Optimizing NX Nastran Performance

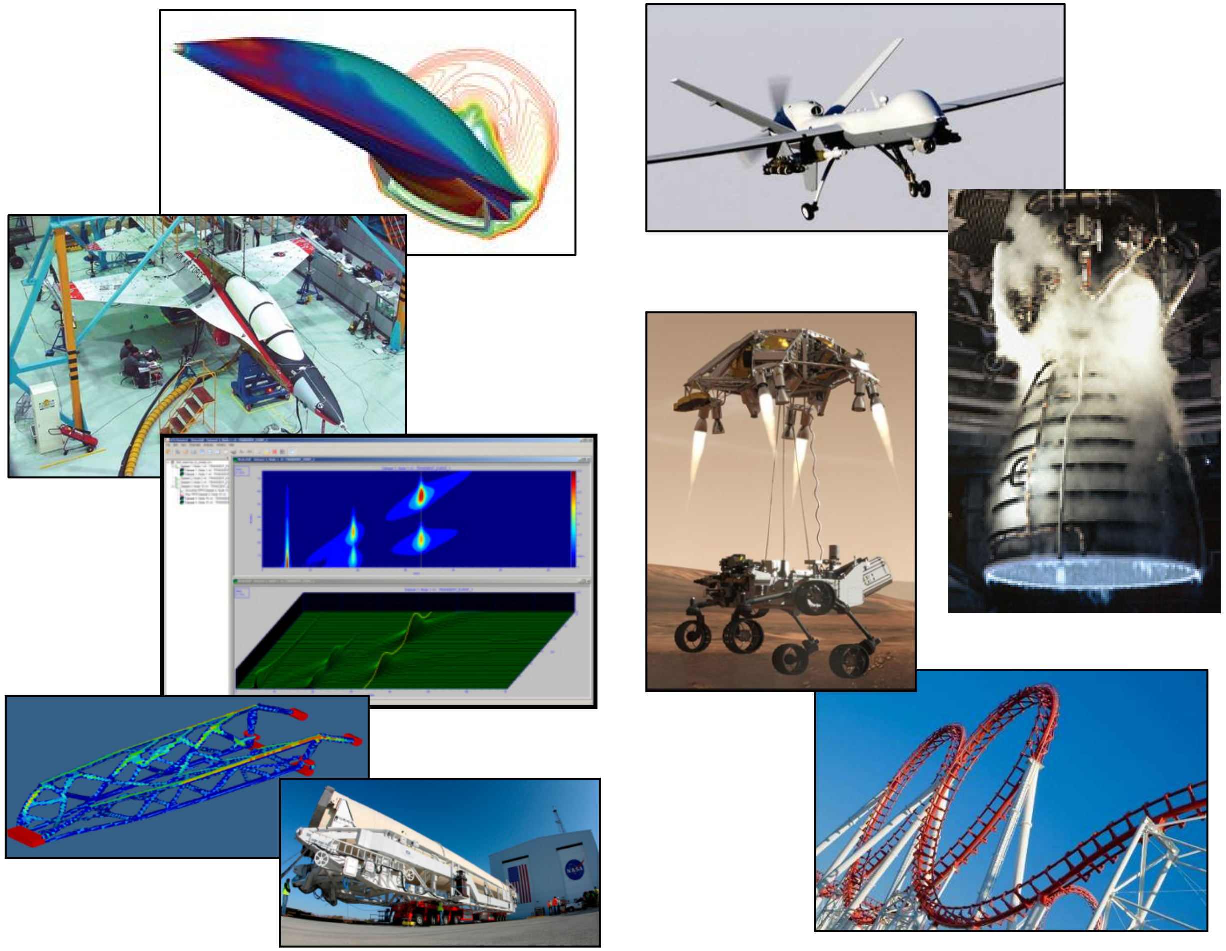
Paul Blelloch, Director Aerospace Analysis, ATA Engineering, Inc.

For audio, if not already connected please dial in to:
+1-855-797-9485 US Toll free
+1-415-655-0002 US Toll
Access code: 994 539 664

Smarter decisions, better products.

ATA Engineering, Inc.

Innovative Solutions through Test and Analysis-Driven Design



ATA Engineering Offers Services for:

- Design
- Analysis
- Test
- Process Improvement

ATA has 30+ years experience running Nastran on wide variety of problems

ATA is a value-added-reseller of Siemens PLM Software's CAD/CAE and PLM software packages.

For more information, visit our website

http://www.ata-e.com

http://www.ata-plmsoftware.com

Outline

What are Major Factors that Control Nastran Performance?

- I/O, I/O and I/O

Understanding Nastran Files

- LP vs. ILP versions of Nastran
- Using .f04 file to understand performance

Effective Use on an HPC Cluster for NX Nastran

- Controlling amount of I/O – Getting fast I/O
- Direct vs. Iterative solutions

Overview of Parallel Processing

- SMP, DMP, etc.

Goal is to provide you with tools to get best NX Nastran performance

ATA ENGINEERING, INC.

SIEMENS

NX Nastran Performance by I/O, I/O, I/O

- To a large degree NXN performance governed by I/O speed to DBALL, SCRATCH and SCR300 files
- No matter what else you do if you don't have fast I/O for SCRATCH/SCR300 you don't have good performance

DataBase (.MASTER/.DBALL)

Scratch (.SCRATCH)

Module Scratch (.SCR300)

Lots of I/O

Input (.dat or .bdf)

Run Time Summary (.f04)

Human Readable Output (.f06)

Machine Readable Output (.pch)

Post Processing (.op2)

Understanding LP/ILP Versions of NXN

All versions of NX Nastran come in 2 flavors

- LP uses 4 byte integers and addresses up to 8 GBytes of RAM
- ILP uses 8 byte integers and addresses large amount of RAM

{~INSTALL_PATH}\NXNastran\nxn9\64bit\bin

nastran64.exe ← 64-bit LP command line version

nastran64L.exe ← 64-bit ILP command line version

nastran64Lw.exe ← 64-bit ILP GUI version

nastran64w.exe ← 64-bit LP GUI version

On Linux in /bin directory

```
nastran -> nast9
nastran64L
```

- Some larger models require ILP version to solve
- <u>LP version has 4 bytes per word/ILP has 8 bytes per word</u>
- Can run into trouble transferring files between versions
  - e.g. external superelements
- Look at “Executable:” in .log file to see which version you’re running
  - `Executable:  c:/program files/siemens/nxnastran/nxn9/64bit/nxn9/em64tnt/analysis.exe` ← LP on Windows
  - `Executable:  c:/program files/siemens/nxnastran/nxn9/64bit/nxn9/em64tntl/analysis.exe` ← ILP on Windows
  - `Executable:  /usr/local/apps/NASTRAN/NX/9.0/nxn9/x86_64linux/analysis` ← LP on Linux
  - `Executable:  /usr/local/apps/NASTRAN/NX/9.0/nxn9/x86_64linuxl/analysis` ← ILP on Linux

To Understand Nastran Performance Need to Understand .f04 file!

To understand performance of any program we run a “profiler”

The .f04 file is Nastran’s profiler output (effectively a DMAP line profiler)

Contains information on overall memory and disk usage as well as a step by step summary of every line of DMAP code that’s executed:

```
Day_Time  Elapsed    I/O_Mb     Del_Mb     CPU_Sec    Del_CPU    SubDMAP Line (S)SubDMAP/Module

16:24:56     1:15     3845.0         5.0        70.1        0.0     SEMG    123     EMG     BEGN
*8** Module  DMAP Matrix         Cols       Rows  F T     NzWds     Density  BlockT  StrL    NbrStr    BndAvg    BndMax    NulCol
     EMG     123  KELM           5479        300  2 1       300 9.15904E-01      25    35     42490        292        300         0 *8**
     EMG     123  MELM           5319         78  2 1        12 1.53268E-01       3     0     63588         77         78        10 *8**

16:25:25     1:44     7685.0         2.0        93.4        0.0     XREAD   251     READ    BEGN
*** USER INFORMATION MESSAGE 4157 (DFMSYN)
    PARAMETERS FOR SPARSE DECOMPOSITION OF DATA BLOCK  MXX      ( TYPE=RSP ) FOLLOW
                     MATRIX SIZE =    31516 ROWS            NUMBER OF NONZEROES =    11137293 TERMS
          NUMBER OF ZERO COLUMNS =        0        NUMBER OF ZERO DIAGONAL TERMS =        0
               CPU TIME ESTIMATE =       62 SEC                I/O TIME ESTIMATE =        0 SEC
      MINIMUM MEMORY REQUIREMENT =    3499 K WORDS              MEMORY AVAILABLE = 1407554 K WORDS
    MEMORY REQR'D TO AVOID SPILL =   10636 K WORDS          MEMORY USED BY BEND  =    8311 K WORDS
    EST. INTEGER WORDS IN FACTOR =     11138 K WORDS          EST. NONZERO TERMS =     11140 K TERMS
    ESTIMATED MAXIMUM FRONT SIZE =    3141 TERMS                  RANK OF UPDATE =     128
16:25:25     1:44     7921.0       236.0        93.9        0.5     SPDC BGN TE=62
16:25:28     1:47     8155.0       234.0        97.0        3.1     SPDC END
*** USER INFORMATION MESSAGE 6439 (DFMSA)
    ACTUAL MEMORY AND DISK SPACE REQUIREMENTS FOR SPARSE SYM. DECOMPOSITION
        SPARSE DECOMP MEMORY USED =   10636 K WORDS            MAXIMUM FRONT SIZE =    3141 TERMS
          INTEGER WORDS IN FACTOR =        37 K WORDS    NONZERO TERMS IN FACTOR =     11124 K TERMS
   SPARSE DECOMP SUGGESTED MEMORY =    5519 K WORDS
```

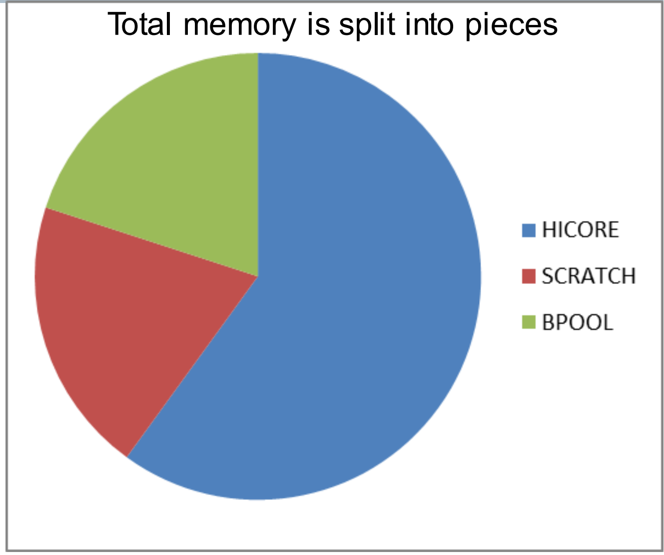
Use DIAG,8 to write matrix trailers (other diagnostics also useful)

Understanding Memory in Nastran

From .f04 file:

```
0     ** MASTER DIRECTORIES ARE LOADED IN MEMORY.
         USER OPENCORE (HICORE)          =   804908563  WORDS
         EXECUTIVE SYSTEM WORK AREA      =      316925  WORDS
         MASTER(RAM)                     =       80917  WORDS
         SCRATCH(MEM) AREA               =   268443648  WORDS (     8192 BUFFERS)
         BUFFER POOL AREA (GINO/EXEC)    =   268427231  WORDS (     8189 BUFFERS)

         TOTAL NX NASTRAN MEMORY LIMIT   =  1342177284  WORDS
```



Total memory is what's requested on mem= keyword (default is 45% of physical)

- Total RAM available to Nastran throughout its run time

Scratch(MEM) is what's requested on smem = keyword (default is 20% of mem)

- RAM available to write scratch files before spilling to disk (very useful for slow disks/lots of RAM)

Buffer pool is what's requested on bpool = keyword (default is 20% of mem)

- RAM available to buffer writes to scratch files (100 buffers is 50 Mbytes)

User OpenCore is what's left over for "in-core" operations (default is 60% of mem)

- RAM available for memory intensive operations such as decomposition (typically not much needed)

Maximum memory is 8 Gbytes for LP version

- memorymaximum keyword sets maximum allowed memory request (default is 80% of RAM)

Defaults work "on average" but our experience is that maximizing smem optimizes performance

“Optimal” Memory Allocation

Typically not much advantage to allocating more memory to HICORE than required to avoid spill in decomposition step (there are some exceptions)

```
 PARAMETERS FOR SPARSE DECOMPOSITION OF DATA BLOCK  KXX     ( TYPE=RSP ) FOLLOW
                     MATRIX SIZE =    31516 ROWS             NUMBER OF NONZEROES =    11864738 TERMS
          NUMBER OF ZERO COLUMNS =        0        NUMBER OF ZERO DIAGONAL TERMS =        0
                CPU TIME ESTIMATE =     204 SEC                  I/O TIME ESTIMATE =       0 SEC
      MINIMUM MEMORY REQUIREMENT =    3518 K WORDS              MEMORY AVAILABLE = 1407554 K WORDS
    MEMORY REQR'D TO AVOID SPILL =   19946 K WORDS          MEMORY USED BY MMD    =    3362 K WORDS
    EST. INTEGER WORDS IN FACTOR =     18846 K WORDS          EST. NONZERO TERMS =     36674 K TERMS
    ESTIMATED MAXIMUM FRONT SIZE =    3466 TERMS                  RANK OF UPDATE =     128


*** 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                  (MB)   DAY_TIME      NAME         MODULE
   1391409310   14:20:21    XREAD     251  READ                3850.188   14:21:11    XREAD     251  READ
```

For this problem maximum HICORE memory was 8*1391409310/1024^3=10.4 Gbytes (that’s a lot)

- This particular problem includes fluid mass that results in a very dense mass matrix
- Most solutions in Nastran actually don’t require that much (only 155 Mbytes required to avoid spill in above example)

To calculate memory start from total and leave maximum amount for smem:

- 80% of physical is 25.6 Gbytes (mem = 25.6 gb) – same as memorymaximum default
- Buffpool of 100 blocks is 50 Mbytes – not much advantage beyond that
- 10.5 Gbytes required for HICORE, so about smem=15 gb is optimal

Look at Bottom of .f04 for Database Usage

```
*** DATABASE USAGE STATISTICS ***

+------------------ LOGICAL DBSETS -------------------+  +------------------------- DBSET FILES --------------------------+
 DBSET       ALLOCATED   BLOCKSIZE        USED    USED    FILE        ALLOCATED     HIWATER       HIWATER  I/O TRANSFERRED
              (BLOCKS)     (WORDS)    (BLOCKS)       %                 (BLOCKS)    (BLOCKS)          (MB)             (GB)

 MASTER           5000       65536          42    0.84    MASTER           5000          42        10.500            0.290
 DBALL         2000000       65536           3    0.00    DBALL         2000000           3         0.750            0.004
 OBJSCR           5000        8192         285    5.70    OBJSCR           5000         285         8.906            0.016
 SCRATCH       4000100       65536          19    0.00   (MEMFILE           100         100        25.000            0.000)
                                                          SCRATCH       2000000      450748    112687.000         1017.365
                                                          SCR300        2000000        4883      1220.750           11.971
                                                                                                            ==============
                                                                                                    TOTAL:        1029.645
```

- Size in GBytes = BUFFSIZE*BLOCKS*BYTES/WORD/1024^3
  - Maximum BUFFSIZE is 65537 (recommended for large runs)
    - Recommendation is maximum for more than 400,000 DOF
  - <u>Bytes/Word is 8 for ILP and 4 for LP</u>
  - Above example has limit of 2,000,000*65,537*8/$1,024^3$=976.6 GBytes allocated to DBALL
    - Allocation is logical limit, not related to actual disk space available
  - Maximum size of SCRATCH+SCR300=222.5 GBytes
    - Total I/O to SCRATCH+SCR300>1 Terabyte (in 14 minutes on a desktop)

Summary of Terms so Far

- LP – long precision version of Nastran (32 bit integers)
- ILP – Integer long precision version of Nastran (64 bit integers)
- WORD - 4 bytes for LP version and 8 bytes for ILP version
- BUFFSIZE - Number of words in a block
- BLOCK (Buffsize words – 65536*8 = 512 Kbytes)
- SMEM – Memory reserved to “write” scratch files
- BPOOL – Memory reserved to cache writing of scratch files
- HICORE – Memory reserved for in-core operations
- SCRATCH – File used by Nastran to store intermediate data
- SCR300 – Scratch file used for temporary storage within a module
- DBALL – Database file used to store data for restart
- DMAP – Language that Nastran solution sequences are written in

How Do You Control I/O?

- Files can be controlled with INIT cards in input deck
- Easier to control by keywords on nastran submittal or RCF file
  - System .rcf file in Nastran installation (e.g. …/conf/nast9rc or …\conf\nast9.rcf)
  - Hierarchy of .rcf files (system, architecture, node, user, local)

Keyword	Interpretation
Buffsize	BUFFSIZE in blocks
dbs	Location of MASTER/DBALL files
sdball	Logical size limit of DBALL file
sdir	Location of SCRATCH/SCR300 files
sscr	Logical size limit for SCRATCH/SCR300
scr	What gets written to DBALL file (yes, no, mini)
smem	RAM allocated to SCRATCH/SCR300
mem	Total RAM allocated to run

Typical .rcf file

```
Buffsize=65537
dbs=S:\scratch
sdball=500gb
sdir=S:\Scratch
sscr=200gb
scr=no
```

I/O Speed

Faster

RAM
Local SSD
Local HD
Network

Slower

Principles for Getting Fast I/O in NXN

- Largest amount of I/O is to SCRATCH and SCR300 file
  - If you have LOTS of RAM (100+ GBytes) use smem to allocate some portion of total memory (mem) to SCRATCH
    - mem=200gb, smem=190gb allocates 190 GBytes to SCRATCH, 10 to solver
  - If you don’t have enough RAM to fit entire SCRATCH/SCR300 files make sure that these are written to a fast local disk
    - Solid state disks are faster than spinning disks (SSD’s not created equal)
- If you want general RESTART use scr=no to save max data to DBALL
  - Make sure that DBALL is on fast disk
- If you want just data recovery RESTART use scr=mini
  - Location of DBALL less critical
- If you don’t want to RESTART use INIT MASTER(S) (default in FEMAP)

Anatomy of Nastran Direct Static Solution

1. Generate elemental matrices KELM, KDICT (EMG)
2. Assemble elemental matrices KGG (EMA)
3. Reduce down to L-set KLL (MCE1/MCE2, etc.)
4. Solve equations KLL*UL=PL (DCMP and FBS)
5. Expand results to G-set UG (SDR1)
6. Calculate outputs OUGV, OES etc. (SDR2)

For most problems the solve step is the most expensive
(only step that Nastran runs in parallel using SMP)

The reduction and expansion steps (3 and 5) are largely matrix multiplies.
(For some solutions these can become expensive)

Output processing (step 6) can get expensive if LOTs of outputs are requested
(e.g. GPFOR or STRESS on entire model)

Anatomy of .f04 File for a Direct Static Solution

Clock Time (min:sec)

Total I/O

Module I/O

Total CPU (sec)

Module CPU (sec)

```
Day_Time  Elapsed     I/O_Mb      Del_Mb     CPU_Sec    Del_CPU    SubDMAP Line (S)SubDMAP/Module

14:48:35     0:00      144.0         3.0         0.1        0.0    XSEMDR   BGN
                                                                   …
14:53:00     4:25      143.6G        0.0       249.1        0.0    SEMG    85      EMG     BEGN
                                                                   …
15:49:46    61:11     1345.8G        1.0      3614.4        0.0    SEKRRS  166     DCMP    BEGN
*** USER INFORMATION MESSAGE 4157 (DFMSYN)
    PARAMETERS FOR PARALLEL SPARSE DECOMPOSITION OF DATA BLOCK KLL     ( TYPE=RSP )  FOLLOW
                     MATRIX SIZE = 20757327 ROWS             NUMBER OF NONZEROES =   651129790 TERMS
          NUMBER OF ZERO COLUMNS =         0       NUMBER OF ZERO DIAGONAL TERMS =        0
                    SYSTEM (107) =   32784                      REQUESTED PROC. =     16 CPUS
               CPU TIME ESTIMATE =  108111 SEC                I/O TIME ESTIMATE =    107 SEC
      MINIMUM MEMORY REQUIREMENT =  363328 K WORDS             MEMORY AVAILABLE = 1045357 K WORDS
    MEMORY REQR'D TO AVOID SPILL =  414682 K WORDS         MEMORY USED BY BEND  =  919646 K WORDS
    EST. INTEGER WORDS IN FACTOR  =   6427873 K WORDS         EST. NONZERO TERMS =  13189781 K TERMS
    ESTIMATED MAXIMUM FRONT SIZE  =   12015 TERMS                 RANK OF UPDATE =      32
                                                                   …
16:15:39    87:04     1541.3G      195.6G    19410.3    15795.9    SEKRRS  166     DCMP    END
                                                                   …
16:15:39    87:04     1541.3G        1.0     19410.3        0.0    FBS BEGN
16:20:32    91:57     1741.6G      200.2G    20004.0      593.7    FBS END
                                                                   …
16:26:26    97:51     1848.9G        0.0     20339.8        0.0    SESTATIC407     EXIT    BEGN
```

Matrix Generation

Matrix Decomposition

Maximum Memory Required (8*414682/1024^{2}=3Gbytes)

4 for LP version

Largest Influence on Run Time

Matrix Forward/Backward Substitution

What Can be Done to Speed up Direct?

Solution time is most sensitive to FRONT SIZE

Defaults work OK for most/many problems

Try modifying DCMP reordering algorithm (default is BEND (EXTREME), but METIS is sometimes better)

- NASTRAN DCMPSEQ=8 (or 9)
- Check to see if FRONTSIZE decreased

Use scr=mini to store LLL on scratch rather than DBALL (also results in much smaller DBALL and is the default we use in our .RCF file for the HPC machine)

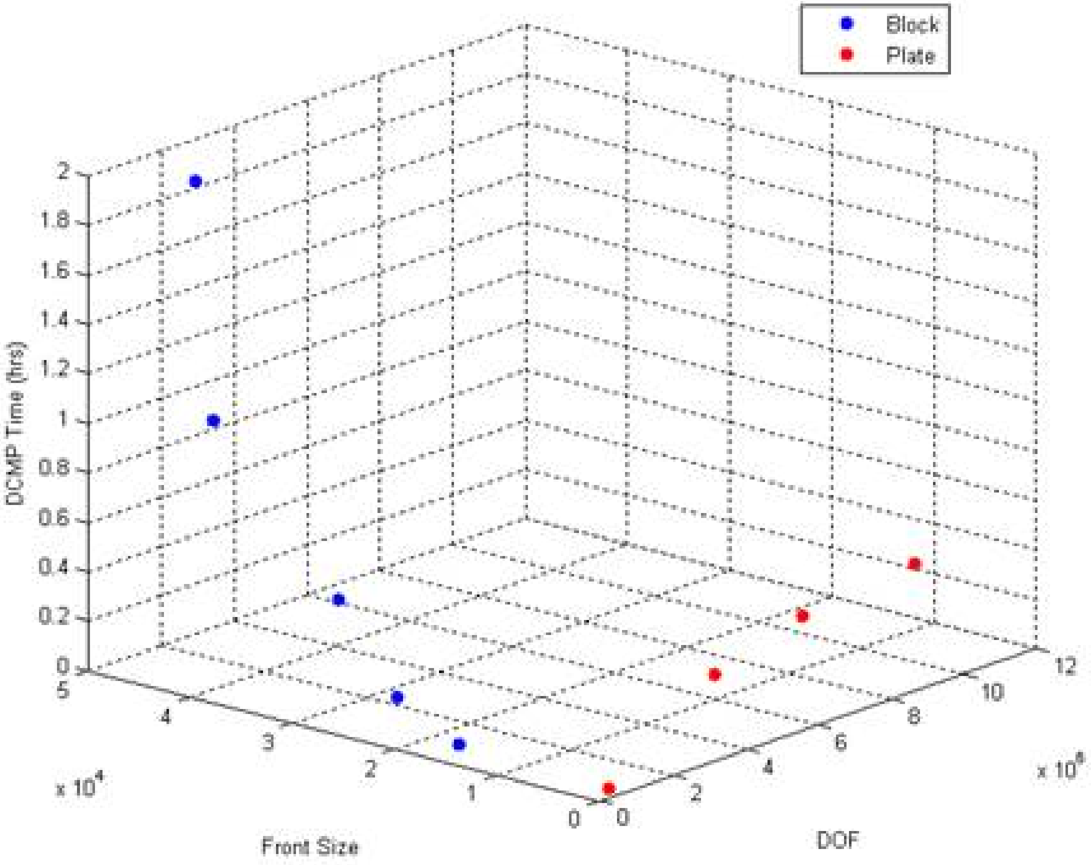
- This only helps speed is SCRATCH file has faster I/O than DBALL

Use larger value of BUFFSIZE (Can set this in the .rcf file by default)

Give Nastran the most amount scratch memory (smem) you can

- smem comes out of mem so to get 190 GBytes of smem and 16 GBytes of memory you need mem=206GB, smem=190GB (new SNAS script does this)
- In our experience Nastran is able to get about 80% of available RAM on Linux
- If you don’t have enough RAM it will spill to disk. This can be slow if scratch disk is slow.

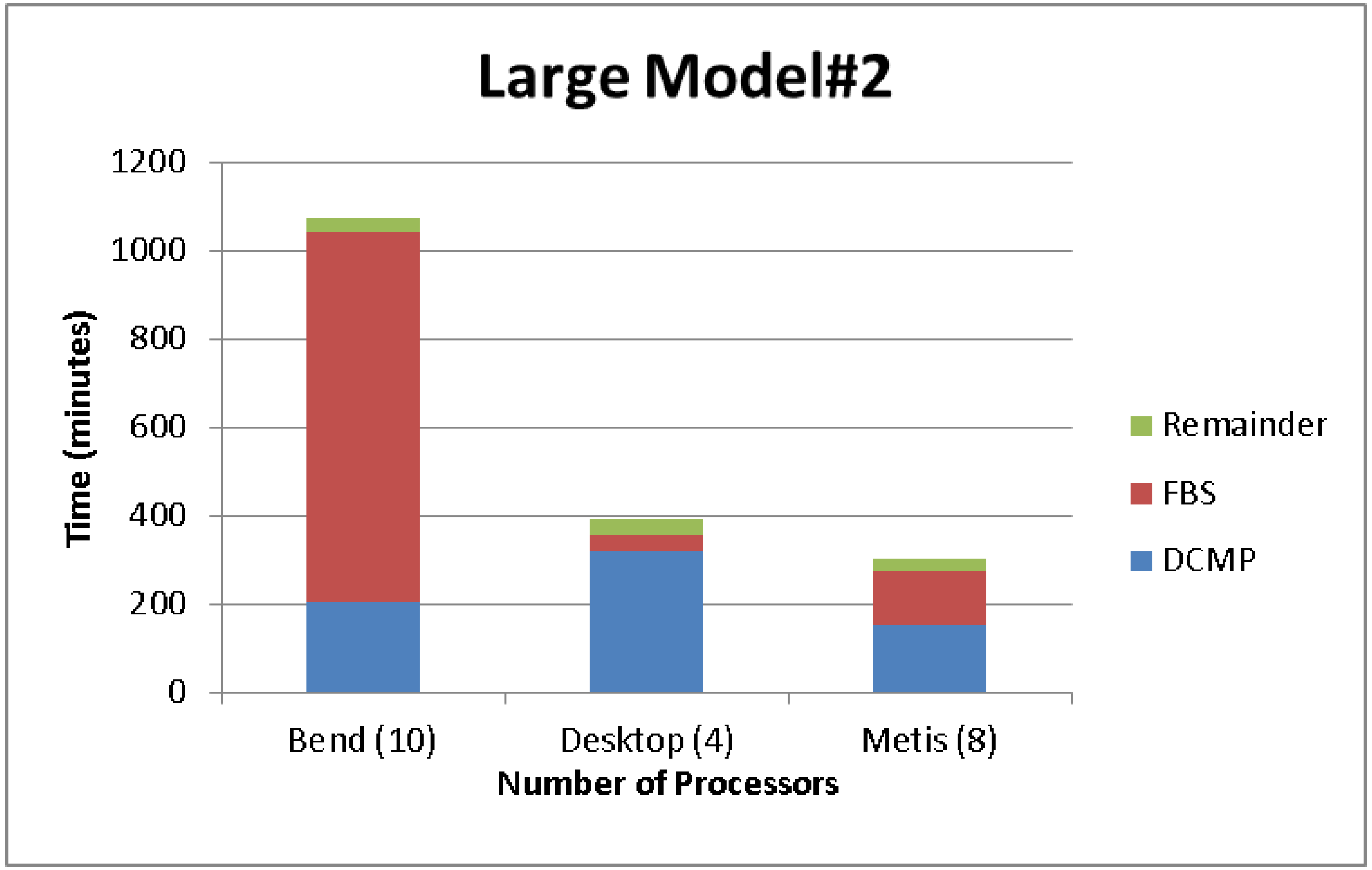
Solution Time Much More Sensitive to Front Size Than # DOF



- “Blocky” models tend to have larger Front Size for given # of DOF
- Run time is fairly low with > 10 million DOF as long as Front Size remains low
- Run time increases very rapidly with increasing Front Size even with relatively low number of DOF

$$Time^{0.222} = 0.04085DOF + 23.8FrontSize$$

Example Where Switching Reordering Helped



Forward-Backward Substitution was really slow with default Bend ordering.

Default BEND ordering – FS = 57,669

METIS ordering – FS = 51,948 (10% less)

(DCMPSEQ=8, or 9)

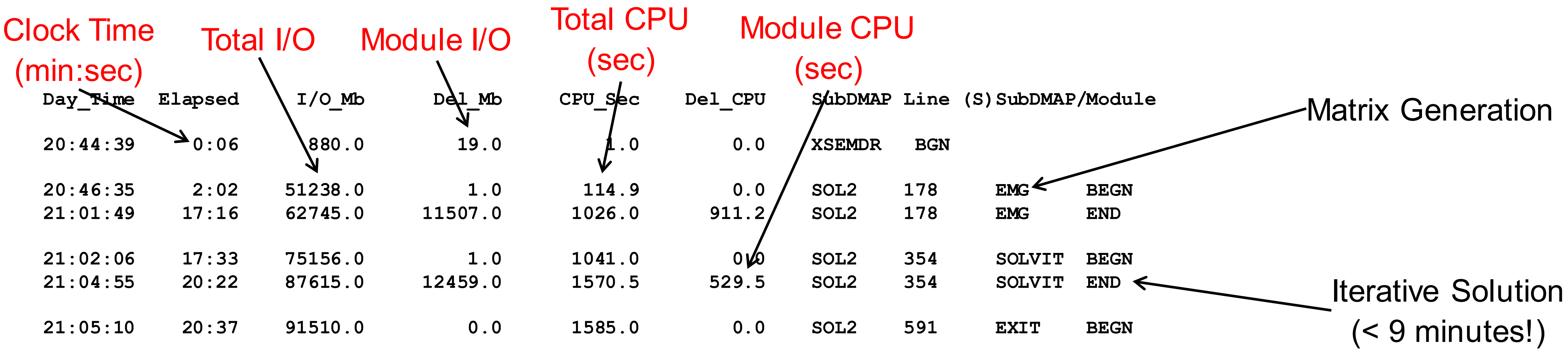
Run time decreased by > 3x with fewer processors

Performance is VERY job specific

Changing reordering is something to try

Iterative Solver Can Sometimes be MUCH Faster

- Iterative solution is usually fastest way of speeding up static solution (for small number of load cases)
- Skips matrix assembly, decomposition and forward/backward substition
- Longest step is typically matrix generation (EMG)
- e.g. large solid model (front size=53,532) with 1 load case reduced from 226 to 20 minutes
- NASTRAN ITER=YES,ELEMITER=YES
- ITER,100,ITSEPS=1E-10 (improves accuracy at very little extra cost)



Clock Time (min:sec)

Total I/O

Module I/O

Total CPU (sec)

Module CPU (sec)

```
 Day_Time   Elapsed       I/O_Mb       Del_Mb     CPU_Sec     Del_CPU    SubDMAP  Line  (S)SubDMAP/Module

 20:44:39      0:06        880.0         19.0         1.0         0.0    XSEMDR    BGN

 20:46:35      2:02      51238.0          1.0       114.9         0.0    SOL2     178       EMG     BEGN
 21:01:49     17:16      62745.0      11507.0      1026.0       911.2    SOL2     178       EMG     END

 21:02:06     17:33      75156.0          1.0      1041.0         0.0    SOL2     354       SOLVIT  BEGN
 21:04:55     20:22      87615.0      12459.0      1570.5       529.5    SOL2     354       SOLVIT  END

 21:05:10     20:37      91510.0          0.0      1585.0         0.0    SOL2     591       EXIT    BEGN
```

Matrix Generation

Iterative Solution (< 9 minutes!)

Modal Solution Typically Dominated by READ Step

```
Day_Time  Elapsed     I/O_Mb      Del_Mb     CPU_Sec    Del_CPU    SubDMAP Line (S)SubDMAP/Module

14:42:13      0:01      487.0         9.0         0.1        0.0   XSEMDR   BGN

14:42:22      0:10     3798.0        58.0         9.2        0.1   SEMG    118     EMG     BEGN
14:42:38      0:26     4751.0       953.0        24.0       14.8   SEMG    118     EMG     END

14:43:22      1:10    22862.0         2.0        55.6        0.0   XREAD   251     READ    BEGN
*** USER INFORMATION MESSAGE 4157 (DFMSYN)
    PARAMETERS FOR SPARSE DECOMPOSITION OF DATA BLOCK  KXX     ( TYPE=RDP ) FOLLOW
                     MATRIX SIZE =  1919657 ROWS            NUMBER OF NONZEROES =    47382024 TERMS
          NUMBER OF ZERO COLUMNS =        0        NUMBER OF ZERO DIAGONAL TERMS =        0
               CPU TIME ESTIMATE =     838 SEC                I/O TIME ESTIMATE =        1 SEC
       MINIMUM MEMORY REQUIREMENT =   26225 K WORDS             MEMORY AVAILABLE =  229403 K WORDS
    MEMORY REQR'D TO AVOID SPILL =   44234 K WORDS          MEMORY USED BY BEND  =   56574 K WORDS
    EST. INTEGER WORDS IN FACTOR =    218629 K WORDS          EST. NONZERO TERMS =    464643 K TERMS
    ESTIMATED MAXIMUM FRONT SIZE =    3290 TERMS                 RANK OF UPDATE =      32
14:43:28      1:16    25874.0      3012.0        62.6        7.0   SPDC BGN TE=838
14:44:10      1:58    32540.0      6666.0       101.7       39.1   SPDC END

*** USER INFORMATION MESSAGE 5403 (LNNRIGL)
   BREAKDOWN OF CPU TIME (SEC) DURING LANCZOS ITERATIONS:
   OPERATION                  REPETITIONS   AVERAGE      TOTAL
   FBS (BLOCK SIZE=  7)               52       5.18     269.49
   MATRIX-VECTOR MULTIPLY            117       0.02       2.64
   SHIFT AND FACTOR                    3      39.32     117.95
   LANCZOS RUN                         2      54.61     109.22

14:56:28     14:16      507.0G        0.0       815.6        0.0   SEMODES 492     EXIT    BEGN
```

Matrix Generation

Maximum Memory Required

(4*44234/1024^2=172 Mbytes)

8 for ILP version

Most time typically in FBS

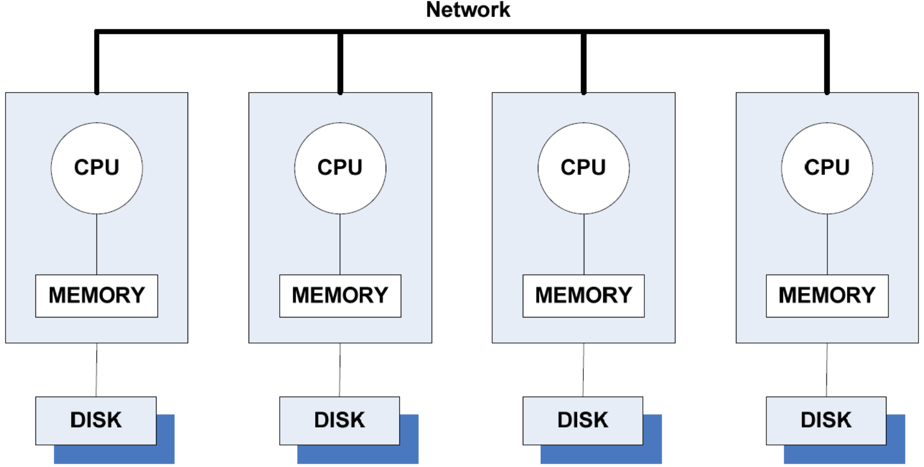
What Have We Learned So Far?

- For Nastran I/O is king
- I/O to SCRATCH and SCR300 files is typically most important
  - Amount of I/O to DBALL file controlled by scr keyword (yes, no, mini)
- RAM is fastest I/O → Using smem to allocate RAM to scratch can be VERY effective
  - Only works with LARGE amounts of RAM (256 GBytes/node on our HPC cluster)
- If you don’t have enough RAM, fast (i.e. SSD) local disk is critical
- You typically need much less RAM to solve than you might think
  - Throwing lots of RAM without using smem is typically **not** effective
  - Look at “minimum to avoid spill” and round up to get reasonable RAM
  - Remaining RAM is best left to OS or used for smem
- If direct static solution seems long try METIS ordering (DCMPSEQ=8 or 9)
- Try using iterative solver for statics with small number of load cases
- Use .f04 file to understand where job is using time
  - EMG, (DCMP for statics and READ for modes) good points to look at solution
- No one set of options appears to be optimal for all problems
  - Defaults are good for most problems

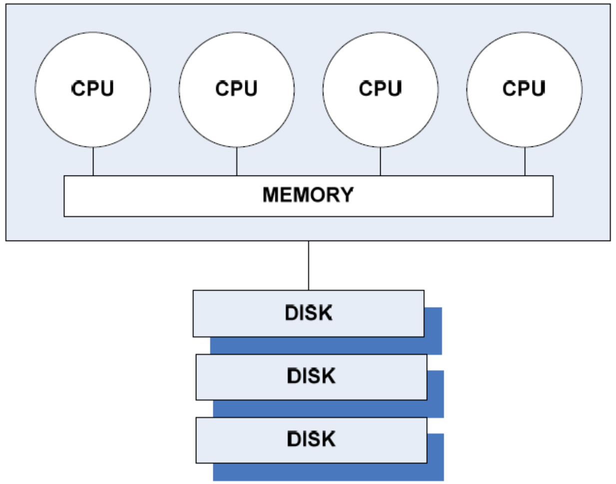
SMP vs DMP in NX Nastran

- SMP works on any computer with multiple processors
  - Including almost all desktops
  - A single node of an HPC cluster
  - smp = n, or parallel = n
  - Fine grain parallelism only applies to specific steps
    - DCMP, FBS, SOLVIT
  - Easy, but of limited benefit
  - Limited benefit above n=4 in our experience
- DMP works on a network of computers
  - Typically multiple nodes on HPC cluster
  - dmp = n (nrec = n, dstat = n, numseg = n, nclust = n)
  - Coarse grain parallelism divides problem into multiple sub-problems
- SMP and DMP can be combined
  - Subdivide at a coarse level, and have each sub-problem solve faster with SMP

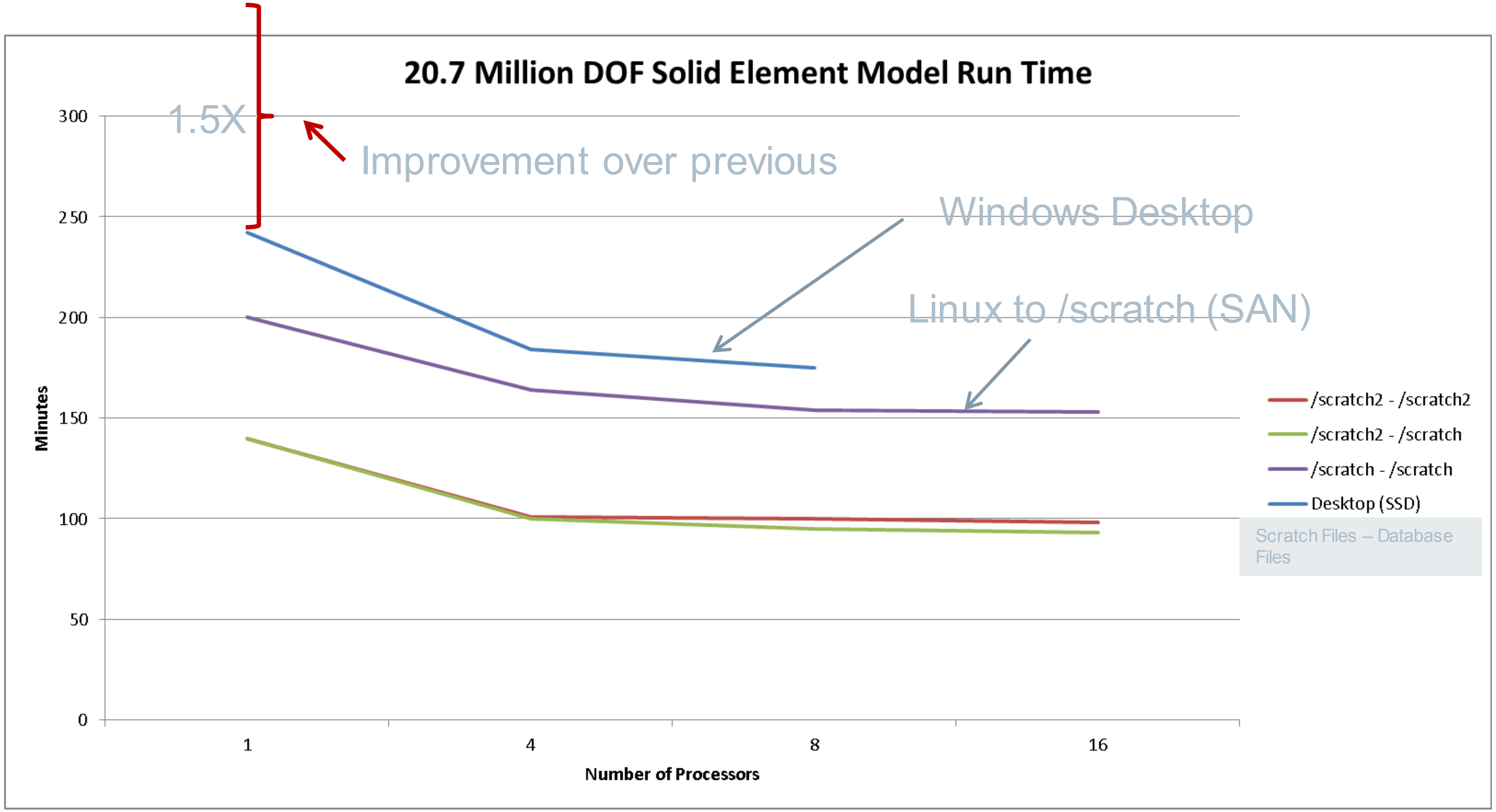
Distributed Memory Parallel (SMP)



Shared Memory Parallel (SMP)



Example of SMP Performance



1.5X

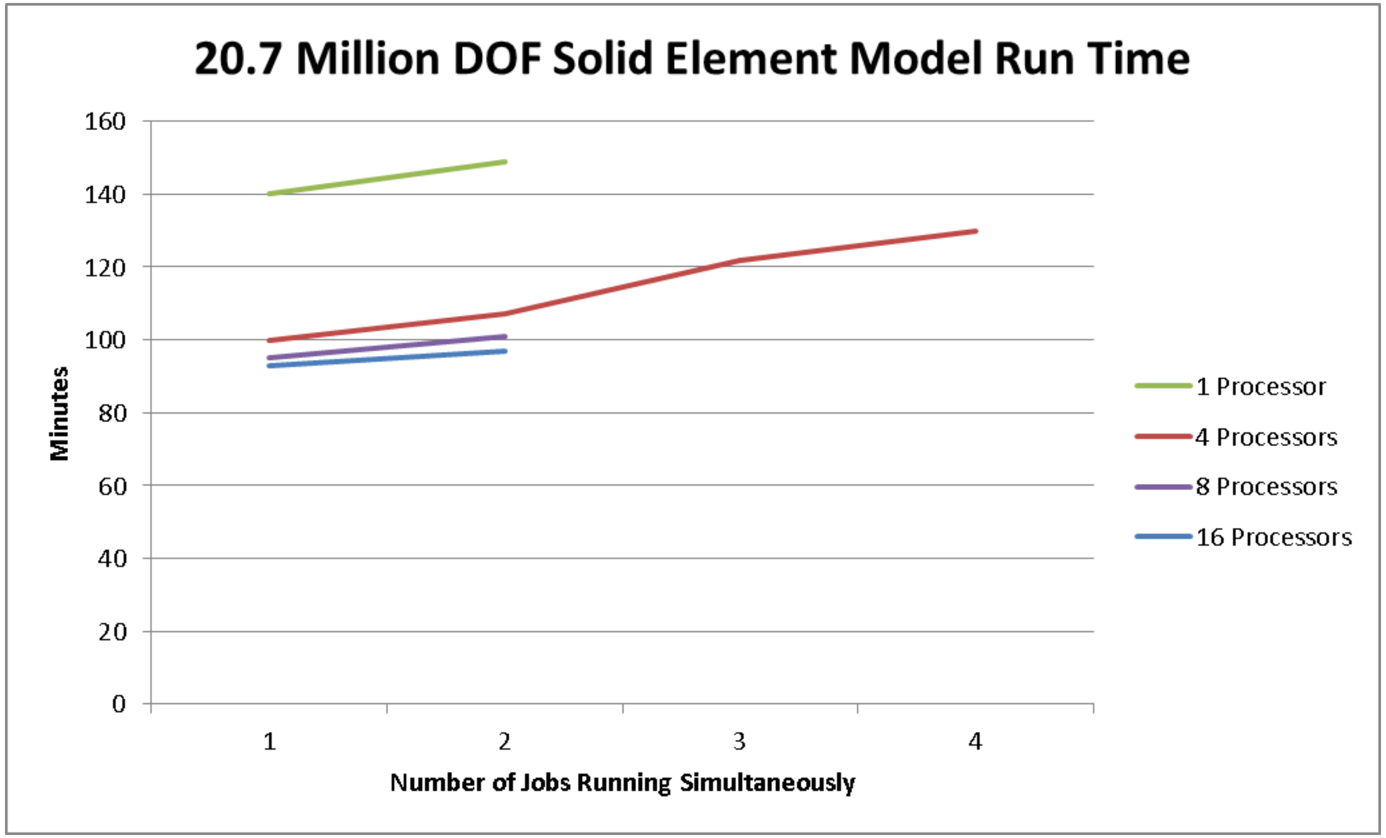
Improvement over previous

Windows Desktop

Linux to /scratch (SAN)

- Performance is VERY problem specific
- Speedup is rarely anywhere close to linear
- Throughput usually better with multiple jobs
  - Within limits
- Performance more sensitive to disk than smp
- Don’t be fooled by looking at CPU usage during run

Multiple Runs on Single Node Provides Better Throughput



Performance very job specific

(single job run time optimal with 1 job)

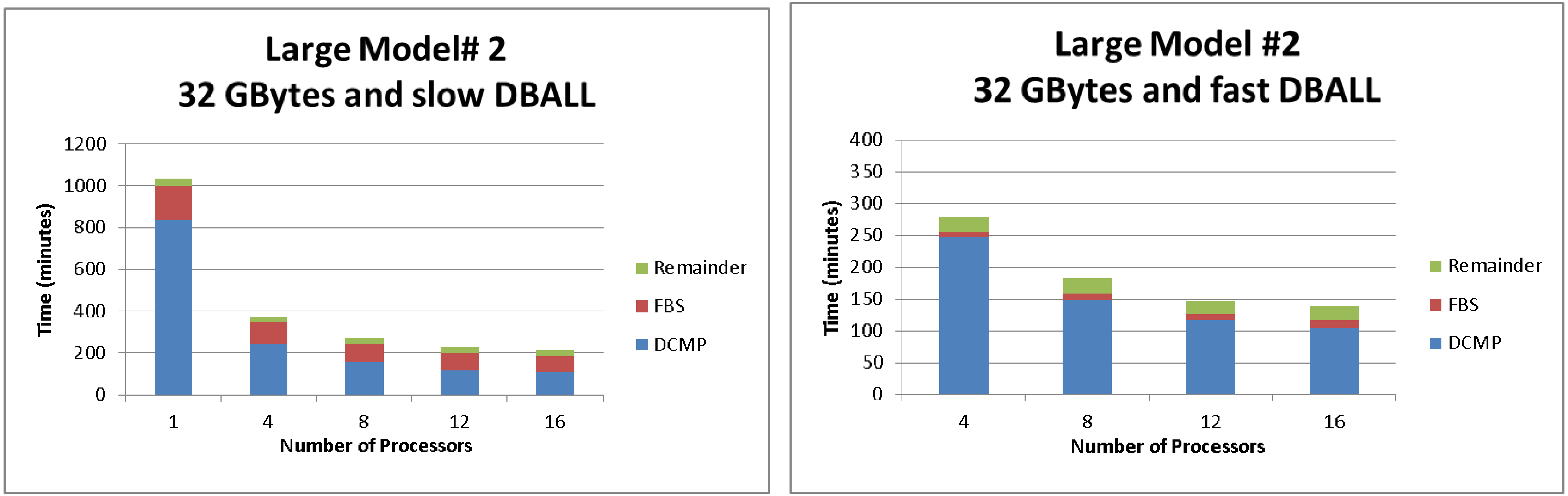
~10% penalty for 2 jobs

~20% penalty for 3 jobs

~30% penalty for 4 jobs (total throughput optimal with 4 jobs)

- Assumes number of licenses not an issue

Different Model Scales Better with Processors

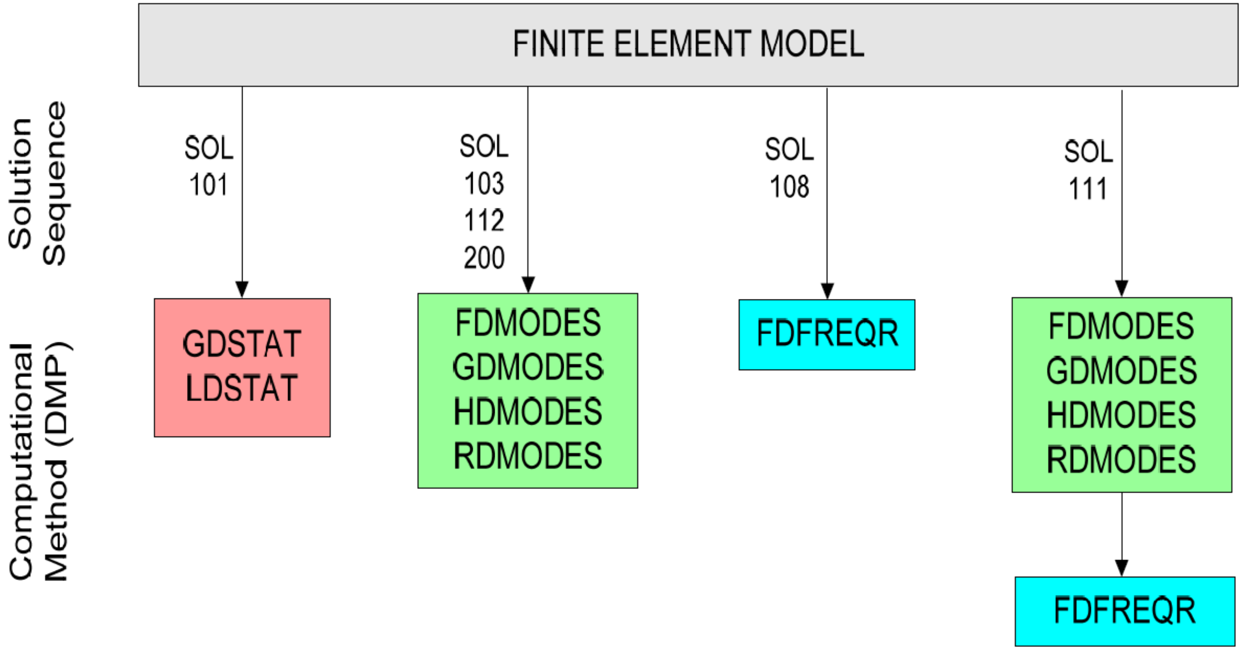


New Conclusion: for this Class of Model (Higher Front Size)
When the Front Size is High, the ordering and the DBALL speed and the number of processors matters!

Lots of Different DMP Methods in NX Nastran

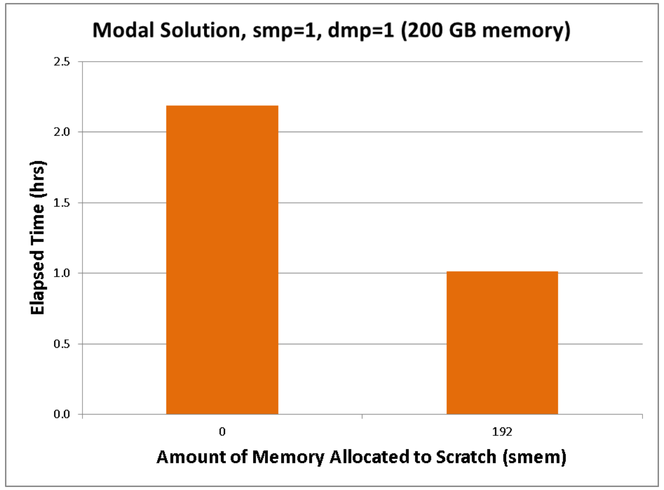
Solution	Method	Command	Suggested Model Type
101	GDSTAT	dmp=p	Large models
101	LDSTAT	dmp=p, dstat=1	Smaller models, large number of laods
103	FDMODES	dmp=p, numseg=p	Small models, large number of frequencies
103	GDMODES	dmp=p	Large models, small number of frequencies
103	HDMODES	dmp=p, nclust=c	Large models, large number of frequencies
103	RDMODES	smp=p, nrec=n	Large models, approximate solution
108	FDFREQR	dmp=p	All models

- Pre-requisites for DMP
  - NXN on all hosts
  - MPI (mpirun) available
  - Input data file visible to all hosts
  - rsh, rcp, rlogin must work to communicate among nodes
  - scp and ssh must work to remotely log into nodes



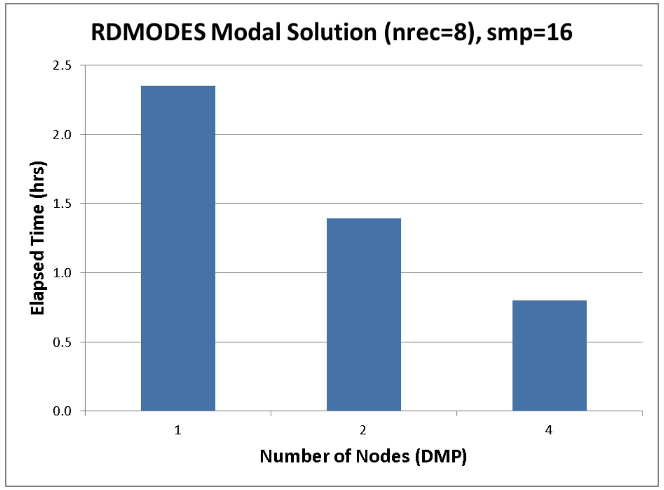
Results for RDMODES/DMP Solution (NXN 10 pre-release)

Allocating RAM to smem Halves Run Time

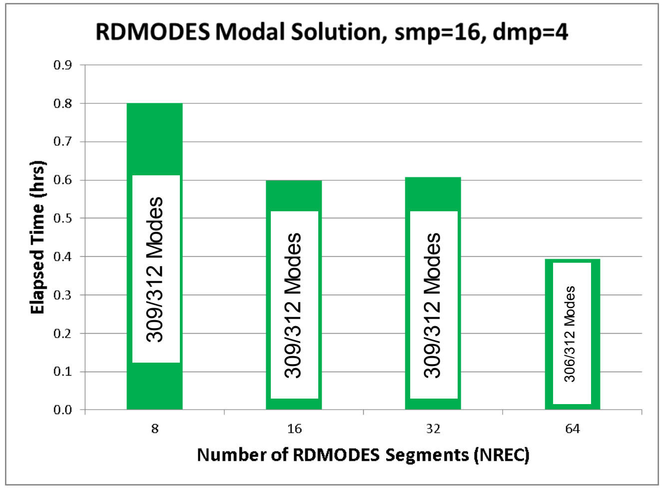


- RDMODES is an approximate method for calculating modes
  - Breaks model into successively smaller pieces (AMLS)
    - Number of pieces controlled by nrec keyword (128 – 256)
  - rdscale keyword increases accuracy with computational cost
  - Works particularly well with DMP (doesn't need to be DMP)
  - Can be used for modal transient (109) or FRF (112)

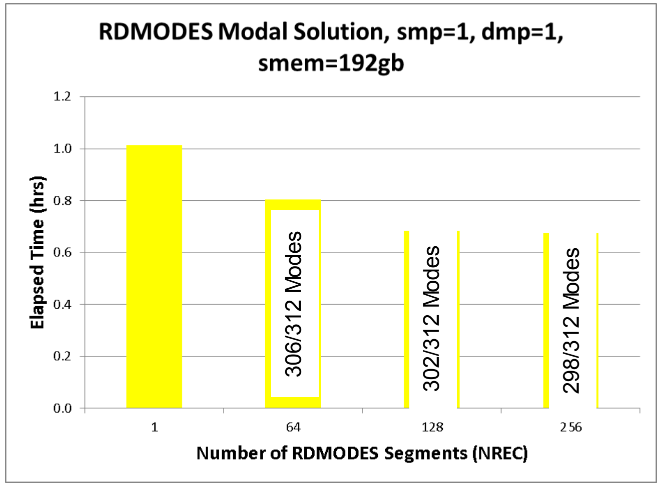
Using more DMP nodes is more beneficial



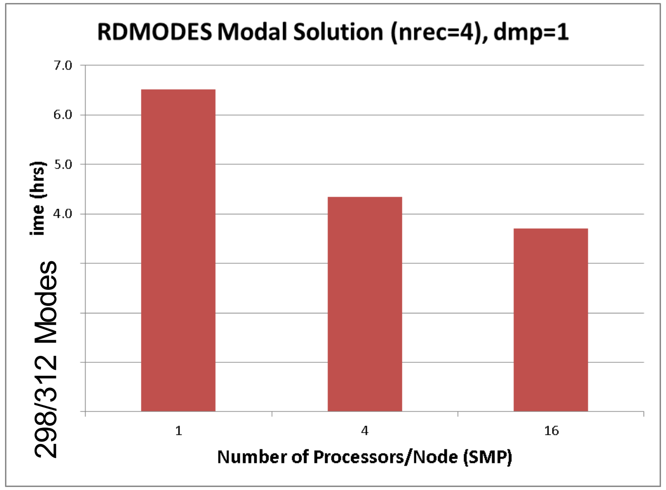
Using more RDMODES segments is beneficial



Some Benefit to RDMODES without DMP



Using more SMP processors has marginal benefit



Summary

- Biggest performance gain in NX Nastran is through fast I/O
  - RAM (smem) fastest followed by local SSD
- Learn to read the .f04 file to understand performance
- Lots of solution options can be used to improve performance
  - Consider iterative solution for statics with small number of load cases
  - Consider RDMODES for modes independent of SMP/DMP
- SMP is fine grain parallelization on a single node
  - Applies equally well to a desktop
  - Very easy to use
  - Provides some advantage but doesn’t scale well
- DMP is coarse grain parallelization across multiple nodes on a cluster
  - Only applicable on a cluster
  - Much more difficult to use (many different options)
  - Can scale much better than SMP