ENHANCEMENTS TO THE IBM VERSION OF COSMIC/NASTRAN

by

W. Keith Brown
RPK Corporation
Hayes, Virginia

SUMMARY

Major improvements have been made to the IBM version of COSMIC/NASTRAN by RPK Corporation under contract to IBM Corporation. These improvements will become part of COSMIC's IBM version and will be available in the second quarter of 1989. The first improvement is the inclusion of code to take advantage of IBM's new Vector Facility (VF) on its 3090 machines. The remaining improvements are modifications that will benefit all users as a result of the extended addressing capability provided by the MVS/XA operating system. These improvements include the availability of an in-memory data base that potentially eliminates the need for I/O to the PRIxx disk files. Another improvement is the elimination of multiple load modules that have to be loaded for every link switch within NASTRAN. The last improvement allows for NASTRAN to execute above the 16 mega-byte line. This improvement allows for NASTRAN to have access to 2 giga-bytes of memory for open core and the in-memory data base.

INTRODUCTION

Very few changes have been made to the IBM version of COSMIC/NASTRAN in the last few years in order to take advantage of new hardware capabilities and new MVS/XA operating system features. One of IBM's new hardware capabilities is the Vector Facility that provides significant CPU time reductions for programs with vector operations (Reference 1). Use of IBM's new Vector Facility allows NASTRAN to solve larger problems in a much faster manner. Problems that spend a large amount of CPU time in symmetric decomposition, matrix multiplication, forward/backward substitution and eigenvalue analysis could greatly benefit from use of the Vector Facility. NASTRAN has been modified to take advantage of the Vector Facility in these areas.

With the release of MVS/XA, IBM allowed users to reference up to 2 giga-bytes of memory in a given job step. However, NASTRAN could only run under the 16 mega-byte line because of the assembly language code and the memory management design. Of the 16 mega-bytes, the most a user could get was about 8 mega-bytes because of the operating system. Although 8 mega-bytes was probably acceptable for open core, it was insufficient to contain any in-memory data files. NASTRAN has now been modified to allow it to execute above the 16 mega-byte line and this in turn allows access to a maximum of 2 giga-bytes of memory.

With access to 2 giga-bytes of memory, NASTRAN can now have the option of keeping DMAP files in memory. These files were previously written to the PRIxx files using BSAM I/O. This in-memory capability has been implemented and the implementation automatically allows for the use of external files when memory for the in-memory files has been exhausted. The end result of this

feature is that job turnaround will be improved because of the reduction of disk I/O.

Lastly, NASTRAN was previously delivered as 16 load modules. One load module was always resident. The other 15 load modules were loaded into memory when needed but only one could reside in memory at a given time. If a new load module was needed, the current memory-resident load module would be deleted and the new load module loaded in its place. Therefore, as NASTRAN was processing the job, load modules would be loaded and then deleted to allow for other load modules. This resulted in lost time. Though the time that was lost was not appreciable, NASTRAN is now designed to eliminate this reloading procedure and all of NASTRAN now remains completely in memory.

EXECUTION TIME OPTIMIZATION CONSIDERATIONS

There are several considerations that a user should be aware of when trying to setup his NASTRAN run for optimal execution time. First, a user should be aware of the amount of open core that he may need. If open core is not large enough, users may experience severe execution time degradation due to possible spilling during decomposition or multiple passes on matrices used in matrix multiplication or forward-backward substitution. There is no single formula that may be used to determine the amount of open core needed for all cases; however, the following formulas provide some rough estimates.

Size = 6 * (larger of (number of degrees of freedom in the a-set)
or (number of degrees of freedom in the o-set))
* (number of output cases (e.g., number of loads or number of eigenvalues))

Size = .04 * (square of degrees of freedom in problem)

The above formulas are crude and should only be used as rough estimates. Users should check all symmetric decomposition messages to determine if any spill groups were required and also all matrix multiply-add (MPYAD) messages to determine if multiple passes were required. If either spill groups or multiple passes were required, then open core should be increased.

In addition to the amount of open core given to a problem, all problems will benefit from the use of the in-memory data base. The size of the in-memory data base is controlled by the REGION size given by the user, the memory to be returned to the operating system and the memory to be used for open core. The amount of memory to be used by the in-memory data base is computed as follows:

DB Size = (Region Size) - (Open Core Size) - (Operating System Memory)
- (NASTRAN Load Size)

(Note, the size of the NASTRAN load module is approximately 8000k bytes)

The in-memory data base capability eliminates much of the I/O performed by NASTRAN. This results in faster turnaround and faster execution. CPU savings will be of the order of about 5% for most problems.

USE OF IBM's VECTOR FACILITY

Efficiency improvements come also from the use of IBM's new Vector Facility. IBM's new Vector Facility allows programs with vector operations to use vector instructions for faster CPU execution (Reference 2). Inherent in the use of vector hardware is the length of the vector(s) to be processed. Due to the startup time and other associated overhead that goes with vector processing, vectors must be of a certain minimum length before CPU gains can be realized. In some cases where the vectors are very short, degradation can occur and longer CPU times can result. In general, vectors should have a length of 10 elements or more for real CPU gains to be realized. This is true for the majority of problems in NASTRAN.

NASTRAN has been optimized for vectorization in the following areas: symmetric decomposition, forward-backward substitution, eigenvalue analysis (Givens, Inverse Power and Feer) and matrix multiplication (Reference 3). The gains to be realized to a user are dependent upon the amount of the total CPU time that is spent in these areas. For most CPU intensive runs, analysis shows that these are the areas where most of the CPU time is used.

The modifications to NASTRAN took advantage of IBM's new Engineering and Scientific Subroutine Library (ESSL) (Reference 4). This library is a set of high performance mathematical subroutines that can be used by higher level languages such as Fortran. Use of the ESSL will ensure users of optimal performance in the future as the hardware characteristics of the Vector Facility may change.

Figure 1 shows improvements that were made on a statics, a normal modes and a frequency response problem using the optimized version of NASTRAN. The jobs were run on an IBM 3090E computer. A decrease of approximately 10% of the total CPU time could be realized if these problems were run on an IBM 3090S computer. The vector affinity column gives a measure of how much of the CPU time was spent in vector computations. Efficiency gains were realized from the vector code, the ability to have larger open core (eliminating multiple passes in matrix multiplication and forward-backward substitution) and the use of the in-memory data base (eliminating much of the I/O).

Depending on the problem characteristics, percentage improvements can vary greatly. Therefore, there is no hard and fast rule that users can use to show improvements. However, Figure 2 shows percentages that may be obtained during various matrix operations in NASTRAN. These figures are given only so the user can have some idea of expected improvements using the Vector Facility.

USE OF IBM's EXTENDED ARCHITECTURE (MVS/XA)

Prior to the release of IBM's MVS/XA operating system, the MVS operating system was based on 24-bit addressing. This meant that any given program could not address more than 2**16 bytes or 16 mega-bytes of memory. Of this 16 mega-bytes, approximately 8 mega-bytes were available to a program because of the operating system requirements. As software developments proceeded, especially in the area of graphics, and as user problems to be solved became larger, it became apparent that there was a need for programs to have access to more memory. With the coming of MVS/XA, IBM switched to 31-bit addressing and this allowed for 2**31 or 2 giga-bytes of memory

to be available. Many of the IBM-supplied compilers were modified to take advantage of this and, most noticeably for NASTRAN, was the IBM VS Fortran compiler (Reference 5). Users could now write programs in Fortran and take full advantage of the 31-bit addressing capability (Reference 6).

The 16 mega-byte line that resulted from 24-bit addressing still exists in a limited sense for programs that have assembly language subroutines. Assembly language subroutines that perform I/O functions must execute in 24-bit addressing mode with the possible exception of subroutines that use execute channel program instructions (EXCPs) directly. Programs that have assembly language I/O must be designed to take this into account. The assembly code and the I/O buffers must reside below the 16 mega-byte line. In addition to the I/O considerations, sometimes assembly language codes have other 24-bit design dependent considerations that require that they execute in 24-bit addressing mode.

The previous IBM version of NASTRAN had a very small percentage of assembly language subroutines (Reference 7). These subroutines were required to allow for dynamic loading of NASTRAN load modules, to optimize computationally bound codes, to perform memory management and to provide efficient random access I/O. Because of these codes, NASTRAN required modifications to allow it to execute in 31-bit addressing mode. The IBM version of COSMIC/NASTRAN has now been redesigned to take advantage of 31-bit addressing. The new design resulted in two load modules. One load module is called NASTRAN and it executes above the 16 mega-byte line. This load module contains all of the analysis code and is by far the larger module. The other load module is called IO and it executes below the 16 mega-byte line. This module does all the non-Fortran I/O functions required by NASTRAN. Open core and the new in-memory data base reside above the 16 mega-byte line. Figure 3 shows this design.

New assembly language programs were created to allow for this design. The main program in the NASTRAN load module is the assembly language program NASTRAN. The main program in the IO load module is the assembly language program IO. The functions of the NASTRAN assembly language program are given below:

1. Reads and processes the job step parm. The format of the job parm has been changed and will be described below.

2. Initializes the Fortran run-time environment. The job parm is also passed by NASTRAN to Fortran during initialization to allow users the ability to take advantage of the new Fortran job parms that became available with Fortran Version 2. One such Fortran parameter that is recommended for use is the "NOXUFLOW" parameter that suppresses a program interrupt from a floating point underflow and allows for a hardware fixup instead of a software fixup.

3. Performs memory management. The REGION value as given on the job step EXEC card specifies the maximum amount of memory that NASTRAN can obtain. All memory that is available after the loading of the program is obtained. Memory is then released for the operating system use and the remaining memory is used by NASTRAN for open core and the in-memory data base.

4. Loads the IO load module and initializes the interface between the two. This allows for communication to all I/O subroutines that must reside below the 16 mega-byte line.

The IO program provides all of the non-Fortran I/O for the NASTRAN data files.

The job step parm has been modified for this new level of NASTRAN. The format is as follows:

```
// EXEC     PGM=NASTRAN,
// PARM='NOXUFLOW,OSMEM=200K,OCMEM=7000K,DBMEM=100000K'
```

The parameters are defined as follows:

NOXUFLOW	Parameter to Fortran to suspend underflow interceptions.
OSMEM	Amount of memory to free back to the operating system.
OCMEM	Amount of memory to allocate for open core.
DBMEM	Amount of memory to allocate for the in-memory data base. A non-zero value implies that memory is to be allocated for the in-memory data base. A zero value eliminates the use of the in-memory data base.

The last three parameters determine how memory is to be allocated during the run. NASTRAN first obtains all memory available based on the job step REGION value. It will then release back to the operating system the amount of memory as specified on the OSMEM parameter. The memory that remains will all be given to open core if DBMEM=0. Otherwise, the amount of memory as specified by the OCMEM parameter is designated for open core and all that remains is used for the in-memory data base. If the memory for the in-memory data base is greater than that specified on DBMEM parameter, the larger value is used. If there is a lesser amount of memory available than that specified on the DBMEM parameter, then the lesser amount is designated. If there is only sufficient memory for open core, then no memory is allocated for the in-memory data base regardless of the value of DBMEM.

NASTRAN prints a summary of the load addresses and the memory allocations at the beginning of the NASTRAN log file. A sample listing is shown in Figure 4.

IN-MEMORY DATA BASE

The in-memory data base will benefit users in job throughput and to a limited sense in CPU utilization. Job throughput will increase because of the elimination of I/O to disk. The CPU savings will be of the order of 5%. The in-memory data base is based on blocks of memory that are chained together. There is one chain for free space. The block sizes in the free chain will vary. There is also one chain for each DMAP file. Each DMAP file is a file in the in-memory data base. The size of the blocks allocated for the DMAP files is based upon the size specified for the NASTRAN GINO files. Blocks of memory are allocated for a DMAP file as the file is written and pointers are maintained as to the current block being processed. Memory that is available for allocation of the blocks is maintained by the free chain. As files are opened for write with rewind, the previously allocated memory data blocks to a DMAP file are released back into the free chain and a new DMAP file chain is established. Savings are realized because there is no I/O taking place and also there is no moving of data. All data being written into the DMAP file are written directly into the allocated memory block with no secondary transfer of data.

The in-memory data base is designed so that when there is insufficient memory for additional blocks of an existing DMAP file or for the creation of a new file, the external PRIxx, SECxx and TERxx files are used as spill. Users should be aware that the PRIxx, SECxx and TERxx DD cards are still needed, however the space allocations may need to be adjusted based on the amount of memory provided for the in-memory data base.

A directory of the in-memory data base is provided when DIAG 2 is turned on. The directory comes out after each DMAP module execution. Use of this DIAG is not recommended because of the large amounts of printout that it may generate. Users may opt to use the following technique to turn on DIAG 2 in order to get the in-memory data base directory at a specific point in the DMAP.

```
ALTER  n  $
PARAM //*DIAG*//2 $
ALTER  n+1 $
PARAM //*DIAGOFF*//2 $
ENDALTER $
```

where n is a DMAP instruction number.

A sample printout of the in-memory data base is given in Figure 5. The unit number defines the unit number allocated to the DMAP file in the NASTRAN File Allocation Table (FIAT) and will be used as the ''xx'' value to determine which PRIxx file is to be used for spill. The name field gives the DMAP file name and the current number defines the current block at which the file is positioned. For files that are closed with rewind, this value will be zero. For files that are closed without rewind, this value will be the last block that was either read (file opened for read) or was written (file opened for write). The in-mem blocks value defines the number of blocks allocated in the in-memory data base to the DMAP file and the disk blocks value defines the number of blocks that could not be contained in memory and were written to the PRIxx external file. The trailer values are the matrix trailers associated with the DMAP file.

CONCLUDING REMARKS

The IBM version of COSMIC/NASTRAN has been enhanced to take advantage of IBM's Vector Facility and the extended addressing capability provided by IBM's MVS/XA operating system. The enhancements include modifications to subroutines that can take advantage of the Vector Facility, modifications to allow NASTRAN to execute above the 16 mega-byte line and modifications to allow for a new in-memory data base. When all of the above features are used, users will be pleased at the performance increase. In addition, these modifications open up the door for larger problems to be analyzed on IBM that were previously not practical because of CPU requirements and/or open core requirements.

TYPE OF PROBLEM	G-SET	A-SET	SCALAR CPU (Sec.)	VECTOR CPU (Sec.)	VECTOR AFFINITY (Sec.)	VECTOR CPU/ SCALAR CPU
Statics	28828	11426	1264.3	672.3	520.2	53%
Normal Modes	8946	225	634.9	382.6	205.8	60%
Freq. Resp.	3852	3614	584.7	390.2	281.0	67%

Figure 1. Samples of Improvements in CPU Utilitzation
(Problems executed on IBM 3090E computer)

MATRIX OPERATION	VECTOR CPU / SCALAR CPU
Symmetric Decomposition	45%
Forward-Backward Substitution	33%
Matrix Multiply-Add Method 1	72%
Matrix Multiply-Add Method 2	53%
Matrix Multiply-Add Method 3	20%

Figure 2. Improvements in Matrix Computations

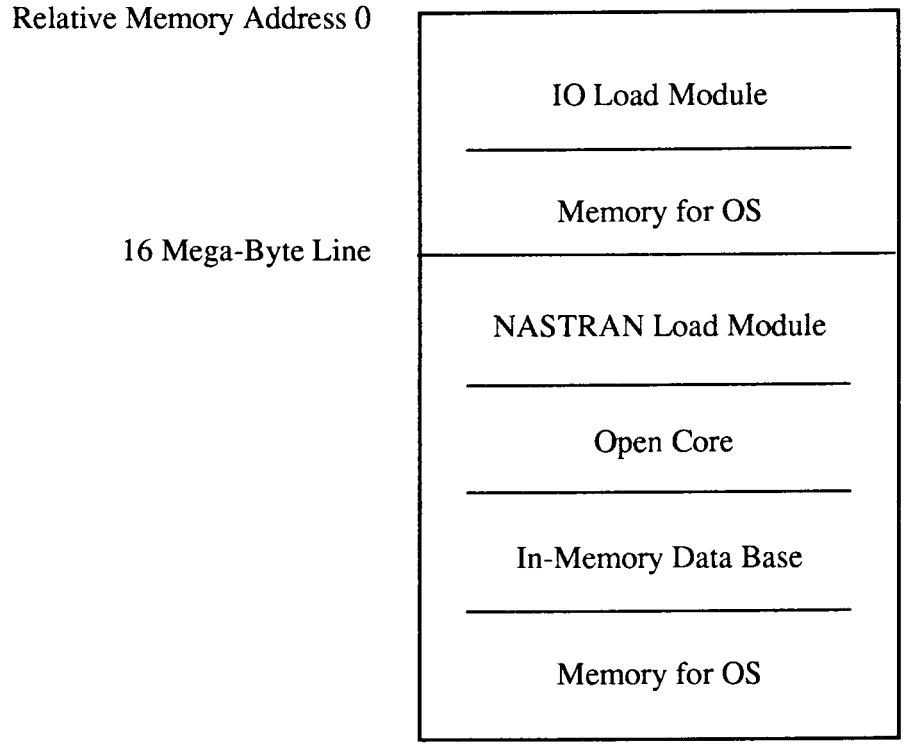

Figure 3. Memory Layout for IBM NASTRAN

```
ADDRESS OF BEGINNING OF NASTRAN LOAD MODULE = 04A009B0 HEX
ADDRESS OF END OF NASTRAN LOAD MODULE       = 0514D000 HEX
ADDRESS OF BEGINNING OF IO LOAD MODULE      = 8000CF30 HEX

ADDRESS OF MEMORY OBTAINED                  = 0514D000 HEX
LENGTH OF MEMORY OBTAINED                   = 0D17F000 HEX  219672576 DEC

MEMORY TO FREE FOR OS                       =  00032000 HEX     204800 DEC
MEMORY TO USE FOR OPENCORE                  = 005DC000 HEX     6144000 DEC
MEMORY TO USE FOR IN-MEM DATA BASE          = 0CB70FF8 HEX  213323768 DEC

ADDRESS OF BEGINNING OF OPENCORE            = 0514D000 HEX
ADDRESS OF END OF OPENCORE                  =  05729000 HEX
ADDRESS OF BEGINNING OF IN-MEM. DATA BASE   =  05729008 HEX
ADDRESS OF END OF IN-MEM. DATA BASE         =  12299FF8 HEX

(NOTE:  ALL UNITS ABOVE ARE IN BYTES)
```

Figure 4. Example of NASTRAN Printed Summary on Log File

MEMORY DATA BASE DIRECTORY
MAXIMUM ENTRIES= 100 CURRENT ENTRIES= 33

UNIT		NAME	CURRENT NUMBER	IN-MEM BLOCKS	DISK BLOCKS	TRAILER					
1	20	PHIG	0	41	0	14	8946	2	2	8778	4906
2	6	SIP	0	1	0	0	0	0	0	0	0
3	22	SCRATCH1	0	1	0	1	8946	2	1	192	214
4	9	CASECC	0	1	0	1	1	279	0	0	0
5	4	BGPDP	0	2	0	1491	0	0	0	0	0
6	5	SCRATCH2	0	39	0	14	8754	2	2	8394	4794
7	14	QG	0	3	0	0	0	0	0	0	0
8	12	KGGX	0	406	0	8946	8946	6	2	486	75
9	10	MI	0	1	0	14	14	1	2	28	10000
10	19	MPTA	0	1	0	32768	128	0	0	0	0
11	18	SCRATCH6	0	26	0	225	225	6	2	450	10000
12	17	OEIGS	0	1	0	14	225	2	2	450	10000
13	7	GPL	0	2	0	1491	0	0	0	0	0
14	11	OPHIG	0	3	0	0	8512	0	0	0	0
15	13	CSTM	0	1	0	1491	2	0	0	0	0
16	8	EQEXIN	0	2	0	1491	0	0	0	0	0
17	15	BGPDT	0	2	0	1491	0	0	0	0	0
18	16	SIL	0	1	0	1491	8946	0	0	0	0
19	27	EST	0	10	0	560	0	0	6	1024	0
20	29	GPECT	0	33	0	82	1491	6	120	0	0
21	30	SCRATCH6	0	41	0	14	8946	2	2	8778	4906
22	32	MDICT	0	3	0	2	0	0	0	0	0
23	23	LAMA	0	1	0	225	0	0	0	0	0
24	21	SCRATCH3	0	29	0	14	4197	2	2	8394	10000
25	28	SCRATCH8	0	14	0	225	225	4	2	450	5022
26	33	GO	0	440	0	225	3972	2	2	7944	10000
27	34	USET	0	3	0	0	8946	0	2039	0	0
28	35	KFF	0	321	0	4197	4197	6	2	486	330
29	36	KFS	0	15	0	4557	4197	2	2	228	10
30	24	PHIA	0	2	0	14	225	2	2	450	10000
31	25	SCRATCH4	0	2	0	14	192	2	2	384	10000
32	26	SCRATCH5	0	2	0	14	225	2	2	450	10000
33	31	USETD	0	3	0	0	0	0	0	0	0

TOTAL IN-MEMORY BLOCKS = 1453
TOTAL DISK BLOCKS = 0

TOTAL FREE SPACE IN WORDS = 48508896
NUMBER OF BLOCKS IN FREE SPACE CHAIN = 2585

Figure 5. Example of In-Memory Data Base Directory

REFERENCES

1. IBM System/370 Vector Operations, SA22-7125-2, Third Edition, August, 1987.

2. IBM Designing and Writing Fortran Programs for Vector and Parallel Processing, SC23-0337-00, First Edition, November, 1986.

3. The NASTRAN Theoretical Manual, NASA SP-221(06), January, 1981.

4. IBM Engineering and Scientific Subroutine Library Guide and Reference Release 3, SC23-0184-3, Fourth Edition, November, 1988.

5. IBM VS Fortran Version 2 Language and Library Reference Release 3, SC26-4221-3, Fourth Edition, March, 1988.

6. IBM Fortran Version 2 Programming Guide Release 3, SC26-4222-3, Fourth Edition, March, 1988.

7. The NASTRAN Programmer's Manual, NASA SP-223(05), Level 17.5, December, 1978.