount of memory available to the solver by approximately 40 GB such that the available core memory for PRDLDL is greater than 59411 MB.

For the CASI iterative solver, the amount of memory needed by CASI can be found in the F04 file with the following system information message:

```
*** SYSTEM INFORMATION MESSAGE 3008 (CAPMEM)
     MEMORY AVAILABLE TO NASTRAN IS SUFFICIENT FOR CASI ITERATIVE SOLVER. TOTAL HIWATER MARK FOR NASTRAN
     WITH CASI CAN BE CALCULATED BY ADDING THE BELOW MAXIMUM ESTIMATED MEMORY NEEDED TO NASTRAN HIWATER
                            NASTRAN MEMORY AVAILABLE = 58568MB,
                            MINIMUM ESTIMATED MEMORY NEEDED FOR CASI ITERATIVE SOLVER = 5358MB,
                            MAXIMUM ESTIMATED MEMORY NEEDED FOR CASI ITERATIVE SOLVER = 8126MB
```

Examples

Now, we will look at an example of a solid-element model to give the user an idea of the impact of memory amounts on performance. As with other chapters, we start with results on the Small Memory Machine.

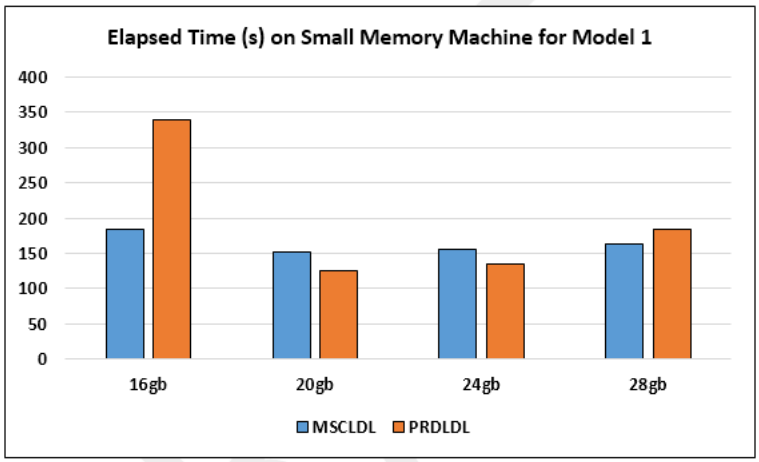


Figure 7-1 Elapsed time on Small Memory Machine for Model 1 with SMP=8

In the above figure, we have ran both the Pardiso Solver and the MSC Sparse Direct Solver on a machine with 32 GB of RAM and then set different memorymax amounts. At “memorymax=16gb”, we have forced the Pardiso solver to go out-of-core so Pardiso requires I/O operations.This indicates the need to keep Pardiso in-core. If we look at the 16 GB case, and in particular, at the F04 file, we will see the following two memory statements:

```
** MASTER DIRECTORIES ARE LOADED IN MEMORY.
   USER OPENCORE (HICORE)          =  1608444937  WORDS
   EXECUTIVE SYSTEM WORK AREA      =      940215  WORDS
   MASTER(RAM)                     =      200000  WORDS
   SCRATCH(MEM) AREA               =      819300  WORDS (     100 BUFFERS)
   BUFFER POOL AREA (BPOOL4)       =   536879197  WORDS (   65529 BUFFERS)

   TOTAL MSC NASTRAN MEMORY LIMIT =  2147483648  WORDS
```

and

```
MEMORY PARAMETERS FOR INTEL MKL PARDISO PARALLEL DECOMPOSITION FOLLOW
                    AVAIL. CORE MEMORY =8307 MB
          APPROX. REQUI. IN-CORE MEMORY =9984 MB
      APPROX. REQUI. OUT-OF-CORE MEMORY =3991 MB
```

This indicates that if we can give extra memory to Pardiso so that it runs in-core then we should. Thus, we reran with "memorymax=16gb bpool=1gb" (an extra 3 GB to Open Core by taking away 3 GB from Buffer Pool[1]) and we reduced the wall clock time by 20%. At "memorymax=28gb", we also see that both solvers have a performance degradation due to approaching the Physical RAM amount of the machine. As stated in previous chapters, this is discouraged to go beyond 75% of Physical RAM for memorymax.

In the next figure, we perform the same test but on the Medium Memory Machine.

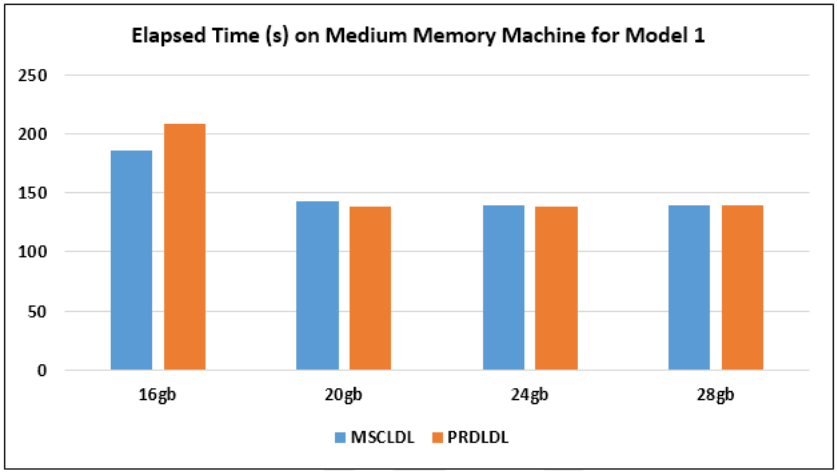


Figure 7-2 Elapsed time on Medium Memory Machine for Model 1 with SMP=8

The results in Figure 7-2 illustrate that the out-of-core behavior ("memorymax=16gb") is different on different hardware. Here the penalty from Pardiso going out-of-core is not as severe. There are two reasons. The first reason is that this is a machine with a more modern Intel CPU – Intel Sandy Bridge, whereas the Small Memory Machine has an older Intel CPU – Intel Westmere. The Intel team optimizes the current versions of their solvers on the newest hardware. The second reason is that this machine has 128 GB of RAM and so potentially more for Linux I/O caching.

Matrix Solver Options

The method selection involves specifying the approach to solving the matrix problem that returns the displacements given the applied loads. In linear algebra terminology, we want to select the method that finds "x" in the linear algebra equation "$Ax=b$". There are two types of methods for solving $Ax=b$: direct methods and iterative methods.

[1] 3 GB of memory was computed by diving the extra 2 GB needed for Pardiso by 0.65, i.e., 3 GB ~ 2/0.65 GB.

Direct Methods

Direct methods are the default in MSC Nastran and require factorizing the stiffness matrix A and then performing forward and backward solves, often referred to as the FBS part of the solution algorithm. The factorization generally takes 80-90% of the total solve time and the FBS generally takes the remaining 10-20% of the total solve time.

There are two main direct methods in MSC Nastran for SOL 400: the MSC Sparse Direct Solver (MSCLDL) and the Pardiso Solver (PRDLDL). The MSC Sparse Direct Solver is the original sparse direct solver that has been in MSC Nastran for over 20 years and it is the default option. It can run in very limited memory settings but has limited parallel scalability. The Pardiso Solver was added in MSC Nastran 2014.0 for SOL 400. It consumes 5-12x as much memory as MSCLDL depending on the model type, but it can exhibit much greater performance with shared memory parallelism, i.e., SMP.

No changes have to be made to the input file to choose the MSC Sparse Direct Solver since it is the default option. To use the Pardiso Solver, the user needs to add the following to the Executive Control Section (as described in the Quick Reference Guide):

```
SOL 400
SPARSESOLVER NLSOLV(FACTMETH=PRDLDL)
CEND
```

Other options with respect to the Pardiso solver can be found in the Quick Reference Guide.

Recommendations for use of MSCLDL and PRDLDL:

1. If number of solid elements is less than 80% of total number of elements, then use one of the two sparse direct solvers, else use iterative methods (described next).
2. If the above is true and memory is sufficient to use PRDLDL, then use it; otherwise, use MSCLDL.

To determine if memory is sufficient for PRDLDL, we provide the rough guide based on model details. We break up the types of models into shell-element models and solid-element models described in the following table.

Table 7-1 Breakdown of Model Type based on Types of Elements

Table 7-2

Model Type Name	Model Description
Shell-Element models	Models with ≤ 80% Solid Elements
Solid-Element models	Models with > 80% Solid Elements

Next, we discuss memory requirements for models of these types using PRDLDL.

For shell-element models, we have tested PRDLDL on a wide variety of models of sizes ranging from 500K DOFs to 12M DOFs. We investigate the memory requirement for PRDLDL to run in-core and then perform analysis to estimate the memory requirements for various problem sizes. This data is seen in the following table where we provide a suggested memorymax and buffer pool amount for various model sizes.

Table 7-3 Estimates of Memorymax for Shell-Element Models using the Pardiso Solver

Table 7-4

DOF=6*ngrids	Suggested memorymax	Suggested bpool
< 1 Million	16gb	50x
1-2 Million	32gb	50x
2-4 Million	64gb	50x
4-8 Million	96gb	50x
8-12 Million	128gb	50x
12-16 Million	160gb	50x

Note that these are rough guides and may at times overestimate or underestimate but it is a good initial guide. These were developed to increase the likelihood that Pardiso runs in-core and to allow for enough buffer pool memory to reduce I/O costs.

We may want to further reduce I/O costs by increasing the buffer pool memory amount but we need to subsequently increase memorymax. To increase bpool to 80x for the "2-4 Million" case and keep memory to Pardiso constant, the user should recalculate the suggested memorymax according to

$$\text{memorymax} = \frac{1 - 0.5}{1 - 0.8} 48$$

where 0.5 corresponds to the original 50x for bpool, 0.8 corresponds to the new 80x for bpool, and 48 is the existing memorymax value for "2-4 Million". The same analysis can be performed for other model sizes. Note that we will continuously improve on these estimates and build them into the solver in future releases. The user will want to run using these estimates in the following way:

```
nast20180 shellmodel.dat memorymax=XXgb bpool=40x
```

or another appropriate bpool amount.

We have performed the same analysis for solid-element models. The data is in the following table where we provide suggestions for memorymax and bpool.

Table 7-5 Estimates of Memorymax for Solid-Element Models using the Pardiso Solver

Table 7-6

DOF=3*ngrids	Suggested memorymax	Suggested bpool
< 1 Million	32gb	50x
1-2 Million	64gb	50x

Table 7-6

DOF=3*ngrids	Suggested memorymax	Suggested bpool
2-4 Million	96gb	50x
4-8 Million	128gb	50x
8-12 Million	196gb	50x
12-16 Million	228gb	50x

Of course, as with the shell-element models, we can increase bpool amount to further reduce I/O and we should adjust memorymax accordingly as described above.

Finally, based on the value of memorymax for shell-element models and solid-element models, we have the following decision table:

Table 7-7 Decision table on when to use the Pardiso Solver in SOL 400.

Table 7-8

Memory Comparison	Solver Choice
memorymax ≤ 0.75*RAM	Use PRDLDL
memorymax > 0.75*RAM	Use MSCLDL

where RAM is the physical memory available on the machine. This does not need to be followed rigorously but is a good first start to determining if your model can run in-core and fast enough to use the Pardiso Solver. It further avoids approaching the Physical RAM of the machine and thus reducing the ability of the operating system from catching I/O.

Iterative Methods

Iterative methods are the other option. They must be selected by the user in MSC Nastran using a Case Control statement. An iterative method does not rely solely on a factorization and an FBS, but instead performs several iterations, $\{x_1, x_2, x_3, \ldots, x_n\}$, where x_n is a converged approximation to the exact answer. Iterative methods are generally based on the Conjugate Gradient method or the GMRES. These methods can be much faster and consume much less memory for certain problems, mainly solid-element models.

While there are a few iterative method options in MSC Nastran, we focus only on one: the CASI Solver. The CASI solver is selected by adding the following in the Case Control section:

```
SMETHOD=element
```

See the Quick Reference Guide as well. Note that the CASI solver is a Conjugate Gradient solver that uses a preconditioner based on element matrices. As stated in the previous section, we have the following recommendations for using the CASI Solver:

- If number of solid elements is **greater than or equal to** 80% of total number of elements, then use the CASI iterative method.

Examples

The first example that we consider is Model 1 again ran on the Small Memory Machine. We test the Pardiso Solver, the MSC Sparse Direct Solver, and the CASI Solver. CASI is best for solid-element models, and that is the defining feature of Model 1. The results are in Figure 7-3. The results illustrate the standard behavior of the iterative solve for a solid-element model.

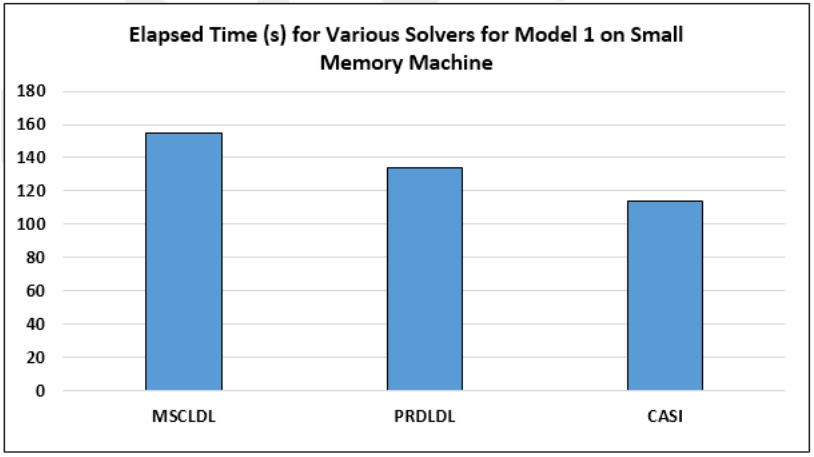


Figure 7-3 Elapsed Time for Various Solver for Model 1 on Small Memory Machine using SMP=8

Recall that this is a solid-element engine-block model where the CASI Solver should be most effective, and it is. From MSCLDL to PRDLDL, we get a 14% performance improvement, and from PRDLDL to CASI, we get a 15% performance improvement as well.

In the next model, we consider Model 2 which is a mostly shell-element model. We again test the three solvers and we first only consider it for serial.

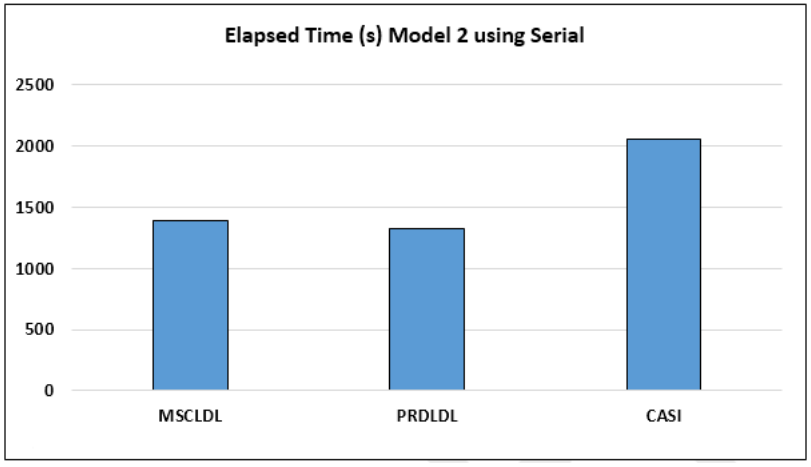


Figure 7-4 Elapsed Time for Model 2 using serial for various solvers

As we see in Figure 7-4, the CASI Solver is not nearly as effective as the two sparse direct solvers. Furthermore, the Pardiso Solver is slightly faster than the MSC Sparse Direct Solver. In the next section, we will see for the case of SMP=8 that the CASI solver is closer in performance to the two sparse direct solvers. However, we will also see that the Pardiso Solver has an almost 20% performance advantage over the other two solvers. One final none on this model is that it has no contact. Contact can negatively affect the performance of the CASI solver for a shell-element model.

Next, we consider Model 4 which has a large amount of contact. We again test with SMP=8 for the three solvers. The results are in Figure 7-5.

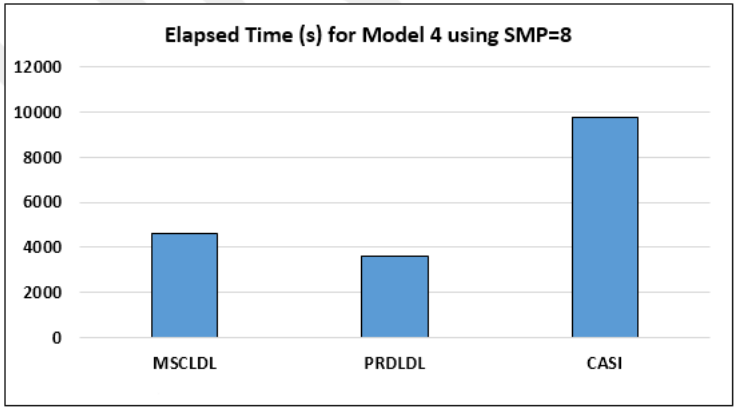


Figure 7-5 Elapsed Time for Model 4, which has nonlinear contact, using SMP=8

We can see in this case that the CASI Solver is not suitable if performance is desired. It reinforces our recommendation that, for shell-element models, the user should employ the Pardiso Solver (PRDLDL) over the MSC Sparse Direct Solver (MSCLDL) or the CASI Solver (CASI) assuming that PRDLDL has sufficient memory to obtain best performance. Refer to "Introduction" on page 111 and "Memory Usage" on page 111 for the memory details.

Parallel Options

There are three parallel options available to users: shared memory parallelism (SMP), distributed memory parallelism (DMP), and general purpose graphics processing units (GPGPU). We describe the benefit of the first two, and later provide the suggestions to the users for each of these options. Since the GPU is only useful for accelerating the MSC Sparse Direct Solver, we refer the reader to Chapter 3 and the details on using the GPU to accelerate the solver.

Before we go any further in the discussion, we remind the user that there are two key parts of the simulation in SOL 400 that affect performance and that benefit from parallelism:

- calculating nonlinear element matrices and stresses
- solving the matrix problem

We have discussed the solver options (MSCLDL, PRDLDL, and CASI), but we have not discussed the time spent computing the nonlinear element matrices and stresses. In the F04 file, a user can see the time spent here in terms of the NLEMG (NonLinear Element Matrix Generation) module. The NLEMG module is called before the solver call and after the solver call in each iteration. Look for the following text in an F04 file:

```
17:07:13     2:31      354.0         0.0       151.4         1.5    NLEMG    BGN
17:07:22     2:40      354.0         0.0       216.8        65.4    NLEMG    END
17:07:25     2:43      354.0         0.0       222.9         6.0    NLEMG    BGN
17:07:33     2:51      354.0         0.0       281.1        58.3    NLEMG    END
```

Since the solver and NLEMG both significantly affect the wall clock time, the user's parallel options should reflect this. We can use either DMP or SMP to accelerate the NLEMG part of the wall clock time just like we can use it for the solver time. Note that GPUs cannot be used to accelerate the NLEMG time.

Distributed Memory Parallel

As stated previously, DMP can be used to accelerate NLEMG. It can also be used to accelerate the MSC Sparse Direct Solver as was illustrated in Chapter 3 - the chapter on Linear Statics performance. Today, if a user runs a SOL 400 NLSTATIC model with "dmp=N" on the command line, then the user will obtain DMP parallelism for both NLEMG and the solver by default. As one can find in the DOMAINSOLVER section of the Quick Reference Guide, this default is the same as having the following option in the Executive Control Section:

```
DOMAINSOLVER NLSOLV,STAT
```

Note that we **do not** recommend this option for users today and will be making changes in the defaults in the future to reflect this. The reason is the poor performance of the DMP Solver (obtained with DOMAINSOLVER STAT) for DMP > 2. If a user wants to run with DMP in SOL 400, then it is recommended to use the following DOMAINSOLVER option:

```
DOMAINSOLVER NLSOLV
```

which is equivalent to turning of the DMP solver and only using DMP for NLEMG. In the remaining tests, we will **not** consider the case of DOMAINSOLVER STAT due to its known poor performance.

While DMP for NLEMG is an effective way to accelerate NLEMG, the approach that we describe in the next section using SMP generally delivers better performance. However, a user should employ DMP in the following cases:

- models with Linear Elements
- models where Linear elements contact Advanced Nonlinear elements

In both of these cases, we cannot use SMP to accelerate NLEMG but have to instead use an alternative like DMP. For the case of contact between element types, a user would run the job with SMP and then see the following User Warning Message in the F06 file:

```
** USER WARNING MESSAGE ***
ROUTINE MATRXA: SMP NOT SUPPORTED FOR CONTACT
BETWEEN MARC AND NASTRAN ELEMENTS
PLEASE SET NLMOPTS,SPROPMAP,2 IN YOUR INPUT FILE.
```

Note that when DMP is used, multiple processes corresponding to the DMP amount are started so a user would see multiple analysis_i8 processes would be observed running on the system. Each process would get a fraction of the total memorymax amount given by 1/(# DMP). However, in SOL 400, only the NLEMG module can take advantage of the DMP capability so much of the other part of the SOL 400 simulation would rely on only the master process. See Figure 7-6 below. This master process would not have all of

memorymax and thus would have less memory and thus potentially do more I/O. This is a limitation of the DMP approach and led to our SMP capability described in the next subsection.

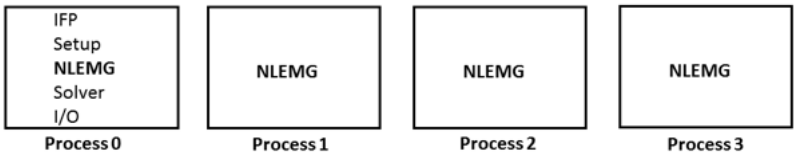


Figure 7-6 Example of DMP=4 and the division of work amongst all four processes showing that NLEMG is only shared while the main (or master) process has to do all other work. Furthermore, each process here as 25% of memorymax amount, even Process 0.

With the SMP approach, we do not divide memory as in Figure 7-6 but one process is employed with multiple threads sharing the memory. Thus, the serial parts of SOL 400 have more memory to use in avoiding I/O and also the SMP aspect of the approach avoids the communication between processes needed by the DMP approach. If DMP is necessary, then the recommendation is to use at most DMP=4 as will be seen in the following simulation results.

Shared Memory Parallel

The SMP approach relies on one process that divides the work amongst multiple threads. There is no need for communicating between processes as with the DMP approach. SMP accelerates every type of solver (MSCLDL, PRDLDL, and CASI) and the NLEMG module. The question then for SMP is how do we choose the value of SMP given the solver type, the model type, and the model size. A general set of guidelines of SMP usage for the various solvers is found below. First, we defined what a small, medium, and large job is and then we define the best SMP options for these cases.

Table 7-9 Definition of Problem Size dependent on Model Type and Number of DOFs.

Table 7-10

Problem Size	Model Types	# DOFs
Small	Solid-Element Dominated	#DOFs < 2M
	All Other Model Types	
Medium	Solid-Element Dominated	2M < #DOFs < 4M
	All Other Model Types	2M < #DOFs < 6M
Large	Solid-Element Dominated	4M < #DOFs < 20M
	All Other Model Types	6M < #DOFs < 30M
Very Large	Solid-Element Dominated	20M < #DOFs
	All Other Model Types	30M < #DOFs

Table 7-11 Recommendation for SMP values for different problem sizes and method types

Table 7-12

Method Type	Small	Medium	Large	Very Large
MSCLDL	SMP=1-2	SMP=3-5	SMP=6-8	SMP=8-16
PRDLDL	SMP=1-4	SMP=5-8	SMP=9-16	SMP=17-32
CASI	SMP=1	SMP=1-2	SMP=3-6	SMP=6-8

Examples

The first model that is considered in this section is Model 3, which is one of the solid-element models. We have tested Model 3 on the HPC Machine for all three solvers: MSCLDL, PRDLDL, and CASI. The results are in Figure 7-7

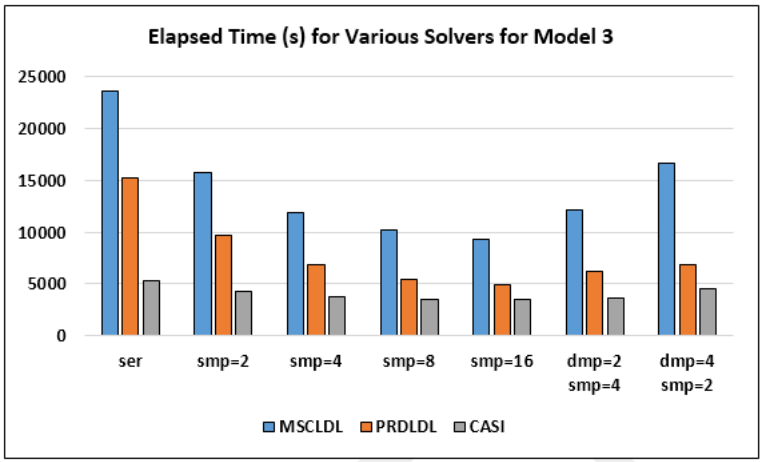


Figure 7-7 Elapsed time for MSCLDL, PRDLDL, and CASI with various parallel options for Model 3

As has been show in Chapter 3, the solid-element models are most suitable for the CASI Solver. We see in Figure 7-7 that using the CASI Solver for serial is competitive against other cases. However, the CASI Solver for SMP=4 gives an additional 30% performance improvement or in this case an additional reduction in wall clock time of 43 minutes.

We also see in Figure 7-7 that the Pardiso Solver is competitive with the CASI Solver at SMP=16 and is almost two times faster than MSCLDL at SMP=16. A user should expect to see much better scalability with the Pardiso Solver than the MSC Sparse Direct Solver.

Lastly, we note the poorer performance in general using DMP instead of strictly SMP, thus we do not recommend DMP in general in SOL 400. The next example that we consider is a shell-element model that

is referred to as Model 4. We again show a broad set of parallel options as well as solver options to provide the users a sense of what they could expect when using various solver options with various parallel options.

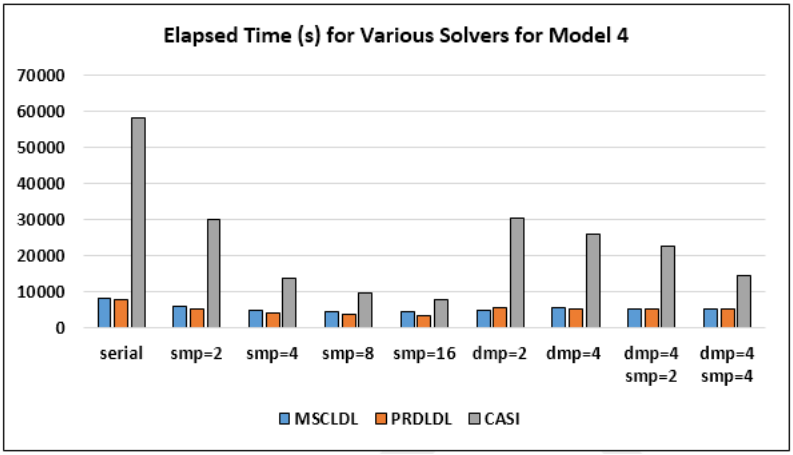


Figure 7-8 Elapsed time for MSCLDL, PRDLDL, and CASI with various parallel options for Model 4

The results in Figure 7-8 illustrate the poor performance of the CASI Solver for a shell-element model. We do see reasonable performance of the CASI Solver at SMP=16 due to the fact that the parallelization is targeted at the iterations. For this model, the iterations taken by CASI were approximately 3000-4000, which is exceedingly for an iterative solver. To see the difference in MSCLDL and PRDLDL, we have included an additional figure without the CASI solver in Figure 7-9.

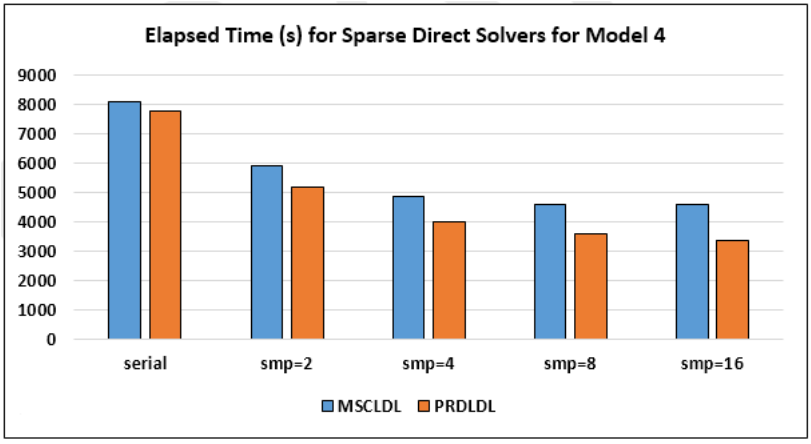


Figure 7-9 Elapsed time for MSCLDL and PRDLDL only with various SMP options for Model 4

These results illustrate the gain in performance in using the Pardiso Solver for SMP and how that gain grows as SMP grows. We can see that at SMP=16 that we can reduce our wall clock time by 20 minutes if we use the Pardiso Solver. Of course, we have to add the caveat as has been observed in previous sections that the model has to fit in memory or else the job will fail. For a system with sufficient memory, this should not be a concern.

Output Request

The output request in SOL 400 has the same dependencies as in SOL 101, 103, and other solution sequences. Thus we advise the reader to refer to Chapter 3 and 4 on the Output Request. We rehash some of the same details here for convenience but also provide some new examples.

We have the same options as for other solution sequences of outputting to an OP2 file, an HDF5 file, or a PCH file. Additionally, the user may request output directly to the F06 file. The amount of output in the output request can have a dramatic effect on the wall clock time, which is due to the time spent in recovering the results and the time spent in writing the data to the results file. A small output request such as

```
SET 1 = 1 THRU 100 $ limited output set
DISP = 1
NLSTRESS = 1
```

will have a minimal impact on the wall clock time, but a large output request such as

```
DISPLACEMENT(SORT1,PLOT,REAL)=ALL
STRESS(SORT1,PLOT,REAL) = ALL
```

with

```
PARAM,POST,1
```

or

```
MDLPRM,HDF5,0
```

can have a noticeable impact on the wall clock time. The choice of whether to use an OP2 or HDF5 file will have a minimal impact on the wall clock time.

Note that a punch file can be generated with

```
DISP(PUNCH,PLOT) = ALL
STRESS(PUNCH,PLOT) = ALL
```

but the use of a PCH file can have a noticeable impact on performance as we will and it will generate a very large output file. We will not explore the use of a PCH file here but refer the reader to Chapter 3 for its performance.

Finally, note that the specification of “PLOT” means that the output quantities are not printed in the F06 file. This will save both time and disk space in the F06 file and should be always used for large output requests.

To get the best performance for the case of a large output request (either use of ALL or a set with a very large number of grid point or element ids), it is imperative that the user following some basic standards:.

1. The user should avoid directing output to a PCH file, if possible.
2. The user should employ the “PLOT” option to eliminate writing to the F06 file.
3. The user should direct output to a fast local disk such as an SSD or a fast spinning HHD.

Even with all of the right hardware and memory usage, the wall clock time can still be noticeably affected by the time spent in writing the output to the file. In an example below, we can see that writing to disk for an OP2 file can consume 25% of the overall wall clock time. There is no way around this other than to make sure that you are only outputting what is necessary for your post-processing needs.

Example

In this example, we first consider Model 2, which has the following output requests:

```
DISPLACEMENT(PLOT)=ALL
STRESS(PLOT,CENTER)=ALL
NLSTRESS(PLOT)=ALL
BOUTPUT(PLOT)=ALL
```

For this model which has 155866 2D elements and 729 3D elements, we obtain a difference in the size of the output files as observed in Figure 7-10 depending on if we use an OP2 file, an uncompressed HDF5 file (MDLPRM, HDF5, 0), or a compressed HDF5 file (MDLPRM, HDF5, 1).

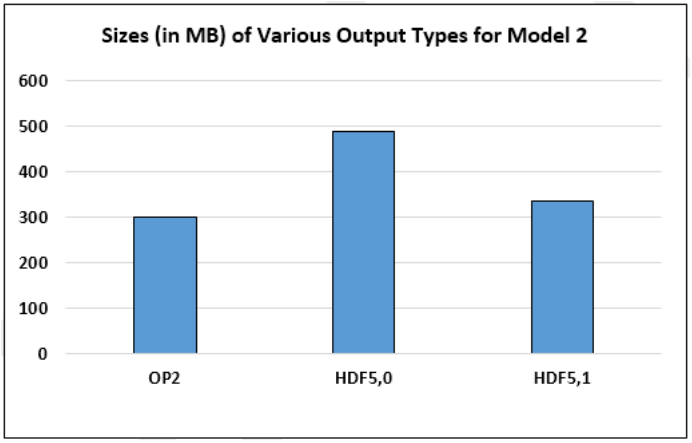


Figure 7-10 Sizes in MB of the Three Main Output Types for Model 2

We have seen similar file size differences in Chapter 4 for Normal Mode Analysis but slightly different behavior in Chapter 3 for Linear Static Analysis. Note that we only observe a 2% difference in wall clock time between OP2 and HDF5,0. Also, note that the penalty in using the compressed HDF5 is very small and thus should be the recommended approach in SOL 400 if a user chooses to employ the HDF5 output format, i.e., use MDLPRM,HDF5,1.

There is no performance penalty in this case for using HDF5 but there is a small file size penalty, even for the compressed version. However, the following are valid reasons for employing the HDF5 file format in output requests:

- Faster attachment to Patran as compared to OP2 files.
- Open format for HDF5 file delivered with each release.
- Available of third party tools that easily integrate with HDF5.

For more details, please refer to the MSC Nastran 2017.0 and 2018.0 Release Guides.

Additional Topics

Linear versus Nonlinear Elements

SOL 400 provides a set of nonlinear finite elements that are not available in linear solution sequences. These are in addition to the standard linear finite elements, like the CQUAD4, that are available in all linear and nonlinear solution sequences. The nonlinear finite elements are more expensive to compute due to the need for more quadrature points and due to the need for repeated calculations during iterations. The nonlinear finite element calculations are thus generally an order of magnitude more expensive.

Thus, we have not provided SMP acceleration for the linear finite elements in SOL 400 to date. Generally, if a user is employing only these linear finite elements in their model in SOL 400, then the cost of element matrix generation is only a few percent of the overall time. In contrast, the cost of computing the nonlinear finite elements can be 40-50% of the overall time. It is for this reason that we have specifically targeted the advanced nonlinear finite elements for SMP acceleration.

LMT2MPC

The default contact approach in SOL 400 is node-to-segment. Node-to-segment contact is implemented internally using Lagrange Multipliers. As a result, when there is contact in the model, the matrix problem that is solved at each iteration is a symmetric highly indefinite matrix of the following form:

$$\mathbb{A} = \begin{bmatrix} A & B \\ B^T & 0 \end{bmatrix}$$

In SOL 400, a user can add the following to the bulk data section:

```
MDLPRM,LMT2MPC,1
```

or

```
MDLPRM,LMT2MPC,2
```

This converts the above matrix into a symmetric positive definite matrix assuming certain condition of the model are satisfied, e.g., no friction. The actual details involve a conversion of Lagrange Multiplier equations to equivalent multi-point constraint equations. The result, in any case, is that the MSCLDL or PRDLDL Sparse Direct Solver may have better behavior.

In the following example, we have investigated Model 4 with the Pardiso Sparse Direct Solver using SMP=16 and turning on the LMT2MPC option. Note that LMT2MPC,1 uses an unsymmetric solver internally and LMT2MPC,2 uses a symmetric solver internally. The symmetric option is often faster. Also, since the result yields a symmetric positive definite matrix in most cases, we have also considered forcing Pardiso to consider the model as positive definite. We do this by adding the following to the bulk data section:

```
MDLPRM,PRDMTYPE,2
```

The results are in Figure 7-11 below. Note that setting the PRDMTYPE to 2 is not recommended without LMT2MPC as the factorization will fail due to the presence of the Lagrange multipliers leading to an indefinite matrix. Also, see the Quick Reference Guide for more details on PRDMTYPE.

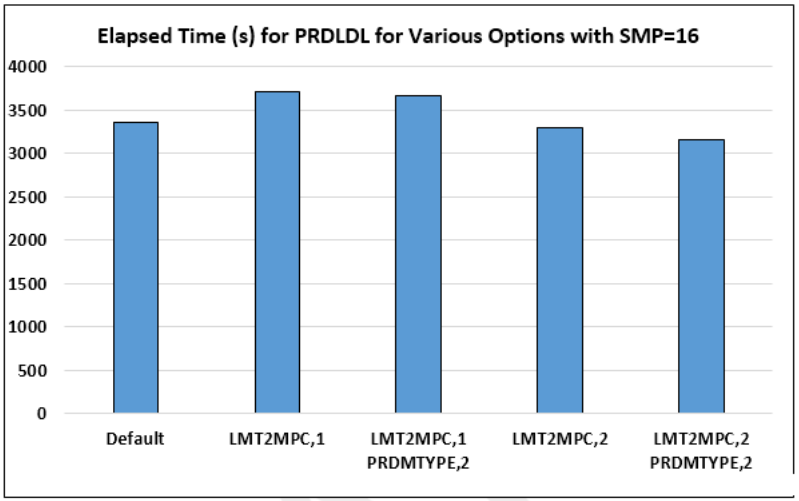


Figure 7-11 Elapsed time for Pardiso with various LMT2MPC options and various Pardiso Matrix Type options for Model 4.

As can be observed, we get a slight performance gain by using LMT2MPC=2 and PRDMTYPE=2. In this case, it is a 6% reduction in the wall clock time.

One last point to note is that LMT2MPC is not available currently with the Nonlinear Transient Analysis, i.e., ANALYSIS=NLTRAN.

Special Settings Known to Affect Convergence

The type of SOL 400 options chosen by the user can significantly impact the wall clock time of the simulation by increasing or decreasing the number of iterations observed in the STS file. While we do not provide detail in this manual on these options, we warn the reader about them and refer them to the Nonlinear User's Guide for further details. Thus, please note the following options:

- "ADAPT" on NLSTEP
- CONV/EPSU/EPSP/EPSW under "MECH" on NLSTEP
- MRCONV under "MECH" on NLSTEP
- NDMAP (Artificial Damping) on NLSTEP
- MDLPRM,NLDIFF,2
- MAXSEP on BCPARA
- NODSEP on BCPARA
- THKOFF on BCPARA
- ICOORD on BCNTPRG

Of particular importance to note is the default change for NODSEP in MSC Nastran 2016.0 from NODSEP=2 to NODSEP=5. This can lead to an increase in the number of iterations for some models but with better contact convergence.

Model and Hardware Details

The following models are used in this section to illustrate our points about obtaining best performance in a linear statics run.

Table 7-13 Summary of Model Types for Performance Comparison

Table 7-14

Model#	Description	DOFs	2D Elems	3D Elems	Iterations
1	Solid-Element Engine-Block Model	2755752	0	282128	1
2	Partial Body-In-White Model	1008732	155866	729	25
3	Large Solid-Element Engine-Block Model	4187376	0	568264	34
4	Large Shell-Element Model	3891930	555984	81738	35

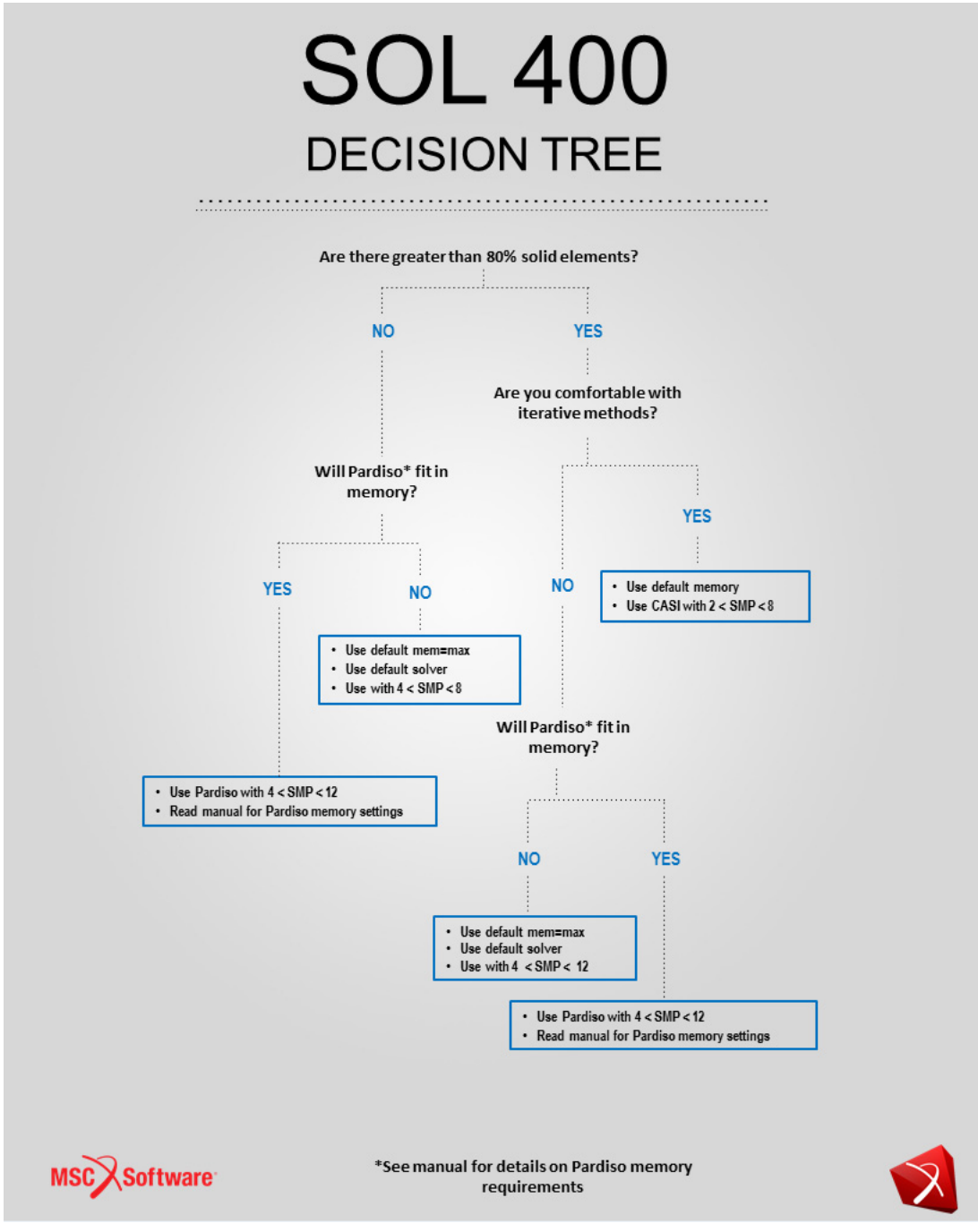
These machines are a large HPC machine, a GPU machine, a small memory machine, and a Windows machine. Details follow:

Table 7-15 Summary of Machine Types for Performance Comparison

Table 7-16

Machine Name	Memory	CPUs	HDD	OS
HPC Machine	256 Gb	2x10 Intel® Xeon® CPU E5-2550 v3 @2.60GHz	4 x 600 GB 15K RPM in RAID0	RH 7.1
Medium Memory Machine	128 Gb	2x8 Intel(R) Xeon(R) CPU E5-2650 v2 @2.60GHz	1 x 550 GB 10K RPM Hard Drive	SuSE SP3
Small Memory Machine	32 Gb	2x10 Intel® CPU @2.8GHz	1 x 1 TB 7200 RPM Hard Drive	RH 6.3

Decision Tree for Nonlinear Static Analysis



8 Automatic Solver Selection

- Introduction 134
- Details 134
- Solver Selection 135
- Parallel Settings 136
- Examples 138
- Machine Learning 140
- Simplified Decision Tree 141
- FAQ 145

Main Index

Introduction

Selecting specific solvers, memory partitions and parallel specifics can be challenging to users. Specifying `SOLVE=AUTO` allows users a simple way to select the ideal settings automatically. The settings are derived from the implementation of Machine Learning for memory prediction for the Pardiso solver as well as a wealth of heuristics from MSC performance tests. Specifically, the `SOLVE=AUTO` process currently is:

- Determine Model characteristics (SOL, size, dominant, element type)
- Determine the best solver (Pardiso, CASI, ACMS, etc). CASI=NO disables the casi solver as an option.
- Determine memory requirements (for a static solver)
- Examine hardware availability (memory / CPU)
- Select Solver
- Select memory and BPOOL
- Select how to partition cores (SMP / DMP)

Details

Model Size Grouping

The definition of model size is set according to the following table, where a Solid Model has at least 80% solid elements (like CHEXA or CTETRA) and the Shell Model has less than 80%.

Table 8-1 Definition of model size based on number of degrees of freedom and on model type.

Model Size	#dof	Model Type
Extra Small	#DOFs <= 0.2M	Solid and Shell Models
Small	0.2M < #DOFs <= 2M	Solid and Shell Models
Medium	2M < #DOFs <= 6M	Shell Models
Medium	2M < #DOFs <= 4M	Solid Models
Large	6M < #DOFs <= 30M	Shell Models
Large	4M < #DOFs <= 20M	SolidModels
Very Large	30M < #DOFs	Shell Models
Very Large	20M < #DOFs	Solid Models

Solver Selection

SOL 101 and 400

In SOL 101 and 400, the key two questions to answer are whether or not the model is solid-dominated so that the CASI Solver will be efficient and whether or not there is enough memory for Pardiso if it is not a solid-dominated model. The most difficult question of the two is determining whether or not there is enough memory for Pardiso.

To address the challenge in SOL 101 and 400, we have added a feature that predicts the memory required by the Pardiso Solver and actually improves on those predictions over time with machine learning techniques. The approach is defined such that if a user runs with a given `memorymax` value for which Pardiso has enough memory to be chosen and then reruns with a smaller `memorymax` value, then a less memory intensive solver will be chosen. Note that the machine learning approach only improves if Pardiso is chosen at the solver and so we have added `solve=train` option that we recommend to use the first few times. `memorymax` defaults to `0.75*PhyscialRam` if solve=auto is specified.

SOL 103 and 111

In SOL 103 and 111, we must determine whether to use the ACMS Solver or the Lanczos Solver. The choice is based on the size of the model and the definition of the number of roots (ND) in the EIGRL entry of the Bulk Data section. The models and default solversdefault solver are broken up by modal solver type according to the following table:

Table 8-2 Selection of modal solver based on the model size.

Model Size	Model Solver
Extra Small	Lanczos
Small	Accelerated ACMS
Medium	Accelerated ACMS
Large	Accelerated ACMS
Very Large	Accelerated ACMS

The ACMS Solver cannot be used with an EIGRL entry that has only an ND entry – it requires a frequency range. Thus, if the user has an EIGRL defined by ND, then we force the Lanczos method.

SOL 107 and 108

Lastly, in SOL 107 and 108, the `solve=auto` method selects the Pardiso Solver in all cases without memory prediction and without a dependency on model size. SMP is then set to CPUMAX / DMP.

SOL 200

Finally, in SOL 200, we again do not use memory prediction, but we determine the type of optimization that is being performed. The decision is based on whether it is topology optimization, topometry optimization, topography optimization, or size and shape optimization. Based on the model characteristics, a combination of direct solver, eigen solver and their parallel implementations with optimal parallel options from the available resources are selected. In SOL 200, the eigen solver is determined based on size of the model and the definition of the number of roots (ND) in the EIGRL entry in the bulk data section similar to SOL 103 and 111. For the selection of eigen solver in sol 200, please refer to Table 8.2. The selections are broken up by solver type according to the following table.

The table below shows the choices that are made based on the optimization type.

Table 8-3 Selection of special solver based on the optimization type

Optimization Type	Special Solver Type	Eigen Solver Type
Topology - Solid Model	CASI for Statics	Accelerated ACMS or Lanczos for modes
Topology - Shell Model	Pardiso for Statics, Frequency Response	Accelerated ACMS or Lanczos for modes
Topometry	Default	Accelerated ACMS or Lanczos for modes
Topography	Default	Accelerated ACMS or Lanczos for modes

Parallel Settings

The choice of SMP and DMP are a function of the solution sequence, Model Size and the Cores available. Below are examples for 16 core systems:

SOL 101, 107, and 400 with solve=auto and with cpumax=16

Solver Type	Parallel Selection for Model Sizes				
	Extra Small	Small	Medium	Large	Very Large
MSCLDL	SMP=4	SMP=4	SMP=8	SMP=12	SMP=12
Pardiso	SMP=4	SMP=4	SMP=8	SMP=16	SMP=16
CASI	SMP=4	SMP=4	SMP=8	SMP=12	SMP=12

SOL 103 and 111 with solve=auto and with cpumax=16

For SOL 111 w/ ACMS

DMP = min(ncore, 4, memory_estimate/memory_max)

if model is (extra small or small)

```
   SMP = min(  ( (Ncore/2)/DMP ), SMP with available memory)
else
   SMP = min (   (Ncore/DMP), SMP with available memory)
```

Solver Type	Parallel Selection for Model Sizes				
	Extra Small	Small	Medium	Large	Very Large
Lanczos	DMP=1 SMP=1	DMP=1 SMPp=4	DMP=2 SMP=6	DMP=2 SMP=8	DMP=2 SMP=8
ACMS w/ SOL 103	DMP=1 SMP=4	DMP=2 SMPp=4	DMP=2 SMP=6	DMP=2 SMP=8	DMP=2 SMPp=8
ACMS w/SOL 111	DMP=1 SMP=4	DMP=2 SMP=4	DMP=2 SMP=6	DMP=4 SMP=4	DMP=4 SMP=4

SOL 108 with solve=auto will use:

DMP = min(Ncore, Nfreq, Memory_Estimate/memorymax)

SMP = Ncore/DMP

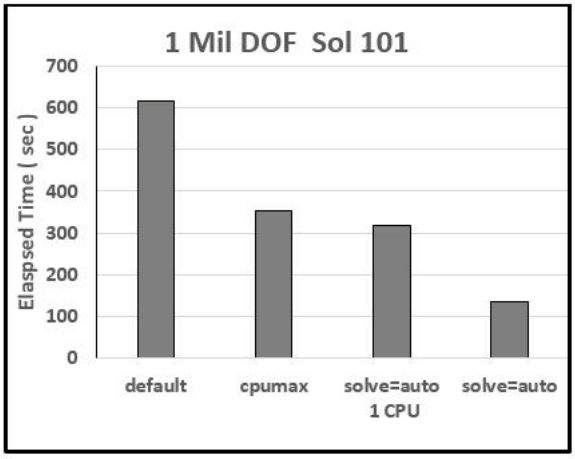
Solver Type	Parallel Selection for Model Sizes				
	Extra Small	Small	Medium	Large	Very Large
Default	DMP=8	DMP=2 SMP=8	DMP=2 SMP=8	DMP=2 SMP=8	DMP=2 SMP=8

Examples

SOL 101

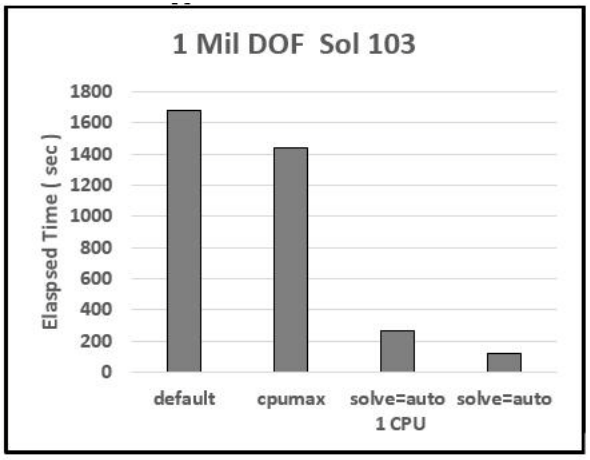
Below are the results of a SOL 101 1.2 M DOFs Radiator model. The model is made up of half 2-D and 3-D elements. The default serial run took 616 seconds. When “cpumax=20” was added, the SMP=4 was selected and the time was reduced to 355 seconds. `SOLVE=AUTO` was specified (with cpumax=1) to see the effect of the solver selection. Sufficient memory was available for Pardiso and using the Pardiso solver reduced

the time to 319 seconds. When "cpumax=20" was added, SMP=4 was selected and the time was further reduced to 136 seconds.



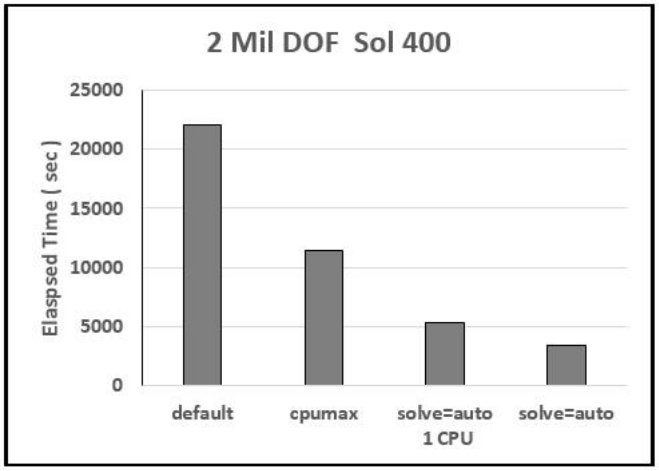
SOL 103

Below are the results of a SOL 103 1.3 M DOFs Car Body model. The model is made up entirely of 2-D elements. The final run resulted in 1000+ modes. The default serial run took 1679 seconds. When "cpumax=20" was added, the SMP=4 was selected and the time was reduced to 1437 seconds. `SOLVE=AUTO` was specified (with cpumax=1) to see the effect of the solver selection. ACMS was selected, reducing the time to 262 seconds. When "cpumax=20" was added DMP=2/SMP=4 was selected and the time was further reduced to 116 seconds.



SOL 400

Below are the results of a SOL 400 2.1 M DOFs Car Engine model. The model is made up entirely of 3-D elements. The default serial run took 22,021 seconds. When “cpumax=20” was added, the SMP=4 was selected and the time was reduced to 11,461 seconds. SOLVE=AUTO was specified (with cpumax=1) to see the effect of the solver selection. CASI was selected, reducing the time to 5,308 seconds. When “cpumax=20” was added SMP=8 was selected and the time dropped further to 3446 seconds.



Machine Learning

Pardiso memory prediction coefficients are stored in $MSC_BASE/$MSC_VERSD/nast/util/mlCSV. If solve=train is specified, then:

1. User’s coefficients will be used. If a user does not have any, then the system values will be initially copied.
2. If solve=train is specified, then the model will run with Pardiso and memory coefficients are stored in:

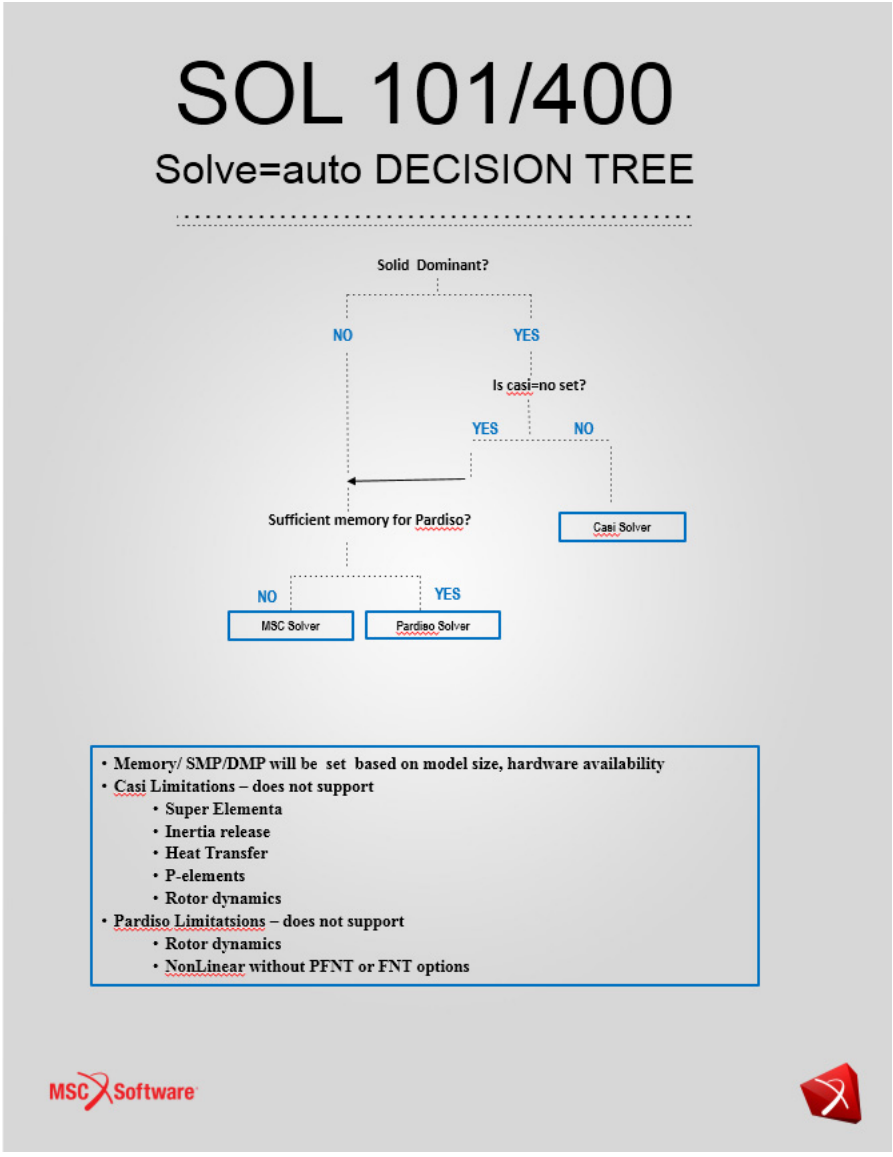
   $HOME/ .coefFile* and $HOME/.trainset* on linux

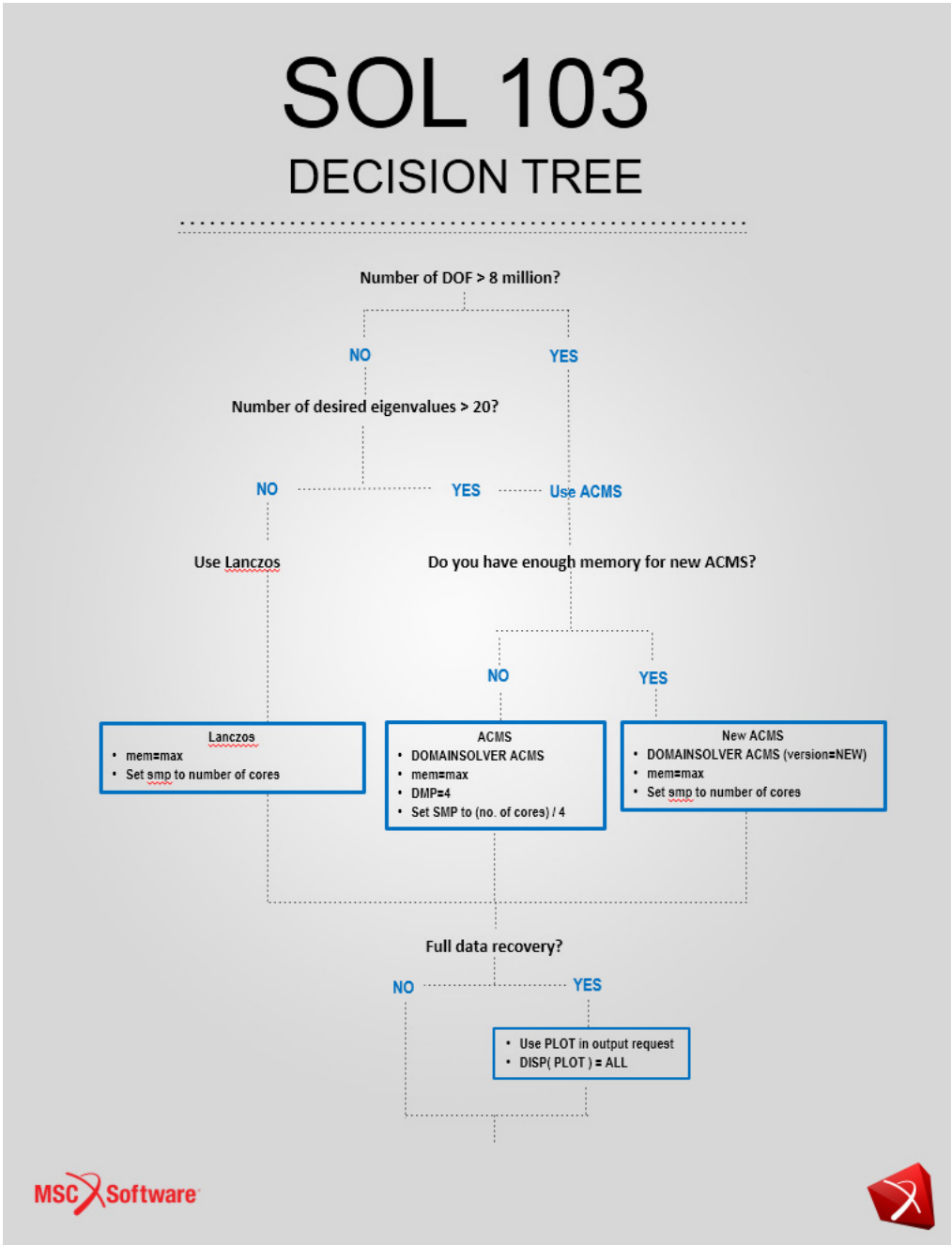
   Roaming Profile/coefFile* and Roaming Profile/trainset* on Windows
3. The model will run with Pardiso. and coefficient will be updated.

If a model is run with solve=auto and the Pardiso solver is used, then the machine learning coefficients for memory prediction will be updated.

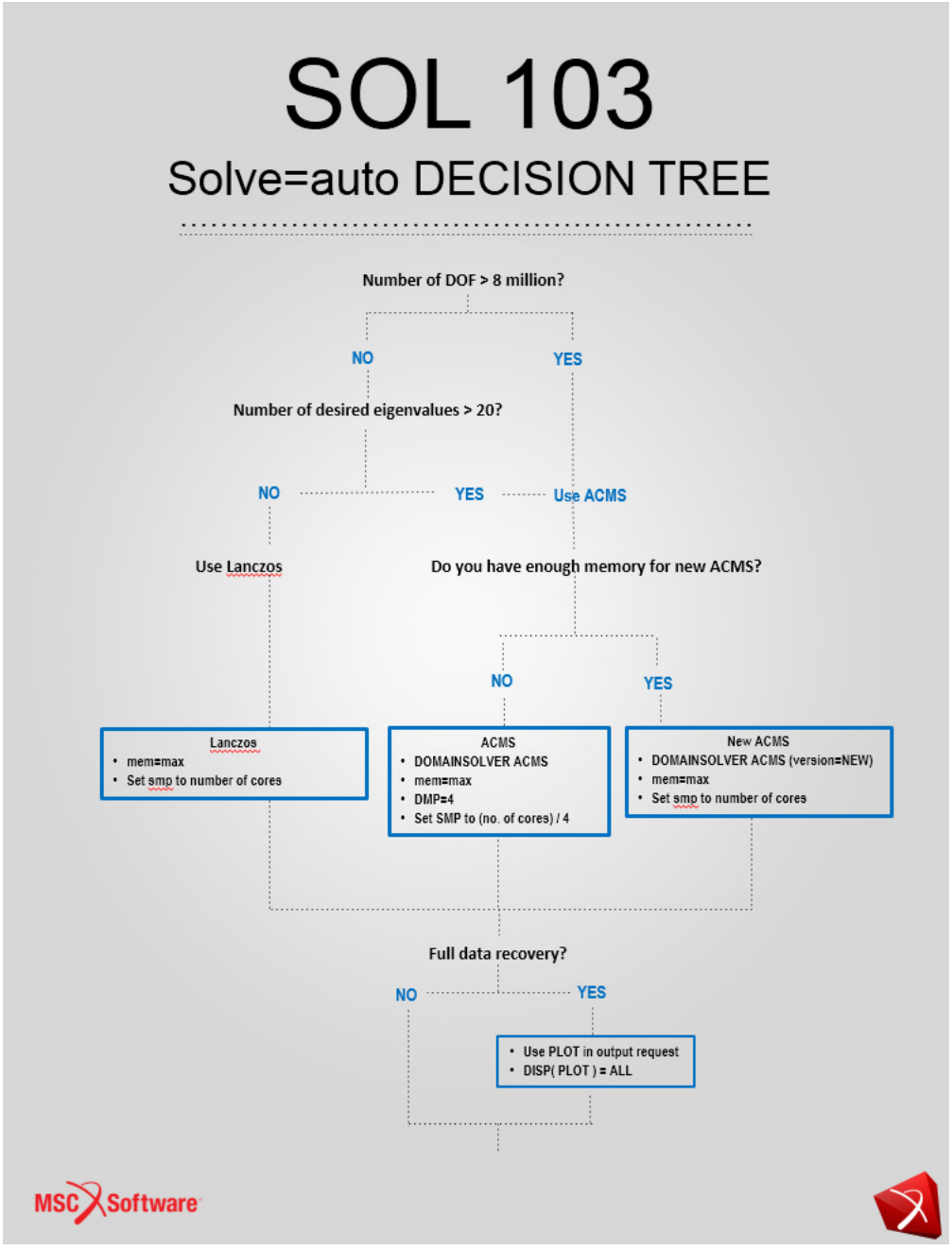
Simplified Decision Tree

Simplified decision trees when "SOLVE=AUTO" is specified.



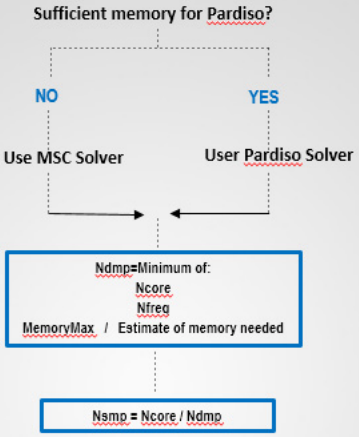

SOL 103
DECISION TREE
Number of DOF > 8 million?
NO
YES
Number of desired eigenvalues > 20?
NO
YES
Use ACMS
Use Lanczos
Do you have enough memory for new ACMS?
NO
YES
Lanczos
• mem=max
• Set smp to number of cores
ACMS
• DOMAINSOLVER ACMS
• mem=max
• DMP=4
• Set SMP to (no. of cores) / 4
New ACMS
• DOMAINSOLVER ACMS (version=NEW)
• mem=max
• Set smp to number of cores
Full data recovery?
NO
YES
• Use PLOT in output request
• DISP(PLOT) = ALL
MSC Software

Main Index


SOL 103
Solve=auto DECISION TREE
Number of DOF > 8 million?
NO
YES
Number of desired eigenvalues > 20?
NO
YES
Use ACMS
Use Lanczos
Do you have enough memory for new ACMS?
NO
YES
Lanczos
• mem=max
• Set smp to number of cores
ACMS
• DOMAINSOLVER ACMS
• mem=max
• DMP=4
• Set SMP to (no. of cores) / 4
New ACMS
• DOMAINSOLVER ACMS (version=NEW)
• mem=max
• Set smp to number of cores
Full data recovery?
NO
YES
• Use PLOT in output request
• DISP(PLOT) = ALL
MSC Software

SOL 108

Solve=auto DECISION TREE



MSC Software

FAQ

General Questions

1. What if Pardiso is selected and there is insufficient memory?

   The solver will be automatically switched internally to the MSC Sparse Solver for sol 108 and sol 400.

2. What if a method is selected that I do not have a license for?

   The job will crash with a "SYSTEM FATAL MESSAGE 3060". Currently licenses are not verified. In this case either purchase additional licenses or turn off SOLVE=AUTO.

3. What if the Pardiso memory estimates are inaccurate?

   The initial estimate will not include data from your model, thus if you rerun with solve=auto, then your memory estimates should improve.

9 General Guidelines

- System Settings 148
- Interpreting the F04 file 148
- Disk Usage 151
- SMP Parallel CPU Usage 152
- DMP Parallel Communication 152
- Known Errors 152
- FAQ 155

Main Index

System Settings

We have previously covered memorymax, memand bpool in specific chapters, like Chapter 3 on Linear statics. We make the following general recommendations:

```
memorymax=0.5Xphysical $ leaving 50% RAM for the OS
mem=max                $ a function of memorymax, Ndmp, number of nodes
buffsize=65537         $ decrease to 8153 if DOF < than 1 Million
```

Note that bpool should be automatically set by the “mem=max” or “solve=auto” commands. For certain usages, like with the Pardiso solver, we will recommend command-line overrides for memorymax to 0.75Xphysical, i.e.,

```
memorymax=0.75Xphysical $ leaving 25% RAM for the OS
```

For details on buffsize, refer to FAQ #18.

Interpreting the F04 file

Memory

Memory Requested and Layout is listed near the top of the F04. E.g.:

```
USER OPENCORE (HICORE)          = 116348 MB
EXECUTIVE SYSTEM WORK AREA      =      9 MB
MASTER(RAM)                     =      2 MB
SCRATCH(MEM) AREA               =     50 MB(100 BUFFERS)
BUFFER POOL AREA (BPOOL4)       =  12884 MB(25768 BUFFERS)

TOTAL MSC NASTRAN MEMORY LIMIT = 129345 MB
```

The TOTAL amount is the amount requested on the command line 0.75xPhys. It is partitioned into a section for the solver (HICORE) and a section for the executive system (dominated by BPOOL). Setting `"solve=auto"` defaults to requesting up to 75% of the physical amount of RAM. “`mem=max`” defaults to requesting 50% of the physical amount of RAM. This amount can be increased by adding options like “`memorymax=0.8xphysical`”. The “memorymax” value can also be a number such as 128 GB, i.e. “`memorymax=128gb`”.

Note that the MSC Nastran HPC team does not recommend setting memorymax above 75% of physical RAM since the OS needs memory for secondary I/O caching.

Memory Usage within a module is listed for certain modules:

UIM 4157 – Symmetric Decomposition (matrix factorization):

```
*** USER INFORMATION MESSAGE 4157 (DFMSYM)
     PARAMETERS FOR SPARSE DECOMPOSITION OF DATA BLOCK  SCRATCH ( TYPE=RSP ) FOLLOW
                    MATRIX SIZE =     35374 ROWS            NUMBER OF NONZEROES =    256848 TERMS
          NUMBER OF ZERO COLUMNS =         0       NUMBER OF ZERO DIAGONAL TERMS =         0
          ELIMINATION TREE DEPTH =      3175
               CPU TIME ESTIMATE =         8 SEC              I/O TIME ESTIMATE =         0 SEC
       MINIMUM MEMORY REQUIREMENT =        23 MB             MBMEMORY AVAILABLE =     30136 MB
     MEMORY REQR'D TO AVOID SPILL =        26 MB            MEMORY USED BY BEND  =     26 MB
     EST. INTEGER WORDS IN FACTOR =        26 MB              EST. NONZERO TERMS =     53 MB
     ESTIMATED MAXIMUM FRONT SIZE =       987 TERMS                RANK OF UPDATE =        64
  7:55:40     3:03    60213.0        12.0       177.0        0.4    SPDC BGN TE=8
  7:55:40     3:03    60238.0        25.0       177.6        0.5    SPDC END
 *** USER INFORMATION MESSAGE 6439 (DFMSA)
     ACTUAL MEMORY AND DISK SPACE REQUIREMENTS FOR SPARSE SYM. DECOMPOSITION
        SPARSE DECOMP MEMORY USED =      14 MB                 MAXIMUM FRONT SIZE =       987 TERMS
```

```
        INTEGER WORDS IN FACTOR =       2 MB            NONZERO TERMS IN FACTOR =      6865 K TERMS
SPARSE DECOMP SUGGESTED MEMORY =       9 MB
```

SIM 4199 – FBS (forward-backward substitution):

```
*** SYSTEM INFORMATION MESSAGE 4199 (PREFACT)
    A PORTION OF THE SPARSE FACTOR HAS BEEN CACHED IN MEMORY FOR THE FBS.
    371527 FRONTAL MATRICES OUT OF A TOTAL OF 611683 ARE STORED IN MEMORY.
    MEMORY AVAILABLE: 41 MBS
    ESTIMATED ADDITIONAL MEMORY NEEDED TO STORE THE ENTIRE FACTOR: 97 MB
```

UIM 4216 – Unsymmetric Sparse (matrix factorization):

```
*** USER INFORMATION MESSAGE 4216 (UDSFA)
                    PARAMETERS FOR PARALLEL SPARSE UNSYM. DECOMP OF DATA BLOCK  SCRATCH ( TYPE=RSP ) FOLLOW
                    MATRIX SIZE =   34782 ROWS             NUMBER OF NONZEROES =   11704 TERMS
          NUMBER OF ZERO COLUMNS =       0        NUMBER OF ZERO DIAGONAL TERMS =       0
                    SYSTEM (107) =   32778                   REQUESTED PROC. =      10 CPUS
               CPU TIME ESTIMATE =       0 SEC                I/O TIME ESTIMATE =       0 SEC
     ESTIMATED MEMORY REQUIREMENT =     6 MB                  MEMORY AVAILABLE =    154076 MB
     EST. INTEGER WORDS IN FACTOR = 1 MB                   EST. NONZERO TERMS =        42 K TERMS
     ESTIMATED MAXIMUM FRONT SIZE =      18 TERMS               RANK OF UPDATE =      64
 *** USER INFORMATION MESSAGE 6439 (UDSFA)
     ACTUAL MEMORY AND DISK SPACE REQUIREMENTS FOR SPARSE UNSYM. DECOMPOSITION
     SPARSE DECOMP MEMORY REQUIRED = 8 MB                      MAXIMUM FRONT SIZE =      18 TERMS
           INTEGER WORDS IN FACTOR = 1 MB                 NONZERO TERMS IN FACTOR =      42 K TERMS
    SPARSE DECOMP SUGGESTED MEMORY =    11 MB
```

Pardiso Memory Usage:

```
      PARAMETERS FOR INTEL MKL PARDISO PARALLEL SPARSE SOLVE
       MATRIX SCRATCH  on unit 305 ( TYPE=2) FOLLOW
                     MATRIX SIZE =1330140 ROWS
           NUMBER OF NONZEROES =144242379 TERMS
         NUMBER OF ZERO COLUMNS =0
                NUMBER OF LOADS =1
                   SYSTEM (107) =10
           MKL REQUESTED PROC. =10 CPUS
       MEMORY PARAMETERS FOR INTEL MKL PARDISO PARALLEL DECOMPOSITION FOLLOW
                       AVAIL. CORE MEMORY =117641 MB
          APPROX. REQUI. IN-CORE MEMORY =71895 MB
       APPROX. REQUI. OUT-OF-CORE MEMORY =12004 MB
```

SIM 4171: CASI Iterative Solver Memory in the F06:

```
*** SYSTEM INFORMATION MESSAGE 4171 (ITSCAS)
     CASI ITERATIVE SOLVER SOLUTION PROCESS STATISTICS FOLLOW FOR LOAD CASE NUMBER:     1
     TOTAL OPERATION COUNT                    =     9.610127E+11
     NUMBER OF PCG ITERATIONS                 =           291.
     TIME IN SECONDS FOR CONJUGATE GRADIENT =     6.842972E+01
     TIME IN SECONDS FOR TOTAL PCG SOLVER   =     1.466713E+02
     MEMORY REQUIRED IN MB                    =           2197.3
     DISK SPACE REQUIRED IN MB                =           2627.7
```

MPYAD (Multiply-Add):

In versions prior to 2018, occasionally an MPYAD may take a long time. Often the excessive time is a result of a poor method selection. DIAG 19 may be used to see the estimates of each method. If a poor method is chosen you can change the method (See *Numerical User's Guide*).

Total Memory Usage is listed near the end of the F04, however it is not reliable. E.g.:

```
*** TOTAL MEMORY AND DISK USAGE STATISTICS ***

+---------- SPARSE SOLUTION MODULES -----------+         +------------- MAXIMUM DISK USAGE -------------+
    HIWATER               SUB_DMAP        DMAP               HIWATER               SUB_DMAP         DMAP
    (WORDS)       DAY_TIME     NAME      MODULE                 (GB)    DAY_TIME      NAME          MODULE
      788832329   05:09:14    MDACMS    166  READ             691.936   03:57:00    ADJSENF    50  DSADJ
```

MPYAD output starting in version 2018 looks like:

```
============================
||| M MATRIX VQSPRX   TRL(     2970     8185  2  1     1199   9.51627E-02   1.00653E-01)
||| P MATRIX MDPHIQX  TRL(     3102     8185  2  1     8185   1.00000E+00   1.00000E+00)
||| Y MATRIX [C] IS PURGED
||| A WORKING MEMORY = 13677332361     SYSTEM(66) =        0      SYSTEM(107) =    32772
||| D NULL  COLSA    =          162     NULL ROWSB =        0      NULL COLSB  =        0
|||   TRANSPOSE FLAG =            1     METHOD:   DENSE
=============================================================================================
 *19** MPYAD DENSE:       TIME TO FILTER A     =    0.0007
 *19** MPYAD DENSE: ALL MATRICES IN CORE
 *19** MPYAD DENSE:         TIME TO READ A     =    0.0346
 *19** MPYAD DENSE:         TIME TO READ B     =    0.0787
 *19** MPYAD DENSE:         TIME TO MULTIPLY   =    2.8001
 *19** MPYAD DENSE:         TIME TO WRITE D    =    0.0379
  *19** MPYAD DENSE:        TOTAL ELAPSED TIME =    3.0295
```

MSC Nastran does not track the memory at every stage of each solution. While a value is printed for the Sparse Solution Modules, it should be ignored and may be removed in releases subsequent to MSC Nastran 2018.0.

Disk Usage

The amount of I/O may be greatly reduced with increased memory given to MSC Nastran using memorymax, and thus Buffer Pool. If restarts are not needed, "`scr=yes`" is recommended.

Within a Module: The amount of I/O is printed in the F04. E.g.:

```
 Day_Time  Elapsed     I/O_Mb      Del_Mb     CPU_Sec    Del_CPU    Subroutine
 14:45:34    54:15      981.5G         0.0     54885.7        0.0    SEDRCVR 699      SDR3     BEGN
 14:46:54    55:35      999.7G     18575.0     54965.3       79.6    SEDRCVR 699      SDR3     END
```

Total Usage: The total amount is listed near the bottom of the F04:

```
*** TOTAL MEMORY AND DISK USAGE STATISTICS ***

+---------- SPARSE SOLUTION MODULES ----------+        +------------ MAXIMUM DISK USAGE ------------+
    HIWATER              SUB_DMAP        DMAP              HIWATER              SUB_DMAP        DMAP
    (WORDS)       DAY_TIME    NAME      MODULE                (GB)   DAY_TIME     NAME         MODULE
      788832329   05:09:14    MDACMS    166  READ           691.936   03:57:00    ADJSENF    50  DSADJ


*** DATABASE USAGE STATISTICS ***

 +---------------- LOGICAL DBSETS ----------------+   +------------------------------ DBSET FILES ----------------------------+
  DBSET      ALLOCATED  BLOCKSIZE       USED   USED     FILE      ALLOCATED  ALLOCATED    HIWATER     HIWATER  I/O TRANSFERRED
             (BLOCKS)    (WORDS)    (BLOCKS)      %                (BLOCKS)       (GB)    (BLOCKS)       (GB)             (GB)

  MASTER       25000      65536          63   0.25   MASTER       25000      12.21        1792       0.875           12.943
  DBALL     20000000      65536          12   0.00   DBALL     20000000    9765.62          12       0.006            0.010
  OBJSCR       25000       8192         584   2.34   OBJSCR       25000       1.53         585       0.036           31.083
  SCRATCH   16384100      65536      383790   2.34   (MEMFILE       100       0.05         100       0.049            0.000)
                                                    SCRATCH     8192000    4000.00      997496     487.059        486706.455
                                                    SCR300      8192000    4000.00      417711     203.960          5850.728
                                                                                                             =============
                                                                                                 TOTAL:      492601.219
```

where:

- HIWATER values are the maximum disk space used at any time.
- I/O transferred is the amount of I/O that occurred. Minimizing this value is good.
- SCRATCH is used for storing data used by multiple modules.
- SCR300 is used for storing data within a module.
- The ALLOCATED amount is the internal MSC Nastran file size limit. It is a function of the database block size, which is controlled by the BUFFSIZE parameter. The maximum block size is 65536 WORDS. The BUFFSIZE is defined as the block size plus 1, so that the maximum BUFFSIZE is 65537. This maximum BUFFSIZE is recommend for nontrivial models (i.e. BUFFSIZE=65537). To increase the size of the database further, use ASSIGN statements or the SSCR command line option.

I/O performance is best measured by CPU to elapsed time ratio. For serial processing 100% is the goal (ELAP/CPU=1.0). For parallel it is not as simple but in general, the parallel functions should perform optimally. The F04 file has BEGN and END time stamps for all functions above a certain threshold. Set SYS20=0 to print all values.

SMP Parallel CPU Usage

The CPU time for each process spent in some of the older MSC Nastran math kernels is printed in the F04 file. This list is NOT exhaustive of the SMP benefit, but is listed here for clarification that all modules are not listed.

```
*** PARALLEL PROCESSES CPU INFORMATION (SECONDS) ***

 MODULE    PROCESS-1    PROCESS-2    PROCESS-3    PROCESS-4    PROCESS-5    PROCESS-6    PROCESS-7    PROCESS-8
 --------  -----------  -----------  -----------  -----------  -----------  -----------  -----------  -----------
 P-MPYAD       35.71        35.71        35.71        35.71        35.71        35.71        35.71        35.71
 P-SDCMP       25.39        22.27        22.94        21.07        19.10        15.04        14.52        14.10
 P-SFBS       480.30       477.12       444.51       452.60       475.74       479.68       499.85       455.66
 ========  ===========  ===========  ===========  ===========  ===========  ===========  ===========  ===========
 TOTAL         541.40       535.10       503.16       509.38       530.55       530.43       550.08       505.47
```

DMP Parallel Communication

The amount of communication on a DMP process is printed at the bottom of the F04 file:

```
*** MPI STATISTICS FOR DISTRIBUTED NASTRAN ***

     FROM      MESSAGES   TOTAL (MB)   MAX BYTE   MIN BYTE   AVG BYTE
 ----------  ---------  -----------  ---------  ---------  ---------
    1 -> 2        18105   10300.5361909614544          8     596569
    2 -> 1      1899495    8981.797    134464          8       4958


 TOTAL NUMBER OF MESSAGES SENT =          1917600
 TOTAL AMOUNT OF DATA XFR (MB) =        19282.334
 AVERAGE MESSAGE LENGTH        =            10544.
```

Known Errors

General Messages

DBSET SCRATCH IS FULL The space limit defined has been exceeded.

```
 *** USER FATAL MESSAGE 1012 (GALLOC)
     DBSET SCRATCH IS FULL AND NEEDS TO BE EXPANDED.
     USER ACTION: SEE THE MSC NASTRAN CONFIGURATION AND OPERATIONS GUIDE OR
          KB8012329 ON THE MSC WEB SITE FOR METHODS TO MAKE LARGER DATABASES.
```

User Action: This limit is reached most often due to using a small block size on the data base. Increase the database block size by setting BUFFSIZE=65537 on the command line, or use the NASTRAN entry in your input data file (NASTRAN BUFFSIZE=65537). In addition, you may use an ASSIGN statement or the "SSCR" keyword to increase the software limit.

Pardiso

Pardiso runs out of memory with no memory information in F04 file

```
 *** SYSTEM FATAL MESSAGE 11320 (DIRECT)
     Insufficient memory for the Pardiso Solver.
     USER INFORMATION:
     Pardiso indicates there is not enough memory for solve.
     USER ACTION:
     Try searching .F04 file for Pardiso minimum memory
     requirement. Use the reported values to increase memory using
     'mem=' on the submittal line. Alternatively, if MDLPRM,PRDOOC,0
     was set, un-set it in order to allow out-of-core execution of
     Pardiso Solver.
```

```
0FATAL ERROR
```

User Action: If no memory value reported in F04 file, then determine the number of grids and model type and use estimates from Section 3.2 to select more appropriate memory amount.

Pardiso runs out of memory with following memory information in F04 file

```
 PARAMETERS FOR INTEL MKL PARDISO PARALLEL SPARSE SOLVE
MATRIX KLL      on unit 103 ( TYPE=1) FOLLOW
              MATRIX SIZE =606225 ROWS
     NUMBER OF NONZEROES =23450717 TERMS
  NUMBER OF ZERO COLUMNS =0
         NUMBER OF LOADS =1
            SYSTEM (107) =1
     MKL REQUESTED PROC. =1 CPUS
MEMORY PARAMETERS FOR INTEL MKL PARDISO PARALLEL DECOMPOSITION FOLLOW
               AVAIL. CORE MEMORY =1255 MB
    APPROX. REQUI. IN-CORE MEMORY =4418 MB
APPROX. REQUI. OUT-OF-CORE MEMORY =1454 MB
```

User Action: Increase memory given to solver by 4418-1255=3163 MB. For example, if job ran with mem=2000mb bpool=150, then rerun with mem=5163mb bpool=150. Note that for slightly less memory the job will run out-of-core and it is in fact safe to run with approximately 3976-3163=2721 MB extra instead of 3163 MB extra. Also, see FAQ 13.X on how to increase memory to solver when using memorymax.

Pardiso encounters indefinite matrix in SOL 101

```
*** SYSTEM FATAL MESSAGE 11332 (DIRECT)
     Zero pivot or iterative refinement problem in the Pardiso Solver.
     USER ACTION:
     Pivoting issues can sometimes be resolved by changing the
     pivoting perturbation. See notes for SYSTEM(91).
0FATAL ERROR
```

User Action: Above user action will be updated in a later MSC Nastran release. Correct action today is to add the following to the to bulk data section.

```
MDLPRM,PRDMTYPE,-2
```

ACMS

UFM 3200 with old ACMS

```
*** USER FATAL MESSAGE 3200 (PRESOL)
    LOGIC ERROR DETECTED BY SUBROUTINECCGRAF2        3 = LOCATE CODE OR VALUE.
```

User Action: Add PARTMETH and TIPSIZE to the DOMAINSOLVER line

```
domainsolver acms (partmeth=metiso, tipsize=1000)
```

Stack Memory Issue with new ACMS

```
*** SYSTEM FATAL MESSAGE 3008 (ACMS1)
    INSUFFICIENT MEMORY AVAILABLE FOR SUBROUTINE ACMSXX
```

Note that ACMSXX is a generic name for an ACMS subroutine. Also, the above can occur due to memory limitations but a user should check log file for current resource limit to see if following is present:

```
Current resource limits:
CPU time:                       unlimited
Virtual address space:          unlimited
Working set size:               unlimited
Data segment size:              unlimited
Stack size:                     10240 KB
Number of open files:           1024 (hard limit: 4096)
File size:                      unlimited
Core dump file size:               0 MB
```

User Action: Set stack size to be unlimited to avoid this problem.

Lanczos

SFM 3034 with Lanczos

```
*** SYSTEM FATAL MESSAGE 3034 (LNNHERR)
     INTERNAL FAILURE IN THE LANCZOS PROCEDURE:
     M-ORTHOGONAL QR PROCEDURE FAILED TO CONVERGE. PROBABLE CAUSE:
     MASS MATRIX IS INDEFINITE (MODES) OR STIFFNESS MATRIX IS INDEFINITE (BUCKLING).
     USER ACTION: CONTACT MSC CLIENT SUPPORT.
*** SYSTEM FATAL MESSAGE 7340 (LNNHERR)
     WARNING REPORTED BY SUBROUTINE LNNDRVD (IER= 728)
     USER INFORMATION: GRAM-SCHMIDT DID NOT CONVERGE.
```

User Action:

If SOL 103 (modes), then check model for indefiniteness of mass matrix by trying following:

- Rerun with the Massless Mechanism check activated by setting PARAM,MECHFIX,YES in bulk data.
  - Reduce the number of eigenvalues in order to save time (set ND to 20 or so).
- Examine the F06 file for results from the Massless Mechanism check.
  - If the model contains a massless mechanism, these DOF are identified.
  - These DOF should be constrained, or the model should be examined in this area for valid connectivity.

If SOL 105 (Buckling), then check model for indefiniteness of buckling matrix by trying following:

- Make sure the structure is properly constrained and has no mechanisms, it is not possible to do a buckling analysis with a free-free structure.
- If a static preload is applied be sure that the static is not higher than the buckling load. This can be accomplished by removing the static preload and performing a buckling analysis using the static preload as the live load.

SFM 5407 with Lanczos

```
*** SYSTEM FATAL MESSAGE 5407 (LNCCKS)
    INERTIA (STURM SEQUENCE) COUNT DISAGREES WITH THE NUMBER OF MODES ACTUALLY COMPUTED IN A (PARTIAL) INTERVAL
 *** SYSTEM FATAL MESSAGE 7340 (LNNRIGL)
     NUMBER OF COMPUTED EIGENVALUES EXCEED ALLOCATED STORAGE.
```

User Action:

If SOL 103 (modes), then check model for indefiniteness of mass matrix by trying following:

- Rerun with the PARAM,MECHFIX,YES set in bulk data section (Massless Mechanism checker).
  - Best to reduce the number of eigenvalues in order to save time (set ND to 20 or so).
- Examine the F06 file for results from the Massless Mechanism check.
  - If the model contains a massless mechanism, these DOF are identified.
  - Constrain these DOF or examine the model for valid connectivity in this area.

FAQ

General Questions

1. How can I tell where time was spent in a non-ACMS simulation?

   The "f04reprt" utility will print a summary of the times spent of each module. For example,

   MSC_BASE/bin/MSC_VERSD f04reprt file.f04

   Entries sorted by CPU Time

   ```
   Module          CPU Time    Elapsed Time
   ```

```
-------- -------------- --------------
READ              2569.5           2577
EMA                 44.4             45
EMA                 44.2             44
```

We recommend to use "f04reprt –c –e" to get cumulative and elapsed times instead of default CPU times.

2. How can I tell where time was spent in an ACMS simulation?

   Use "f04reprt" with the following guidelines:

   a. *For fully detailed results, set PRINT=YES on the DOMAINSOLVER command in the executive control input section, and set PARAM,ACMSMSG,1 in the bulk data input section. Then you can use f04reprt as shown in previous examples.*

   b. *If PRINT=YES and PARAM,ACMSMSG are not specified, use the "subdmap" option of f04reprt. This is accomplished by including "-d" on the*

   c. *f04reprt command line. An example is shown below. Note that the time for ACMS Phases 1 and 2 are the sum of the time for subdmap MDACMS0 and subdmap MDACMSX. The time for ACMS Phase 2 is shown as subdmap INVLAN and is a subset of the time taken in MDACMSX.*

```
$ f04reprt -d -c -m 1.0 xxxx.f04
MD Nastran/MSC.Nastran F04 File Summary Utililty    12:56:53 Thu 11/10/16
(Combined Module Listing)

File: xxxx.f04
File Machine ID:  x86-64 Based Sys Intel 3000 MHz (sude Linux 3.10.0-229
File BUFFSIZE:   16385, MEM: * N/A *
File MEM Highwater:  60977712 WORDS
File Disk Highwater: 1313.750 MB
File Total I/O:      2.529 GB (32.37 MB/Elapsed second)

DMAP Usage Information
Entries sorted by CPU Time

DMAP           CPU Time   Elapsed Time   Call Count
-------- -------------- -------------- ------------
SEMODES            65.2             79            1
MODEFSRS           58.9             71            1
MDACMSX            57.1             65            1
ACMSPRT            45.0             54            1
INVLAN             11.4             11            1
PHASE1DR            3.9              5            1
PHASE1A             2.1              3            1
SEMG                2.0              2            1
PHASE1B             1.8              2            1
IFPL                1.8              3            1
IFPSTAR             1.8              3            1
MDACMS0             1.6              6            1
--------                               ------------
49                                               12

Maximum DMAP stack depth is 6.
```

```
Information from EXIT Record:
  EXIT (CPU)            65.5
  ELAPSED               80
  EXIT/ELAPSED         0.819
```

3. Why does "free" show memory not being released after a MSC Nastran run? Is there a memory leak?

   The Linux OS keeps files in cache. Once MSC Nastran is run, the information is left in "cache" and "free" appears to have memory allocated. It will be released as soon another program requests it.

Parallel Usage Questions

1. Why doesn't MSC Nastran scale to X CPU cores on today's multi-core machines with X CPU cores?

   *Amdahl's law [1] states that the theoretical speed-up of software for which p percent of the software is sped-up by s is 1/(1 – p + p/s). In a perfect world, p % of MSC Nastran is parallelized with SMP and DMP such that s = SMP*DMP and we have a speed-up of s, i.e., perfect scalability. In reality, anywhere from 10% - 80% of MSC Nastran is parallelized depending on the solution sequence and s < DMP*SMP. We are always working to increase percent, p, and have s ~ DMP*SMP.*

2. Can I run with multiple nodes on a Windows cluster?

   Not at this time.

3. Should I buy a GPU?

   Each chapter has a specific section on the performance of MSC Nastran for GPUs. It is generally the case that it is only beneficial when models are mainly composed of solid elements and the simulation is dominated by matrix factorizations, not like in SOL 103 where there are lots of modes and FBS dominates. Furthermore, each user should evaluate whether multiple CPU cores can give them the same benefit as a GPU.

4. Can I run DMP on a single node?

   Yes, it may be run on a single node and for many solution sequences it is highly recommended. Please refer to each specific chapter on specific recommendations.

5. Can I use SMP and DMP in the same run?

   Yes, starting with MSC Nastran 2018.0, a user can simply specify cpumax=N and if DMP is appropriate for that solution sequence then it will automatically be chosen along with an appropriate SMP.

6. Why doesn't MSC Nastran scale on lots of nodes?

   For certain solution sequences, like S0OL 108, MSC Nastran can scale across many nodes. For other solution sequences, like SOL 101, the scaling is very limited. The MSC Nastran team has focused on SMP scalability over the last several releases with longer term plans to increase scalability across multiple nodes. Refer to specific chapters on DMP support and usage within each solution sequence.

Hardware Questions

1. Should I buy the newest Intel or AMD CPU?

   MSC Nastran HPC team cannot make a recommendation on chips, but MSC Nastran uses MKL libraries which are tuned to the latest Intel chips so consider that fact in your purchases.

2. Should I buy SSDs or RAM?

   It is nearly always the recommendation to invest in newer, faster, and larger memory amounts than SSDs, but if memory amounts are maxed out and models are large enough that I/O will occur, then SSDs are a good investment for performance. However, it may be most beneficial to buy 4-8 15K HDDs setup in a RAID 0 configuration to get best performance per dollar. Please contact hardware vendors for details.

3. What should I invest in for best performance if I want to upgrade my machine?

   Focus first on maximizing the memory capacity of your current machine with the newest and fastest memory (i.e., DDR4 memory). Memory is inexpensive today as compared to CPUs and the reduction in wall clock time due to avoiding I/O can be significant. After that you should next consider adding faster hard drives or SSDs. See also Chapter 2.

4. Does the performance of MSC Nastran behave differently on a Windows machine versus a Linux machine?

   Windows may be up to 30% slower than Linux. It is highly analysis dependent.

5. Do you recommend to use a Linux machine or a Windows machine for MSC Nastran? If so, why?

   As stated in previous answer to the previous question., Linux generally has better performance than Windows and is the recommended HPC platform.

Memory Questions

1. What is memorymax and "mem=max"?

- *memorymax is the maximum amount of memory allocated for a MSC Nastran job.*
  - *Default is 50% of Physical RAM.*
  - *Can be set on the command line or in RC files (i.e. memorymax=0.75xPhys)*
  - *Should be set to 75% of Physical RAM when using Pardiso Solver and New ACMS.*
- *mem=max looks at "memorymax" and sets the memory allocated with following rules*
  - *mem=0.80 x memorymax if New ACMS and Master*
  - *mem=0.20 x memorymax if New ACMS and Slave*
  - *mem=memorymax / (Ndmp/Nhosts) if DMP*
  - *mem=memorymax in all other cases*
- *mem=max also defines bpool according to following rules*
  - *bpool = mem – (estimate for solver) if SOL 101 or 400*
  - *bpool = 0.25 * mem if not SOL 101 or 400*

Note that use of memorymax and mem=max is recommended memory approach.

2. How do I add an extra "*Ygb*" of memory to the solver (or Open Core) when using "mem=max" and "memorymax=Xgb"?

   If SOL 101 or 400, then you should do the following:

   - *Calculate Buffer Pool amount from the F04 file to be, say, "Wgb".*
   - *If Y < W, then rerun with "bpool=(W-Y)gb"*
   - *Increase memorymax if (W-Y) is small compared to memorymax.*

   If not SOL 101 or 400, then you can do either of the following:

   - *Set memorymax to be "X + Y/0.75"*
   - *If 0.25*X > Y, then add "bpool=Zgb" where Z=(0.25X-Y).*

3. How does buffsize affect performance and what value should I use?

   Buffsize determines the maximum database capacity. A large buffsize for a small model (< 1M DOF) can affect performance negatively. A small buffsize for a large model (> 1M DOF) can cause the simulation to issue a System Fatal Message as the database fills up. For models greater than 1M DOF, it is recommended to use the maximum buffsize=65537.

4. Why does the Pardiso solver require so much more memory than the default solver, MSCLDL?

   The default solver, MSCLDL, is a frontal solver that keeps minimal fronts in memory simultaneously. Thus, it can run in a very limited memory environment but it has limited scalability. The Pardiso solver, obtained via the Intel MKL, is a multifrontal solver that must have more fronts in memory at once to obtain the scalability that it obtains. See https://en.wikipedia.org/wiki/Frontal_solver.

ACMS and Lanczos Questions

1. My answers are different from one version to another, or between Lanczos and ACMS. Why?

   Check your EIGRL frequency range. The maximum frequency on your EIGRL entry (field 4) should be at least two times larger than your maximum forcing frequency. Otherwise, you are at risk of incurring the effect of "modal truncation," where the modal space does not contain sufficient information to represent the motion in the range of interest. An indication of modal truncation is the degree of separation between the highest elastic mode compared to the lowest residual augmentation vector. This separation should be several hundred Hertz, or at least one order of magnitude.

   For example, if your EIGRL frequency range is 0.0 to 200.0 Hertz, and your highest eigenvalue is at 199.9 Hertz, the lowest residual augmentation vector should probably be at least 700 Hertz or 800 Hertz or more. These are hypothetical values; the important thing to note is that these "modes" should be well separated. If they are not well separated, it could indicate that the residual augmentation vectors actually contain some important vibration information. This typically indicates your EIGRL frequency maximum should be higher.

   If your EIGRL maximum frequency is already two to three times higher than your highest forcing frequency, and there is sufficient separation between the highest elastic mode and the lowest residual augmentation vector, you can try increasing the UPFACT parameter. This parameter increases the maximum cutoff frequency used during the ACMS modal reduction process. The default value of UPFACT is 2.0. It is set on the DOMAINSOLVER executive section entry. See Chapter 3 in this manual for more information about UPFACT.

2. Why does new ACMS require so much memory?

   The new ACMS reduction module (ACMS1) operates in SMP mode with a minimum of disk I/O operations. Memory is used to reduce I/O. The lower triangle of the global stiffness and mass matrices are held in memory. If coupled mass is used (PARAM, COUPMASS, 1), the mass matrix will require roughly the same memory amount as the stiffness matrix. If coupled mass is not required, then memory requirements will be reduced, sometimes significantly, since the lumped mass approach reduces the number of mass matrix entries. The lumped mass approach is the default for structural analysis.

Random Solution Sequence Questions

1. How do I accelerate SOL 112 (Modal Transient)?
   - *Use DOMAINSOLVER ACMS(VERSION=NEW) if there are enough modes*
   - *Use SMP based on definitions in Chapter 4 on Normal Modes for ACMS.*
2. How do I accelerate SOL 107 (Complex Eigenvalues)?
   - *Use SPARSESOLVER CEAD(FACTMETH=PRDLU)*
   - *Use SMP=2-8 for extra performance*
3. How do I accelerate Fatigue?
   - *Using Multiple Threads (FTGPARM) scales the Fatigue portion of the run almost linearly.*
   - *Set msglvl (FTGPARM) to be less than 4.*

References

https://en.wikipedia.org/wiki/Amdahl%27s_law