NASA CONTRACTOR REPORT

NASA CR-2527

NASA CR-2527

STUDY OF THE MODIFICATIONS NEEDED FOR EFFECTIVE OPERATION OF NASTRAN ON IBM VIRTUAL STORAGE COMPUTERS

C. W. McCormick and K. H. Redner

Prepared by
THE MACNEAL-SCHWENDLER CORPORATION
Los Angeles, Calif. 90041
for Langley Research Center

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION • WASHINGTON, D. C. • APRIL 1975

1. Report No. NASA CR-2527	2. Government Accession No.	3. Recipient's Catalog No.

4. Title and Subtitle STUDY OF THE MODIFICATIONS NEEDED FOR EFFECTIVE OPERATION OF NASTRAN ON IBM VIRTUAL STORAGE COMPUTERS	5. Report Date April 1975 6. Performing Organization Code
7. Author(s) C. W. McCormick and K. H. Redner	8. Performing Organization Report No.
9. Performing Organization Name and Address The MacNeal-Schwendler Corporation 7442 North Figueroa Street Los Angeles, CA 90041	10. Work Unit No. 502-32-01-02 11. Contract or Grant No.
12. Sponsoring Agency Name and Address National Aeronautics & Space Administration Washington, DC 20546	13. Type of Report and Period Covered Contractor Report 14. Sponsoring Agency Code

15. Supplementary Notes

FINAL REPORT

16. Abstract

NASTRAN is executed on an IBM 360/370 computer by more than 60 percent of the official NASTRAN users. Many of the IBM 370 models now have a new capability called Virtual Storage (V5). This study was conducted to determine the necessary modifications to make NASTRAN operational under virtual storage operating systems (V51 and V52). The study also presents suggested changes which will make NASTRAN operate more efficiently under virtual storage operating systems. Estimates of the cost and time involved in design, coding, and implementation of all suggested modifications are included.

17. Key Words (Suggested by Author(s)) NASTRAN Virtual storage NASTRAN efficiency improvements	18. Distribution Statement Unclassified - Unlimited New Subject Category 39

19. Security Classif. (of this report) Unclassified	20. Security Classif. (of this page) Unclassified	21. No. of Pages 59	22. Price* $4.25

*For sale by the National Technical Information Service, Springfield, Virginia 22151

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

LIST OF FIGURES AND TABLES

STUDY OF THE MODIFICATIONS NEEDED FOR EFFECTIVE OPERATION OF NASTRAN ON IBM VIRTUAL STORAGE COMPUTERS

By: C. W. McCormick and K. H. Redner
The MacNeal-Schwendler Corporation

SUMMARY

Currently, the NASA Structural Analysis (NASTRAN) computer program is operational for three different series of computers, one of which is the IBM 360/370 series. NASTRAN is executed on an IBM 360/370 computer by more than 60 percent of the official NASTRAN users. Many of the IBM 370 models now have a new capability called Virtual Storage (VS).

This study was conducted to determine the necessary modifications to make NASTRAN operational under virtual storage operating systems (VS1 and VS2). The study also presents suggested changes which will make NASTRAN operate more efficiently under virtual storage operating systems.

INTRODUCTION

This report presents the final results for the study of the modifications needed for effective operation of NASTRAN on IBM Virtual Storage computers. The principal goals of this study were as follows:

1. Determine the minimum coding changes required for the conversion of NASTRAN to operate on virtual storage computers.
2. Determine the areas of NASTRAN where modifications will improve the efficiency.
3. Make suggestions for implementing changes with a minimum number of computer-dependent subroutines.
4. Estimate the time and cost involved in design, coding, and implementation of all suggested modifications.

5. Prepare a list of user practices for NASTRAN operations on virtual storage computers.

This study was completed without the use of any computer time. The MacNeal-Schwendler Corporation has had some experience with a modified version of NASTRAN under the VS1 and VS2 Release 1 operating systems. However, the only significant modification in this system relative to VS was to change the procedure for the acquisition of address space. Furthermore, some of the details of the VS operating systems are still being modified in new releases. Consequently, as experience is gained with the operation of NASTRAN under virtual storage operating systems, some of the conclusions and recommendations made in this report may need to be modified.

Information regarding the characteristics of the Virtual Storage Operating Systems and hardware performance were obtained from the IBM manuals listed in the Bibliography. The details of the NASTRAN code were obtained from the NASTRAN Programmer's Manual and the NASTRAN source code. The timing for the mathematical operations was taken from previous experience with actual runs on virtual storage machines.

The report begins with a brief description of virtual storage operations on IBM computers with special emphasis on those characteristics which are most important for NASTRAN performance. The section on operating system considerations includes the changes required to make NASTRAN operational on current versions of the virtual storage operating systems and a discussion of the use of overlay structures. Following a review of NASTRAN performance with virtual storage operating systems, specific changes are suggested to improve the efficiency of NASTRAN in a virtual storage environment. The estimated effort to implement all suggested changes is included. The report concludes with some recommended user practices and some general conclusions.

VIRTUAL STORAGE OPERATIONS

It is assumed in this report that the reader is already familiar with the virtual storage concept and it is not the intent in this section to present an introduction to virtual storage. However, it is desirable to define the virtual storage terminology used in this report and to review those characteristics of virtual storage operations which are most important in NASTRAN performance.

Virtual Storage Vocabulary

The definitions of terms are based on the hardware and software developed by IBM for the implementation of the virtual storage concept.

Address Space - the set of memory addresses used by a program. VS1 and VS2 Release 1 allow a total address space of 16 megabytes. VS2 Release 2 allows each user an address space of 16 megabytes.

Demand Paging - the occurence of a page fault indicates that a new page must be loaded into real memory before program execution can continue. In a demand paging system, pages are never loaded until the page fault occurs which references that page.

Density of Reference - refers to the fraction of a referenced page which is actually used by the program.

Locality of Reference - good locality of reference means to concentrate all storage references in as few pages as possible, not necessarily contiguous pages.

Page - a subdivision of the address space, 2K bytes on VS1 and 4K bytes on VS2.

Page Fault - an interrupt which occurs when an address is generated which falls in a page which is not in real memory.

Page Frame - a subdivision of real memory equal in size to one page of data.

Page Replacement - a page fault signals the need to load a page in the real memory. If all the available page frames are already occupied, then the required page must replace one of the pages already loaded. The job of selecting which page to replace is the job of the page replacement algorithm. ØS/VS has chosen a technique which is closely related to the least recently used scheme. This procedure is based on the idea that a good choice of a page to replace will be found among the group of pages which have not been referenced in some time, since if it has not been used recently it probably won't be used again very soon.

Real Memory - the set of memory addresses which are physically available on the CPU.

Thrashing - this term is applied when a system is spending more time paging than in program execution. The worst possible thrashing situation which can occur would be a page fault on every single instruction execution, resulting in a performance degradation of several thousand to one. Thrashing is usually caused by random references to an address space which is larger than the real memory available. Thrashing is controlled by monitoring the paging rate (i.e., page faults per second), and if it exceeds some level to deactivate a task. The task to be deactivated may be chosen on the basis of priority or resource requirements.

VIØ (Virtual Input/Output) - the VIØ processor enables temporary data sets to reside on the paging device rather than the usual secondary storage devices. I/Ø requests issued against a VIØ temporary data set by conventional access methods (such as BSAM used in NASTRAN) are intercepted before they are executed. VIØ intercepts and simulates the channel programs associated with the request. VIØ data sets are processed using paging I/Ø thus eliminating channel program translation and page-fixing by the I/Ø supervisor.

Virtual Memory - the address space which can be addressed by a relocate CPU. Physically, the virtual memory exists on a disk storage device, and although a program may reference virtual memory in a random fashion, the information must be transmitted from disk to real memory one page at a time.

Working Set - the number of pages that a program actually references over some interval of time. The size of the working set represents the amount of real memory used by the program.

Program Performance in Virtual Storage

The most important consideration is to keep the central processing unit (CPU) busy doing useful work. This can be best accomplished by keeping the working set as small as possible and still maintain reasonable paging rates. Although there is an overhead associated with the dynamic address translation, this time is not likely to be significant as long as the paging rates are low enough to be supported by the paging hardware. If each piece of data transferred from virtual memory is used only once, the address translation overhead will be rather high, but at the same time the paging rates will also be too high for the hardware devices.

For programs like NASTRAN the paging rate is the single most important consideration in program performance. If the paging rate is too high, the operating system will either give the CPU to other jobs, or if NASTRAN is the only job in execution, the CPU will run at very low efficiency. The paging rates can usually be reduced by increasing the size of the working set. However, for some operations this will result in an unreasonably large working set. To maintain efficiency on virtual storage systems the working set must be kept small by having good locality of reference. If at the same time the density of reference is kept high, the paging rates will be low.

Paging Devices

In designing algorithms for virtual storage systems, the upper limit for the paging rate can be taken as the maximum that can be supported by the available paging device. This approach is based on the assumption that if several jobs are simultaneously executing, each job will receive its share of CPU time and paging device time. For example, if the allowable paging rate is 100 pages per second and 10 jobs are in execution, it is assumed that on the average each job will receive 10% of the CPU time and will page at a rate of 10 pages per second. If one job is using more than its share of CPU time, say 20% in the above example, then it is assumed that it will be allowed to page at 20 pages per second.

Available paging devices are the movable head disks, such as the 2314 and 3330, or the fixed head disk, such as the 2305 Model 1 or the 2305 Model 2. The fixed head disks are much faster because the transfer rate involves only the rotational delay and the readtime, whereas the movable head disk also includes the access time associated with the movement of the read head. The paging rates for the various devices are summarized in Table 1.

Theoretically, the size of virtual storage is limited only by the addressing range of the computing system. In System/370 a 24-bit address is used which allows 16 megabytes of addressable storage. In reality, the size of virtual storage is limited by the amount of external page storage available in the system. The capacity of the available paging devices is summarized in Table 2.

OPERATING SYSTEM CONSIDERATIONS

VS1 and VS2 Release 1

While VS1 and VS2 Release 1 are quite different operating systems, they bear a rather close resemblance to ØS. VS1 looks to the user very much like ØS/MFT, while VS2 Release 1 is essentially similar to ØS/MVT. Experience to date with NASTRAN running under these operating systems has indicated that only modest changes are required to make the code operational.

Only two changes are required in the NASTRAN code to make it operational under VS1 and VS2 Release 1. First, the method of core acquisition must be modified to make it compatible with the virtual storage concept. Second, all accidental fetches outside of the user region must be removed.

The current method of core acquisition will only allow NASTRAN to use address space between the point at which NASTRAN is loaded and the end of real memory. Under virtual storage operating systems, NASTRAN should be allowed to use all parts of the address space. The change in the method of core acquisition requires only a modest change to two routines - GNFIAT and EJDUM2. Both of these routines must be modified to issue a variable GETMAIN for the maximum possible value of the address space (16 megabytes), instead of the maximum real address space on the machine. The necessary alters to implement this change on Level 15.7.7 are given in Figure 1.

TABLE 1

HARDWARE PAGING RATES

Device Model	Time - Seconds				Paging Rate (pages/sec.)
	Access	Rotational Delay	Read 4K Bytes	Total per Page	
2314	.0600	.0125	.0128	.0853	12
3330	.0300	.0084	.0050	.0434	23
2305-1	--	.0025	.0013	.0038	260
2305-2	--	.0050	.0027	.0077	130

TABLE 2

CAPACITY OF PAGING DEVICES

Device Model	Bytes per Track	Megabytes per Unit	Pages (4K) per Unit
2314	7,294	26.2	6,392
3330	13,030	94.1	22,968
2305-1	14,136	4.7	1,146
2305-2	14,660	10.2	2,483

```
./ CHANGE NAME=EJDUM2
./ DELETE SEQ1=290,SEQ2=360
MAXCORE  DC     XL4'FFF000'                                       00001090


./ CHANGE NAME=GNFIAT
./ DELETE SEQ1=25700,SEQ2=26200
MAXCORE  DC     XL4'FFF000'                                       00052500
```

Figure 1. Alters for NASTRAN Level 15.7.7.

Under VS operating systems all addresses outside of your own address space are fetch protected. Under real operating systems accidental fetches outside of your region will usually not cause errors unless the data is actually used. However, under VS operating systems any accidental fetch outside your own address space will result in a fatal error. The only known accidental fetch outside of the assigned address space for Level 15.5 occurs in TRD. Changes in the code for TRD on Level 15.7.7 have removed this accidental fetch.

VS2 Release 2

The NASTRAN "open core" concept has been implemented on IBM 360/370 by making certain assumptions about the way in which the GETMAIN/FREEMAIN macros operate. Chief among these assumptions are:

1. Load modules are loaded by the operating system by issuing GETMAINs which are satisfied from the *beginning* of the address space. Furthermore, since the GETMAINs cover blocks within the address space (2K bytes for ØS), any space not used within the block by the load module is freed at the *beginning* of the address space required for the module.
2. User requests for working storage (GETMAINs) are satisfied from the *end* of the address space in the order in which issued.
3. An area released by a FREEMAIN can be subsequently reacquired by a GETMAIN.

Figure 2 depicts NASTRAN on IBM 360/370 based on these assumptions. In this figure, areas 1 and 2 are acquired by the operating system to load the modules of the NASTRAN program. At the time LINKNS01 (the NASTRAN preface) is loaded, areas 3, 4 and 5 are all free. The initial operation in the preface is to read the NASTRAN data card. This causes the FØRTRAN

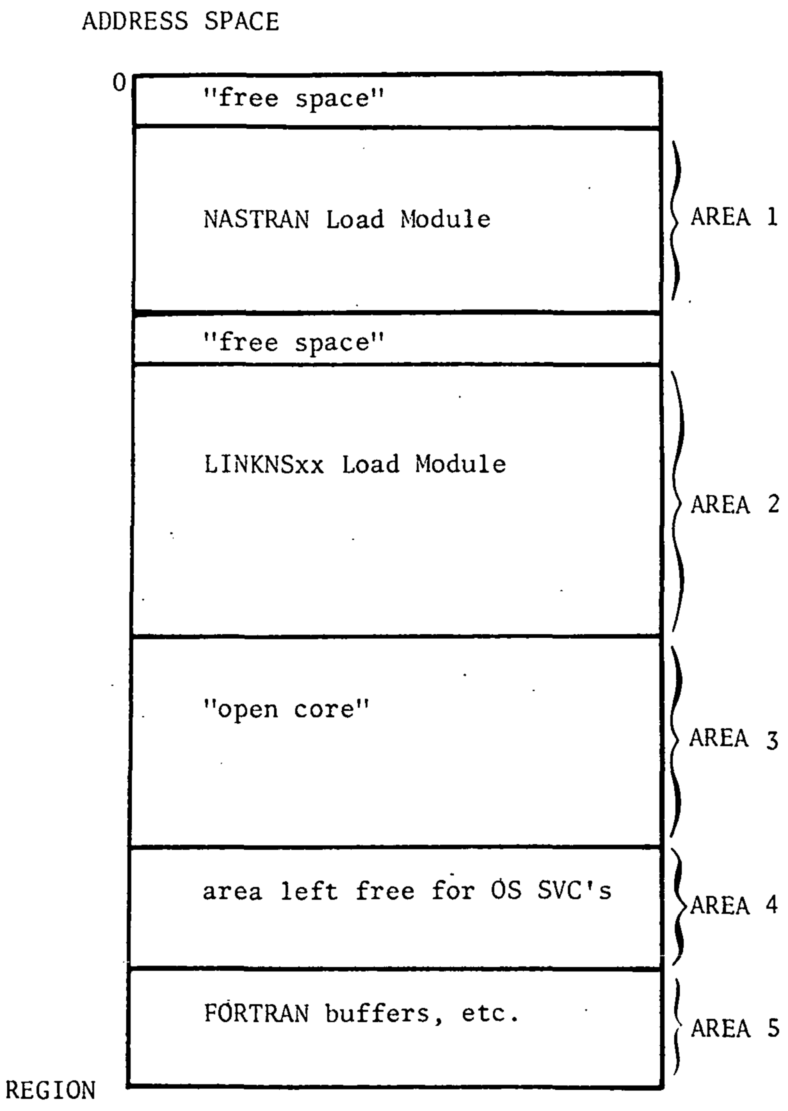

Figure 2. Allocation of Address Space for NASTRAN on IBM 360/370 Computers

I/Ø package to issue a GETMAIN for a buffer for FT05F001. According to assumption (2), this GETMAIN will be satisfied from area 5. Subsequently, a buffer for FT01F001 will be required. Then, when GNFIAT is called, it issues a GETMAIN request for all available address space within the region. This will be satisfied by areas 3 and 4. Based on the KØN360 parameter, area 4 is released by GNFIAT via the FREEMAIN macro.

The additional main requirement for NASTRAN is that area 3 be contiguous with area 2. This is required in order to address continuously from *within* area 2 through the end of area 3. This is, in fact, the basis of the "open core" implementation on IBM 360/370. Continuity is assured if assumptions (1) and (2) are satisfied. And this has been the case on all IBM operating systems through VS2 Release 1.

These assumptions break down, however, in VS2 Release 2. The principal breakdown occurs under assumption (2). Under VS2 Release 2, *all* GETMAIN requests are satisfied from the beginning of the address space. Referring to the above scenario of operations during the NASTRAN preface, this means that areas 1 and 2 will look approximately as described in Figure 2, however area 5 will appear immediately following area 2. For NASTRAN, of course, this means that open core is fragmented and the program cannot operate. The exact failure is an ABEND issued by GNFIAT when the continuity test fails.

It should be possible to modify NASTRAN in such a way that it executes properly under all current IBM operating systems. A potential problem remains, however, and because of this a "generalized" solution may be required. This is included in the recommended modifications. The problem is that IBM will not commit an answer to the question regarding satisfying assumption (3). Sufficient experience has not been gained to date to *know* the answer by experience. Furthermore, even if this assumption is satisfied in Release 2 of VS2, there is no guarantee that the assumption will continue to be satisfied in future releases of VS2.

The recommended changes to NASTRAN to provide compatibility of address space acquisition across IBM operating systems are as follows:

1. Modify subroutine NASTRAN to process a PARM field coded on the EXEC control statement. Two parameters are needed. They are ØPENCØRE = $\left\{\begin{array}{l}\text{LINKEDIT}\\ \underline{\text{GETMAIN}}\end{array}\right\}$ and KØN360 = nnnK. The values for the parameters should be stored in IØTABLE.
2. Modify the SUBSYS deck for LINKNS01 so that EJDUM2 is called in that link.
3. Rewrite the EJDUM2 subroutine in its entirety. If ØPENCØRE=GETMAIN is coded, EJDUM2 will issue a GETMAIN as the first operation in any link. This should insure the acquisition of an address space contiguous with the link. By looking in the inter-link communication vector (DATAB), EJDUM2 will know in which link it is operating. Its operations in LINKNS01 will differ somewhat from other links. If ØPENCØRE=LINKEDIT was coded, EJDUM2 will set the last address of open core (LEND) to the last address in the link. The LINKEDIT option provides a method of operating open core entirely within the link. This may be required if assumption (3) does not hold up. Such an option requires a separately linkedited version of NASTRAN.
4. Modify subroutine GNFIAT to account for the fact that the initial address space acquisition has already occurred in EJDUM2.

Overlays in Virtual Storage

Since the virtual storage operating system loads code only on demand, it is possible to remove the overlay structure and still keep the minimum amount of code resident in real memory. When a program without an overlay structure is loaded for execution, the program is read first from its library into real memory by the loader. The loader finds free page frames, one at a time, until the entire program has been located in virtual storage. This operation will not cause any paging activity unless the program is so large that it takes all of the available page frames. In this case, the first pages will be written to the paging device to make room for later pages. When execution finally begins, the program is dispatched by a branch to its entry point. If this page has been paged out, it must then be paged back in and execution begins. Since the NASTRAN object library consists of about seven million bytes, each loading of the program without overlay would require a large number of unnecessary paging operations.

When a program is loaded for execution with an overlay structure, only the root segment needs to be loaded before execution can begin. With a good overlay structure, there will be little or no paging of the code during execution. In the case of NASTRAN, there will usually be a modest amount of paging activity because a single overlay often contains code which is not used for every execution. If the overlay structure were removed from NASTRAN, it would not only cause a very high paging rate at the beginning of every execution, but would also tend to fill the primary paging device. For example, each module of a 2305 Model 1 fixed head disk will only hold about five million bytes of information. In addition, almost half of the available address space would be used for code.

During execution the program without an overlay structure must rely on paging operations to provide the code as needed. Therefore, when control passes from phase to phase during execution, the old working set is gradually replaced by the pages making up the new working set. Those pages which have been changed will have to be written out if real memory is highly utilized. Some of the page out operations will probably be just wasted motion, because the change in the page was not really essential (for example, an intermediate work area).

The operation of a program using overlays is quite different. When a new segment is requested (either implicitly or explicitly) all of the page frames occupied by the current segment are released immediately with no paging out of changed pages. The new segment is then loaded directly from the program library as if it were a new program. The load modules can be stored on the relatively slow movable head disks as is the current practice on real memory machines.

It is recommended that NASTRAN use an overlay structure on virtual storage machines for the following reasons:

1. There is less contention for page frames because pages are released in blocks when a segment ends.
2. Loading the working set for each segment is faster because there is no page out activity.
3. The initial program load is faster because only the root segment must be loaded.
4. Almost half of the address space will be saved for NASTRAN operations.
5. Valuable space on the high-speed paging device will be saved for more important operations.

The performance of a program in a VS environment is not very sensitive to the details of the overlay structure. The main reason for having an overlay structure with large programs is to avoid overloading the virtual storage and the paging devices. The use of any reasonable overlay structure will usually result in only a modest number of paging operations. Consequently, it is recommended that NASTRAN use the same overlay structure on virtual storage machines as on real storage machines.

REVIEW OF CURRENT NASTRAN CODE

Almost all matrix operations in NASTRAN and some of the data processing subroutines are provided with spill logic to be used in those cases where the arrays being processed are too large for main memory. This spill logic can be thought of as a kind of software paging where the page size is 100K bytes or more rather than the usual 4K bytes in virtual storage operations. In all cases where spill logic is provided, and in most other parts of the NASTRAN code, the working space is used in a random way. This random use of the working space will usually cause excessive paging if the code attempts to use more working space than is available in real memory.

The relatively large address space available with virtual storage will allow NASTRAN code to operate on matrices and tables approaching sizes of a million words without the use of spill logic. One of the important tasks in the review of the current code is to calculate the paging rates when the spill logic is replaced by the use of a large virtual address space. A second major task in the review of the current code is to review the working space requirements for all modules without spill logic with regard to their performance in a virtual storage environment. The working space was taken as a point of departure in the review of the NASTRAN code because all significant data operations in NASTRAN take place in the working space area.

Summary of Working Space Requirements

A summary of the working space requirements was prepared for use as an aid in the review of the code for the functional modules and the matrix operations. The notation in the tables is the same as that used in the NASTRAN Programmer's Manual.

Table 3 tabulates the working space requirements and the basis for the spill logic, if any, for each of the basic matrix subroutines. Further details may be found in Section 3.5 of the Programmer's Manual.

Table 4 tabulates the working space requirements for each of the functional modules. Each functional module is identified as belonging to one of four groups depending on the nature of the working space requirement. The four groups are defined as follows:

Group 0 - modules which have no working space requirement in the "open core" region. Small working arrays may be provided within the module code.

Group 1 - modules which require space for vectors or tables which do not exceed eight times the number of grid points in the finite element model. None of these modules are provided with spill logic.

Group 2 - modules which require space for tables or matrices of variable sizes. Spill logic is provided in those cases where the working space requirements may be large.

Group 3 - modules for which the working space requirements are established by one or more matrix routines. In these cases the basis for any spill logic is given in Table 3.

TABLE 3

WORKING SPACE FOR MATRIX SUBROUTINES

NAME	WORKING SPACE	SPILL LOGIC
ADD	None	No
CDCØMP	2*DECØMP	2*DECØMP
DECØMP	$\bar{B}(B+\bar{B}) + \bar{B}C + (B+\bar{B})\bar{C} + C\bar{C}$	$\bar{B}R+\bar{B}C+C\bar{C}+\bar{C}(B+\bar{B})$
DMPY	Diagonal Matrix	No
FBS	Right-Hand Sides	Columns of R.H.S.
GFBS	Right-Hand Sides	Columns of R.H.S.
MERGE	1 Column	No
MPYAD-1	Right-Hand Matrix plus Result Matrix	Columns of Matrices
MPYAD-2	Packed Left-Hand Matrix	Columns of Matrix
MPYAD-3	Left-Hand Matrix	Columns of Matrix
PARTN	1 Column	No
SADD	1 Column	No
SDCØMP	$\frac{1}{2}(\text{Active Columns})^2$	Rows of Triangle
TRNSP	Full Matrix	Submatrices

In some cases operations within the same functional module have different working space requirements, and hence a single module may be identified with more than one group. Only the critical working space requirements are listed in Table 4 for each of the functional modules. Further information may be obtained in Section 4 of the Programmer's Manual.

Input/Output Operations

With the exception of card image processing, printed output and user I/Ø in the INPUTT and ØUTPUT modules, all input/output operations in NASTRAN are accomplished through IØ360 which utilizes BSAM macros. Since the operations occur at the block level (GINØ accomplishes the blocking/deblocking), they can be optimized for the various types of secondary storage units available in any given computer configuration. This optimization is still valid under VS operating systems.

VS2 offers other possibilities, for example VIØ. The user can specify VIØ through JCL and, as a result, its use is transparent to the operating program. Because of the transparency, some experiments could easily be conducted (VIØ can be used only for temporary data sets such as PRIxx, SECxx and TERxx in NASTRAN). However, the VS2 literature indicates that performance improvements can be expected through VIØ only when more than 7-9 blocks per track are used. Since the NASTRAN standard is usually 1-3 blocks per track, no performance gain is anticipated in this area.

TABLE 4

WORKING SPACE FOR FUNCTIONAL MODULES

MODULE	GROUP	WORKING SPACE	SPILL LOGIC
ADD	3	See ADD in Table 3	No
ADD5	3	See SADD	No
BMG	1	BGPDT + 26*Boundary Points	No
CASE	2	CASECC	No
CEAD	3	DET - See CDCØMP	Yes
	3	HESS - See CDCØMP, GFBS	Yes
	2	HESS - $6N^2 + 8N$	No
	3	INV - See CDCØMP	Yes
	2	INV - 14 Vectors	No
CYCT1	3	See MPYAD	Yes
CYCT2	3	See MPYAD	Yes
DDRMM	2	Largest ØFP Output Record	No
	3	See MPYAD	Yes
DDR1	3	See MPYAD	Yes
DDR2	3	See MPYAD, FBS	Yes
DECØMP	3	See SDCØMP, DECØMP, CDCØMP	Yes
DPD	2	EQDYN + 1 Data Table	No
DSCHK	3	See MPYAD	Yes
DSMG1	2	Same as SMA1	Yes
EMA	2	Complete Packed Matrix	1 Pivot Point
EMG	2	CSTM, MPT, DIT, CØNGRUENT	No
FBS	3	See FBS, GFBS in Table 3	Yes
FRRD	2	Vector*Frequencies	1 Frequency
	3	See MPYAD,SDCØMP,CDCØMP,DECØMP,FBS,GFBS	Yes

TABLE 4 (Continued)

WORKING SPACE FOR FUNCTIONAL MODULES

MODULE	GROUP	WORKING SPACE	SPILL LOGIC
GKAD	3	See MPYAD	Yes
GKAM	3	See MPYAD	Yes
GPCYC	1	EQEXIN+CYJOIN+USET+Vector	No
GPFDR	1	EQEXIN + 1 Vector	No
GPSP	1	USET+SIL+GPL	No
GPWG	3	See MPYAD	Yes
GP1	1	2(Grid+Scalar)	No
GP2	1	2(Grid+Scalar)	No
GP3	1	2(Grid+Scalar)	No
GP4	1	6*Grid+Scalar	No
IFP1	2	CASECC Record	No
INPUT	0	None	No
INPUTT1	0	None	No
INPUTT2	2	Longest Record	No
MATGPR	1	GPL+USET+SIL	No
MATPRN	1	1 Column of Matrix	No
MATPRT	1	1 Column of Matrix	No
MCE1	1	1 Vector	No
MCE2	3	See MPYAD in Table 3	Yes
MERGE	1	1 Column of Matrix	No
MPYAD	3	See MPYAD	Yes
MTRXIN	2	EQEXIN + 1 DMIG	Partitions of DMIG

TABLE 4 (Continued)

WORKING SPACE FOR FUNCTIONAL MODULES

MODULE	GROUP	WORKING SPACE	SPILL LOGIC
ØFP	0	None	No
ØPTPR1	2	6 Data Tables	No
ØPTPR2	2	5 Data Tables	No
ØUTPUT1	0	None	No
ØUTPUT2	2	Longest Record	No
ØUTPUT3	0	None	None
PARAM	0	None	No
PARAML	0	None	No
PARAMR	0	None	No
PARTN	1	1 Column of Matrix	No
PLA1	2	Same as SMA1	Yes
PLA2	1	1 Solution Vector	No
PLA3	1	1 Solution Vector	No
PLA4	2	Same as SMA1	Yes
PLØT	1	Grid + 8*Points in Plotted Set	No
PLTSET	2	Element ID + Grid ID for Plotted Set	No
PLTTRAN	0	None	No
PRTMSG	0	None	No
PRTPARM	0	None	No
RANDØM	2	2*NFREQ+5*RANDPS+NTAU+PRETAB +5*POINTS+POINTS*NFREQ	1 POINT*NFREQ
RBMG1	3	See PARTN	No
RBMG2	3	See SDCØMP	Yes
RBMG3	3	See MPYAD, FBS	Yes
RBMG4	3	See MPYAD	Yes

TABLE 4 (Continued)

WORKING SPACE FOR FUNCTIONAL MODULES

MODULE	GROUP	WORKING SPACE	SPILL LOGIC
READ	3	DET - See DECØMP, SDCØMP	Yes
	2	GIV - Triangle of Matrix	Rows
	3	INV - See SDCØMP	Yes
	2	7 Vectors	No
RMG	2	Triangle of RADMTX	No
	3	See MPYAD, DECØMP, GFBS	Yes
SCE1	3	See PARTN	No
SDRHT	2	14*Elements+Solution Vector	No
SDR1	3	See MPYAD	Yes
SDR2	2	EQEXIN + 1 Vector + CASECC + CSTM	No
	2	Stress Matrices + 1 Vector	Subset of Stress Matrices
SDR3	2	SØRT1 Output Group	Subset of Output Group
SETVAL	0	None	No
SMA1	2	CSTM+MPT+GPCT+submatrices for 1 pivot	1 Column
SMA22	2	Same as SMA1	1 Column
SMA3	2	Matrix for General Element	Yes
	3	See SDCØMP, MPYAD, FBS	Yes
SMP1	3	See SDCØMP, FBS, MPYAD	Yes
SMP2	3	See MPYAD	Yes
SMPYAD	3	See MPYAD	Yes
SØLVE	3	See DECØMP, SDCØMP, FBS, GFBS	Yes
SSG1	2	1 Load Vector	No
SSG2	3	See MPYAD	Yes
SSG3	3	See FBS, MPYAD	Yes
SSG4	3	See MPYAD, FBS	Yes

TABLE 4 (Concluded)

WORKING SPACE FOR FUNCTIONAL MODULES

MODULE	GROUP	WORKING SPACE	SPILL LOGIC
SSGHT	2	3 Vectors + 2 Tables	No
TABPCH	2	2*Longest Record	No
TABPRT	2	1 Record of Table	No
TABPT	0	None	No
TA1	2	Grid*Connections+ECT	Grid*Conn.
TRD	2	Vector*Time Steps	1 Time Step
	3	See MPYAD, DECØMP, SDCØMP	Yes
	2	7 Vectors + Nonlinear Tables	No
TRHT	3	See SDCØMP or DECØMP	Yes
	2	7 Vectors + Nonlinear Data	No
TRLG	3	See MPYAD	Yes
	2	2*USET + 3*Load Points + 2*SIL	No
TRNSP	3	See TRNSP in Table 3	Yes
UMERGE	3	See MERGE	No
UPARTN	3	See PARTN	No
VDR	1	EQDYN + Solution Vector	No
VEC	1	USET	No
XYPLØT	2	(X,Y)Pairs to be Plotted	Single Pair
XYTRAN	2	Curve Data for 1 Frame	Delete Curves

Matrix Operations

When considering the efficiency of NASTRAN, particularly in the solution of large problems, the matrix operations are the most important. Usually the large memory requirements and the long running times are associated with the matrix operations. Matrices in NASTRAN are stored by columns in a packed string format. It does not make any difference whether the matrices are stored by rows or by columns. The important consideration is that once a choice has been made, the matrix operations must be designed to operate efficiently for the type of storage selected.

In reviewing the code for the matrix operations, it will be assumed that the address space is larger than the real memory space. In other words, the hardware paging will be used to the maximum extent possible. The NASTRAN spill logic will not function unless the working space requirement exceeds the available virtual address space.

It will also be assumed that the matrices are large compared to the size of a single page. A page containing 4K bytes will hold about 500 terms for a dense matrix and as few as 200 terms for a very sparse matrix. The performance of the matrix routines for smaller matrices will be less critical because the density of reference will tend to be higher. For example, if only a single term is used from each column for a given operation, the density of reference will be higher if several columns are contained within a single page.

Partition and Merge

The working space requirement for PARTN and MERGE is relatively small as it consists only of a single column plus a packed array of one word for each 31 rows for use as a mapping function. No working space is required for the matrices to be partitioned or merged, as the NASTRAN I/Ø routines provide the nonzero terms sequentially one at a time. No spill logic is provided for this matrix operation.

Matrix Addition

The ADD routine does not require any working space, as the NASTRAN I/Ø routines provide the nonzero terms sequentially one at a time. The result of each unit addition is immediately transferred to the output buffer.

The SADD routine requires working space for a single column of the result matrix. The input matrices are read by the NASTRAN I/Ø routines sequentially a single term at a time. No spill logic is provided for this routine.

Matrix Multiply/Add

The general multiply/add operation may be written as follows:

$$A_{mn}B_{np} + C_{mp} = D_{mp}$$

In Method 1 for MPYAD working space is required for the [B] and [D] matrices as shown in Figure 3. If the address space required for the shaded areas of the [B] and [D] matrices is larger than the real memory available, the shaded areas will be paged once for each nonzero term in the [A] matrix.

The paging rate for the [B] matrix is calculated as follows:

$$\text{Total Number of Pages} = p(mn\rho_A)$$

$$\text{Total CPU Time} = mnp\rho_A M$$

where ρ_A = density of [A] and M = time for single multiply/add operation.

The paging rate required to keep the CPU busy is the ratio of the total number of pages to the total CPU time. Therefore, the paging rate is given

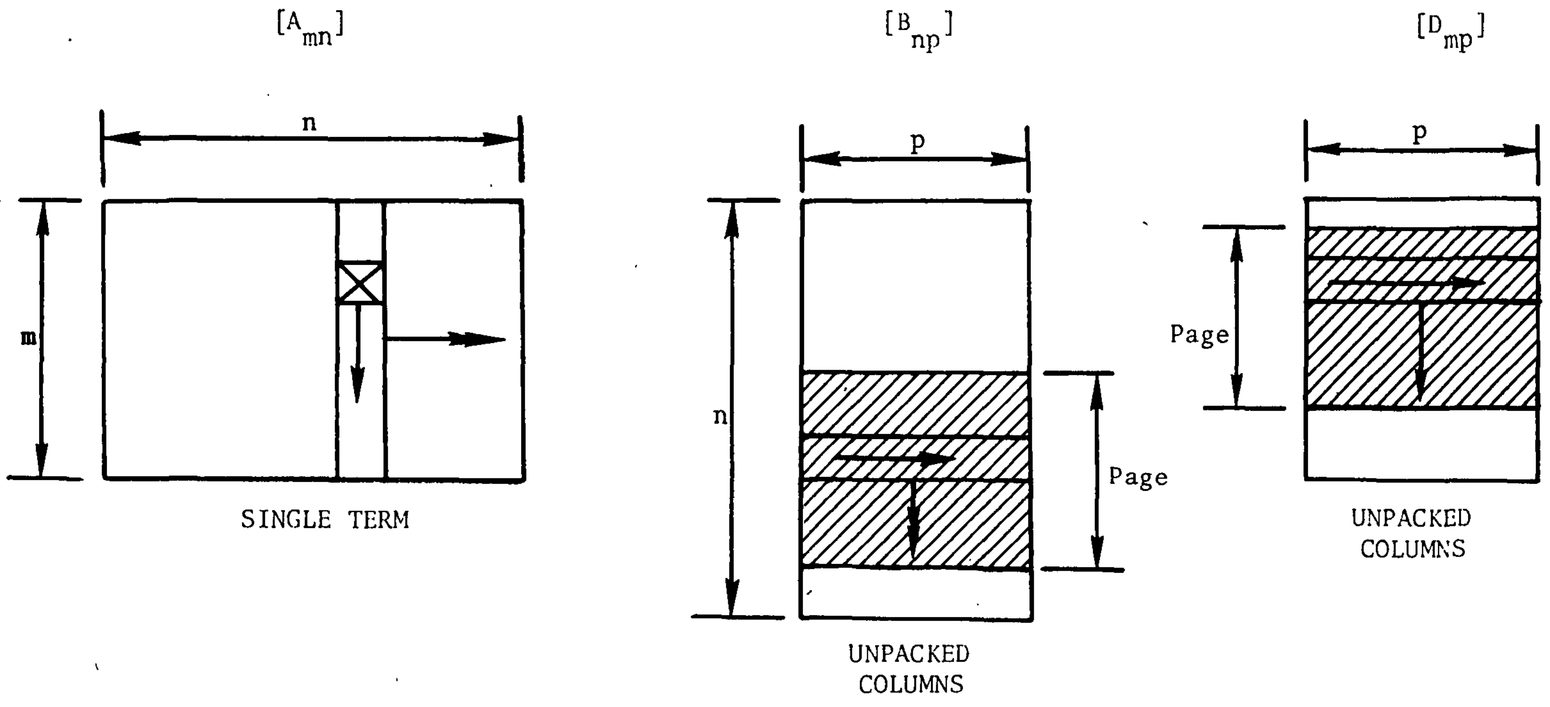

Figure 3. Method 1 MPYAD

as follows:

$$\text{Paging Rate} = \frac{1}{M}$$

For M = 10 microseconds, the paging rate for the [B] matrix will be 100,000 pages per second. A similar paging rate will apply for the [D] matrix. A similar calculation will give the same paging rate for the transpose option of Method 1.

In Method 2 for the multiply/add operation, the main working space requirement is for packed columns of the [A] matrix as shown in Figure 4. If the address space required for the [A] matrix is larger than the real memory available, the [A] matrix will be paged a number of times equal to the number of columns in the [B] matrix.

The resulting paging rate for Method 2 is calculated as follows:

$$\text{Total Number of Pages} = \frac{p(mn\rho_A)}{P}$$

$$\text{Total CPU Time} = mnp\rho_A\rho_B M$$

where P is the number of nonzero terms per page.

The paging rate required to keep the CPU busy is the ratio of the total number of pages to the total CPU time. This paging rate is given as follows:

$$\text{Paging Rate} = \frac{1}{PM\rho_B}$$

The minimum paging rate will result when the [B] matrix is full. However, this method is rarely used when the [B] matrix is full, as Method 1 is usually more efficient. The paging rate for $\rho_B = 1$, P = 500 terms per page, and M = 10 microseconds is 200 pages per second. If the [A] matrix is sparse, which is the usual case for Method 2, the paging rate will be greater because of the reduced number of matrix terms per page.

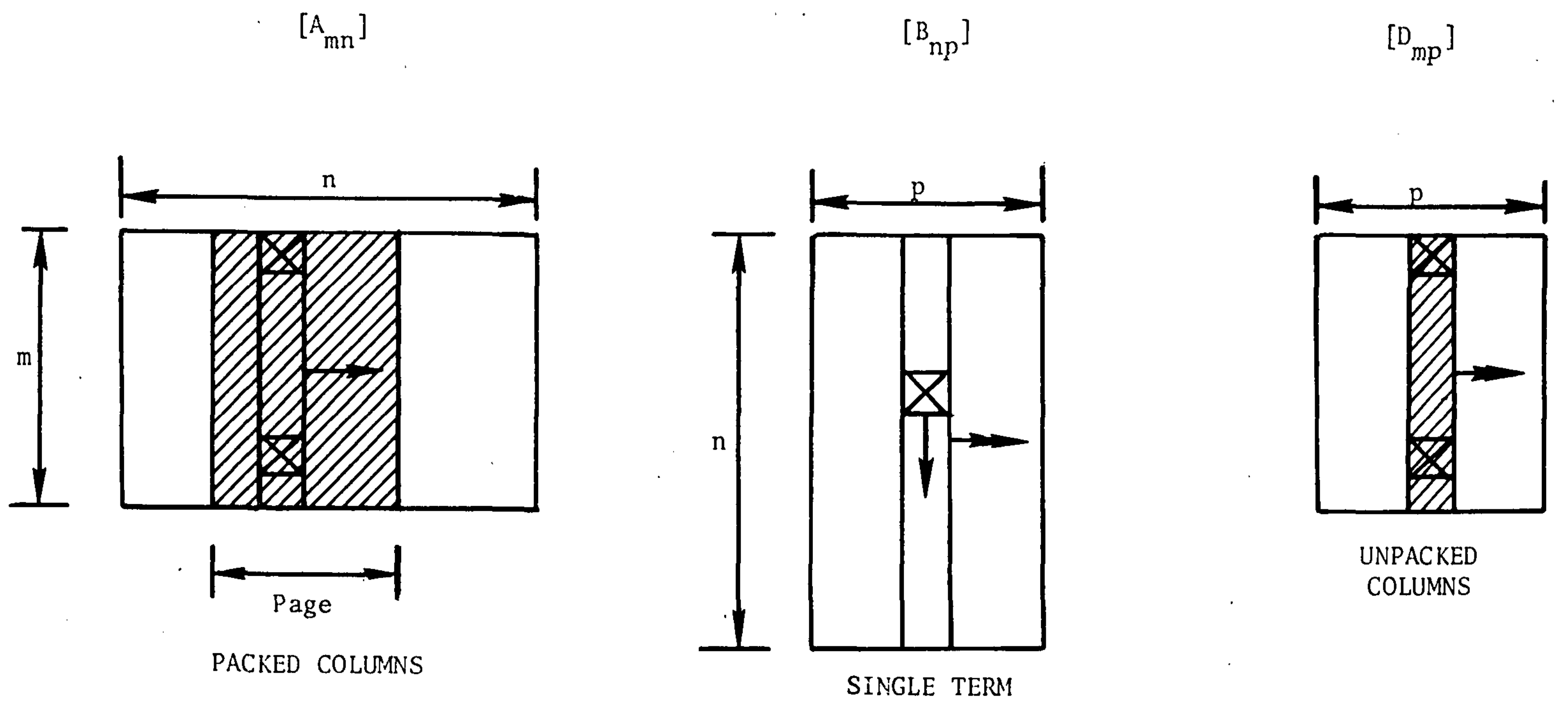

Figure 4. Method 2 MPYAD

The paging rate for the transpose option of Method 2 will be the same as the non-transpose option with ρ_B taken as unity, as no advantage is taken of the density of the [B] matrix in the transpose option.

In Method 3 for the multiply/add operation, the main working space requirement is for unpacked columns of the [A] matrix (rows of $[A]^T$) as shown in Figure 5. If the address space required for the [A] matrix is larger than the real memory available, the [A] matrix will be paged once for each nonzero term in the [B] matrix.

The paging rate is calculated as follows:

$$\text{Total Number of Pages} = \frac{m(np\rho_B)}{P}$$

$$\text{Total CPU Time} = mnp\rho_B M$$

The paging rate is given as follows:

$$\text{Paging Rate} = \frac{1}{PM}$$

For M = 10 microseconds and P = 500 terms per page, the paging rate for the [A] matrix will be 200 pages per second.

The multiply option with a diagonal matrix (DMPY) requires working space for only a single column, and no spill logic is provided.

Matrix Decomposition

For symmetric decomposition (SDCØMP) the working space required is $\frac{1}{2}C^2$, where C is the number of active columns. If the working space required for decomposition operations is larger than the real memory available, the triangular array with the side dimension of C will be paged a number of times equal to the number of rows in the matrix. The paging rate is calculated as follows:

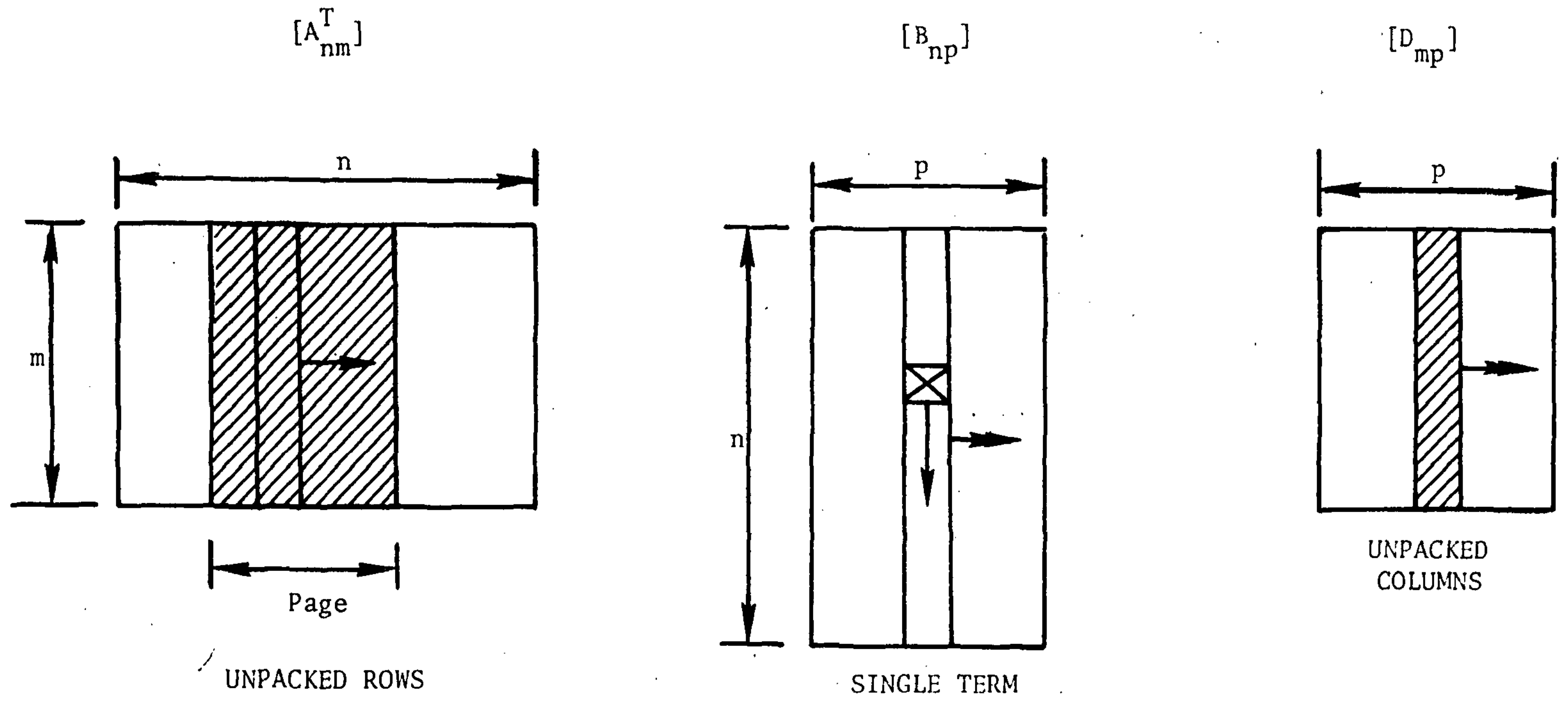

Figure 5. Method 3 MPYAD

$$\text{Total Number of Pages} = \frac{1}{2}\,\frac{C^2N}{P}$$

$$\text{Total CPU Time} = \tfrac{1}{2}C^2NM$$

where N is equal to the number of rows in the matrix.

The paging rate required to keep the CPU busy is given as follows:

$$\text{Paging Rate} = \frac{1}{PM}$$

For the case of 500 nonzero terms per page and a unit multiply time of 10 microseconds, the paging rate will be 200 pages per second.

The paging rate for unsymmetric decomposition (DECØMP) will be the same because the number of terms is four times as great and the CPU time is also four times as great. The paging rates for complex decomposition (CDCØMP) will only be about one-half as much because the number of terms per page is one-half as great and the unit multiply time is almost four times as great.

Forward/Backward Substitution

In the Forward/Backward Substitution operation working space is required for the right-hand sides. The active pages at a particular time during the forward pass are shown in Figure 6. As indicated in Figure 6 the terms from a single row are operated on with strings from the triangular factor and the results are stored in the associated strings for the right-hand sides. If the address space for the shaded areas is greater than the real memory, $\frac{C}{S}$ L pages will be required for each shaded area and for each row of the matrix. The paging rate is calculated as follows:

$$\text{Total Number of Pages} = 2\left(\frac{C}{S}\,L\right)N$$

$$\text{Total CPU Time} = CNML$$

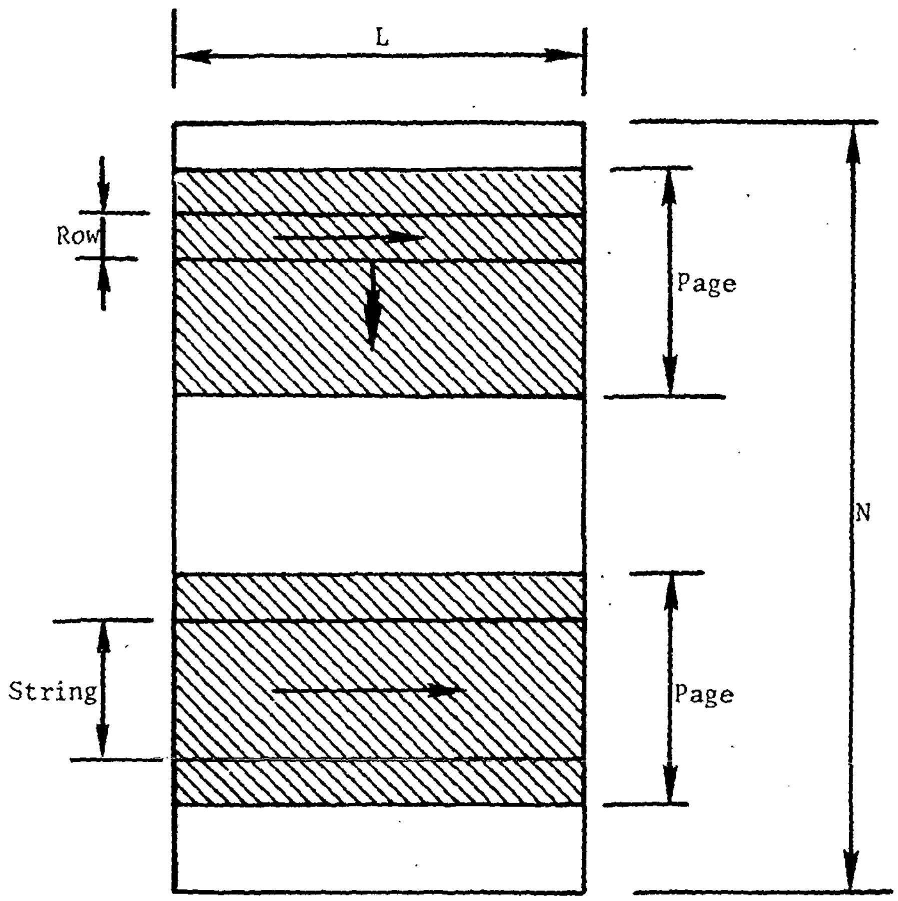

Figure 6. Forward/Backward Substitution

where N = the number of rows on the right-hand side,

L = the number of columns on the right-hand side,

C = the number of active columns in the triangular factor, and

S = the number of consecutive nonzero terms in the string.

The corresponding paging rate is as follows:

$$\text{Paging Rate} = \frac{2}{SM}$$

If the triangular factor is very sparse, that is S = 1, and if M = 10 microseconds, the resulting paging rate is 200,000 pages per second. If single page buffers are used, the maximum value for S is about 500, which would result in a paging rate of 400 pages per second.

Matrix Transpose

For the transpose operation working space is required for the whole matrix. If the required working space is greater than the available real memory, the paging rate is calculated as follows:

$$\text{Total Number of Pages} = NL$$

$$\text{Total CPU Time} = NLI$$

where N = the number of rows,

L = the number of columns, and

I = input/output time.

The input/output time is the CPU time required for the unpacking and packing of the nonzero terms in the matrix. The paging rate is as follows:

$$\text{Paging Rate} = \frac{1}{I}$$

For dense matrices, the input/output time is about the same as the unit multiply time. If I is taken as 10 microseconds, the paging rate is 100,000 pages per second.

Sparse matrices should be transposed by using the transpose option of multiply/add to operate on an identity matrix. The paging rate for transposing sparse matrices will be the same as that for multiply/add Method 2.

Summary of Paging Rates for Matrix Operations

The paging rates for the matrix operations are summarized in Table 5 for computer models 145, 158 and 168. The times for the unit multiply operation and the unit input/output operation were both taken as 80 microseconds for the model 145, 10 microseconds for the model 158, and 3 microseconds for the model 168.

A review of Table 5 indicates that it is not possible to support the paging rates if the current NASTRAN code is used with a large address space. This is equivalent to using hardware paging rather than software paging. Some of the operations could be supported on the model 145 with a fixed head disk. However, this combination of hardware is extremely unlikely. In all other cases, the paging rates are either at the limit of the available devices, or in many cases, far exceed the limits of the available devices. It is concluded that hardware paging cannot be used with the current NASTRAN code.

Executive Operations

Many of the executive modules have essentially no working space requirements at all. Consequently, these modules will operate in very modest amounts of real memory and paging rates will be extremely low.

Several of the executive modules require working space for tables. Since these tables are used in a random way, there must be enough real storage to support the table size or paging rates of several thousand

TABLE 5

PAGING RATES FOR MATRIX OPERATIONS

Matrix Operation	Paging Rate - Pages/Second		
	Model 145	Model 158	Model 168
MPYAD - 1	25,000	200,000	700,000
MPYAD - 2	25	200	700
MPYAD - 3	25	200	700
SDCØMP	25	200	700
CDCØMP	12	100	350
FBS - Dense	50	400	1,300
Sparse	25,000	200,000	700,000
TRANS - Dense	12,500	100,000	300,000
Sparse	25	200	700

pages per second will result. However, the real storage requirement for these modules is not likely to be greater than 200K bytes and frequently much less. Also, the execution time for these modules is relatively short.

The only executive routine for which a large amount of working space is required is the subroutine for sorting the bulk data deck (XSØRT). The XSØRT routine uses the working space in a random way, consequently high paging rates can be expected unless enough working space is provided to hold the complete bulk data deck. Since only a few items of data are used from each page, the paging rates can be expected to be at least several thousand pages per second.

Functional Modules

The working space requirements for the functional modules are given in Table 4. The discussion of the performance of the existing code for the functional modules is organized according to the groups specified in the second column of Table 4.

All of the functional modules in Group 0 have essentially no working space requirements. All operations either take place in small local arrays within the code or in a nominal working space of a few words. This group of modules offers no special problems with regard to efficiency under VS operating systems.

The functional modules in Group 1 all use the working space in a random way. Consequently, the paging rates for these modules will be several thousand pages per second unless the available real memory is at least equal to the required working space. On the other hand, the working space for the modules in this group is a well-defined function of problem size, and the running times for these modules are relatively short. Therefore, the modules in Group 1 should not cause any special problems with regard to efficiency under VS operating systems unless the required working space is a large fraction of the available real memory.

The PLØT module is a little different than the other modules in Group 1. Although the working space for the PLØT module is similar to that for the other modules in Group 1, the amount of code in a single overlay is much greater. On real machines, the minimum core requirement is often controlled by the PLOT module, especially for small problems. However, since much of the PLOT code is dormant at any particular time in execution, the VS operating system will page out much of the code, and the working space requirement should not be any greater than other modules in Group 1.

Some of the functional modules in Group 2 are not provided with spill logic. In real memory systems these modules will issue a fatal error message if there is not sufficient main memory to support the working space requirements. In VS systems these modules will have high paging rates if there is not sufficient main memory to support the working space requirements. Although these modules without spill tend to have small working space requirements, this is not necessarily always true. Consequently, it is possible that many of the modules in this group will experience high paging rates with VS operating systems, particularly for large problems.

The inverse power option for READ has a working space requirement for 7 vectors and CEAD has a similar requirement for 14 vectors. In both cases a fatal error message will be issued by real memory operating systems if the required working space is not available. However, with VS operating systems the working space requirement is less, because not all vectors are needed at the same time. If working space is provided for only two vectors, the average paging rate over the period of an iteration is less than one page per second. However, since some of the vectors are used several times per iteration, local paging rates of several hundred pages per second would occur. This could be relieved by providing working space for four vectors, as three of the vectors are used only once for each iteration. In any event, the high paging rates would be of very short duration as not more than 10 or 20 pages are likely to be required for each vector.

The modules in Group 2 with spill logic can be expected to have working space requirements for large problems which will exceed the available real memory. When the working space requirement for any one of these modules exceeds the available real memory, the paging rates will far exceed the capacity of the fastest paging devices. The high paging rates will result in poor performance with VS operating systems.

The working space requirements for the functional modules in group 3 are controlled by one or more matrix operations. These modules can be recognized in Table 4 by the references to the various matrix operations given in the column headed Working Space. The operation of these modules will be dominated by the matrix operations, hence their performance will be the same as that for the individual matrix operations.

SUGGESTED CHANGES TO NASTRAN CODE

In considering changes to the NASTRAN code to improve performance under VS operating systems, it is important to recognize that actual field experience with VS is rather limited and that the operating systems are still undergoing significant modifications. Under these conditions, it seems desirable to make all changes in such a way as to maintain maximum possible flexibility in the program operations with virtual storage operating systems.

Another important consideration is to make all changes in a machine-independent fashion without degrading the performance on any of the real memory machines. With the exception of the LINKEDIT option associated with Release 2 of the VS2 operating system, all suggested changes are machine independent. The GETMAIN option for VS2 Release 2 described in the section on operating system considerations is machine independent. The LINKEDIT option is only provided as a back-up procedure because the method of acquiring address space is not yet well defined for Release 2 of VS2.

The current NASTRAN code will operate satisfactorily if the address space is limited to that which can be reasonably supported by the available real memory. Although this procedure will not take any special advantage of virtual storage, it will restrict the paging rates to reasonable values. The procedure for doing this is given in the section on recommended user practices.

The single most important change needed in the code is to provide both the size of the address space and a reasonable value for the available real memory. The address space is specified on the JOB card, and stored by the operating system in the same way as for real memory systems. A reasonable value for the real memory could be specified as a parameter on the EXEC card, and the difference between the address space and the real memory could be stored in the SYSTEM common block, which for the purposes of discussion will be called SYSTEM(V). Then, if SYSTEM(V)=0, it either means you are on a real memory machine or for some reason do not choose to use the virtual memory space. With this provision any routine can access SYSTEM(V) and take whatever action is appropriate for that particular routine.

The allocation of a fixed amount of real memory on a VS machine is inconsistent with the whole concept of virtual storage. On the other hand, if there is not enough real memory to support the programs in execution at any one time, the performance will be very poor. If the real memory requirement is too large a fraction of the total available real memory, the required space will be difficult to acquire and the wall clock times will be very long. In general, the operating system will not permit a program to execute unless there is sufficient real memory available to support the working space requirement. If the real memory is insufficient to support the program operations, the paging rates will tend to exceed allowable values, and the operating system will deactivate the job. The deactivation procedure will make the wall clock times long especially when there are other jobs in execution at the same time with smaller real memory requirements.

The major objective of the suggested code modifications is to reduce the amount of working space to the smallest possible value without unreasonable increases in the CPU time. At the same time, the paging rates must be kept within the limits of the available paging devices. Paging rates that are beyond allowable values for extended periods of time will always lead to poor performance.

Input/Output Operations

As indicated in the section on Input/Output Operations under the Review of Current NASTRAN Code, no obvious changes to the input/output operations appear warranted. One possibility for a small improvement would be to have all GINØ buffers be one page in length and aligned on page boundaries. Such a procedure would be relatively easy to implement but it is felt that its payoff in terms of performance improvement would be small.

If it is desired to implement this feature, it could be accomplished as follows:

1. Add to the PARM field the parameter PAGEBUFF to be coded PAGEBUFF=$\left\{\begin{matrix}\text{YES}\\ \text{NØ}\end{matrix}\right\}$ and process this parameter in NASTRAN.

2. Add to the EJDUM2 implementation discussed earlier an additional GETMAIN for, say, 20 pages.

3. Modify GINØ to bypass the buffer addresses which are passed through a CALL ØPEN and substitute instead a page buffer from a pool of buffers established by EJDUM2.

4. Modify GNFIAT to force the SYSBUF parameter to one page.

Matrix Operations

No changes in the code are recommended for PARTN, MERGE, ADD, SADD, or DMPY. None of these routines have large working space requirements. Consequently, there should be no degradation of performance under VS operating systems.

All matrix routines which have large working space requirements are also provided with spill logic. If a limitation on the amount of working space is provided to these modules, the existing spill logic, with slight modifications can be used to improve their performance under VS operating systems. In general, this objective can be accomplished by modifying the code in each of these modules to reference SYSTEM(V) in order to obtain an allowable value for the working space. Details for each of the matrix operations are given in the following sections.

Changes for Multiply/Add

The arithmetic time for Method 1 is a constant regardless of the amount of working space. The product of the time for spill operations and the amount of working space is also a constant. Therefore, the product of total CPU time and time for real memory occupancy will be a minimum if working space is provided for only a single column of the right-hand matrix. However, this will result in excessive paging rates.

If working space is provided for the shaded areas of the right-hand matrix and the result matrix as shown in Figure 7, the paging rates can be reduced in direct proportion to the number of columns, c, per pass. The number of columns per pass can be controlled by limiting the amount of working space. The working space can be limited by making a CALL to CØRSZ and subtracting the contents of SYSTEM(V).

The paging rate is calculated as follows:

$$\text{Total Number of Pages} = (mn\rho_A)\frac{p}{cP}$$

$$\text{Total CPU Time} = mnp\rho_A M$$

$$\text{Paging Rate} = \frac{1}{cPM}$$

For $P \approx 500$ pages, $M = 10$ microseconds, and $c = 20$ columns, the paging rate is 10 pages per second.

Virtual address space can be provided for the left-hand matrix in a machine-independent way by using the option for multiple buffers already available in the NASTRAN code. The address space remaining after subtracting the working space, SYSTEM(V), can be used for multiple buffers. Only a modest change is required in the code for MPYAD to implement this option.

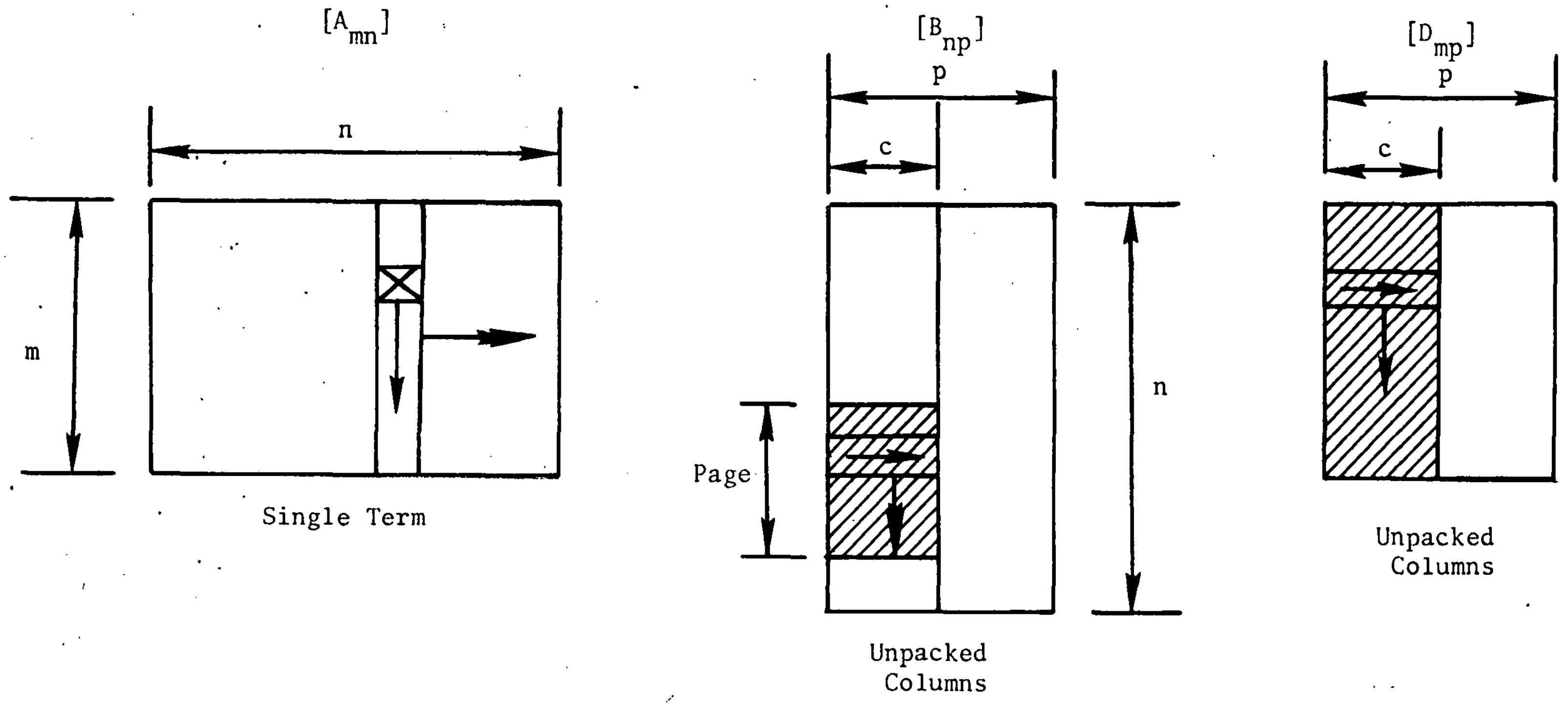

Figure 7. Modified Method 1 MPYAD

If SYSTEM(V) = 0, no virtual memory is available, and the code would operate in the usual way. Multiple buffers should not be used in the case of a single pass (c=p), because in this case there is never any real advantage and c may be small enough to cause high paging rates.

If the full address space is used for Methods 2 and 3, the paging rates shown in Table 5 are much less than those for Method 1, but they still exceed allowable values. The paging rates for both Methods 2 and 3 can be reduced to negligible values by assigning working space based on the difference between CØRSZ and SYSTEM(V). This modification will cause both Methods 2 and 3 to perform in about the same way as on a real memory machine. If the left-hand matrix is dense, the number of passes will be large and the wall clock time will be too large for good performance in a virtual environment. The wall clock time can be reduced by providing multiple buffers in the virtual address space for both the right-hand matrix and the result matrix when the number of passes is greater than one.

Changes for Decomposition Routines

The only change recommended for symmetric decomposition is to refer to SYSTEM(V) in allocating working space in order to control the paging rates. The spill logic in the current symmetric decomposition will operate very well in virtual memory systems, as long as the working space is limited to some reasonable fraction of the total available real memory.

The same procedure for allocating working space should be used for unsymmetric decomposition. In this case the spill logic is not nearly as effective as the new symmetric decomposition, because of the relatively large number of transfers between main storage and secondary storage. However, no further changes are suggested for unsymmetric decomposition. If a new unsymmetric decomposition is designed, consideration should be given to virtual storage operations at that time.

Changes for FBS Routines

The paging rates for the FBS routines can be controlled by using the existing spill logic to hold several columns from the right-hand side in the working space. The paging rates will be negligible, but the wall clock times will be very large in the case of multiple passes. The wall clock times can be reduced by providing multiple buffers for the triangular factor in the virtual memory space.

If a single column of the right-hand side is held in the working space and multiple buffers are provided for the triangular factor, the paging rate for the triangular factor is calculated as follows:

$$\text{Total Number of Pages} = \frac{2NCL}{P}$$

$$\text{Total CPU Time} = 2NCLM$$

$$\text{Paging Rate} = \frac{1}{PM}$$

If P = 500 nonzero terms per page and M = 10 microseconds, the resulting paging rate is 200 pages per second. The total number of pages and the paging rate are reduced in direct proportion to the number of columns of the right-hand side held in the working space for each pass. If working space is provided for 20 columns, the paging rate is reduced to 10 pages per second.

The number of columns on the right-hand side used for each pass can be controlled by the user with his specification of the working space. The program determines the amount of working space by referencing SYSTEM(V) and CØRSZ. If there is insufficient buffer space in virtual memory for the complete triangular factor, all available space will be used, and the remaining part of the factor will be transferred from secondary storage to main memory in the usual way.

Changes for Transpose Routine

The current transpose routine in NASTRAN is not effective for sparse matrices and should never be used for this case. Sparse matrices should always be transposed by using multiply/add with the transpose option operating on an identity matrix. The only change recommended for the current transpose routine is to use SYSTEM(V) in determining the amount of working space. This procedure will avoid the high paging rates discussed in previous sections. The routine will operate with the regular spill logic in those cases where the complete matrix to be transposed cannot be held in the available working space.

Changes in the Executive Operations

No changes are suggested for any of the executive modules except XSØRT. The maximum working space required by all other executive routines is not large and in any event these modules do not have long running times, even for large problems.

The XSØRT routine should be modified to reference SYSTEM(V) in order to determine the amount of working space available. This will prevent the extremely high paging rates which would be associated with the use of virtual memory for sorting operations. If the total address space were used for XSØRT, the paging rates could easily reach tens of thousands of pages per second. This high paging rate results from the poor locality of reference and the poor density of reference associated with sorting operations.

The existing spill logic in NASTRAN confines all references to a well-defined working space.

Changes for Functional Modules

Very few direct changes are suggested for any of the functional modules. Reference to Table 4 indicates that several of the functional modules (Group 0) have virtually no working space requirements. No changes are recommended for any of the functional modules in Group 0.

Many of the functional modules are in Group 3 for which the working space and paging rates are controlled by one or more matrix routines. No modifications are needed for any of the modules in this group. The suggested modifications for the basic matrix routines will automatically provide for efficient operation of the functional modules in Group 3.

The functional modules in Group 1 have working space requirements that do not exceed eight times the number of grid points. None of the modules in this group are provided with spill logic. Since other routines, such as FBS and MPYAD, will require working space for several vectors, the working space for modules in Group 1 can be expected to be less than that required by other operations. Consequently, no changes are recommended for any of the functional modules in Group 1.

All functional modules in Group 2 with spill logic are likely to have very high paging rates for large problems. However, the paging rates for these modules can be easily reduced to negligible values by using the existing spill logic. The only coding change required is to reference both SYSTEM(V) and CØRSZ when the amount of working space is determined.

The only remaining functional modules are those in Group 2 without spill logic. All of these routines compare the working space requirement with that which is available, and in real memory machines issue a fatal error message if there is insufficient working space. In order to prevent thrashing on virtual machines, the same test should be made and a fatal message should be issued if the working space is insufficient. The amount of working space available should be determined by referencing both SYSTEM(V) and CØRSZ.

The working space requirement for most of the functional modules is the same on virtual machines and real machines. However, the inverse power option for eigenvalue analysis is an important exception to this rule. For example, in the case of real eigenvalue analysis working space is required for seven vectors on real memory machines, even though only two are used at any one time. Consequently, on virtual machines working space is needed for only two vectors. However, there will be local high paging rates unless working space is provided for four vectors. This is because four of the vectors are used several times in each iteration.

The vector operations in NASTRAN, such as those used in the inverse power method of eigenvalue extraction and in direct transient response require special consideration. The wall clock time for these operations is rather long because the triangular factors are transferred between secondary storage and main storage during each iteration. For direct transient response, this may be as many as a few thousand times. It is tempting on virtual storage machines, particularly for smaller problems, to place the triangular factors in virtual storage and thereby substantially reduce the wallclock time for these vector operations. However, this should not be done, because the paging devices cannot support the paging rate required for this application. The paging rates for equation solution with a single right-hand side were calculated in the section on changes for FBS routines. Since these paging rates cannot be supported by available devices, the operating system will deactivate the job every time it tries to execute. The result will be that the job will never execute as long as any other job is on the machine. If it is the only job on the machine, it will execute, but the CPU will operate at extremely low efficiency.

Cost Estimate for Suggested Changes

The effort required to design, code and implement all suggested changes is as follows:

	Modification	Man Days
1.	Revise address space acquisition	25
2.	Establish SYSTEM(V) to define working space	1
3.	Align GINØ buffers on page boundaries	2
4.	Modify MPYAD to use SYSTEM(V) and multiple buffers	3
5.	Modify all decomposition routines to use SYSTEM(V)	1
6.	Modify all FBS routines to use SYSTEM(V) and multiple buffers	3
7.	Modify transpose routine to use SYSTEM(V)	1
8.	Modify XSØRT to use SYSTEM(V)	1
9.	Modify PLØT and all functional modules in Group 2 of Table 4 to use SYSTEM(V) - 40 modules	7
	Total to design and code	44
	System integration and documentation	22
	Total for Project	66

Required computer time is 3 CPU hours on an IBM 370/168.

RECOMMENDED USER PRACTICES

The minimum change required to make NASTRAN operational under VS operating systems is to modify the procedure for the acquisition of address space. If this is the only modification made the user must restrict his address space to some reasonable fraction of the total available real memory. The larger the fraction of real memory requested for address space, the greater will be the tendency to begin thrashing and the longer will be the wall clock time. A reasonable choice for address space is to use the same region that would have been used for the same problem on a real memory machine.

If all of the recommended changes are made, the maximum available address space should be requested on the JOB card. For VS2 Release 2, the maximum address space is 16 megabytes. For other operating systems, the maximum address space will be much less than 16 megabytes. The exact amount available will depend on installation procedures. In selecting the address space, consideration should also be given to the amount of hardware available to provide the virtual storage space and to any cost penalties that might be associated with the use of a large virtual address space.

The amount of real memory space to request on the EXEC card depends on problem size. This real memory request is not directed at the operating system but rather at the NASTRAN code. This is the way in which the user tells the NASTRAN code that if all program operations are organized to limit the working space to the real memory specified, there will be no thrashing and the wall clock time for execution will be reasonable.

The real memory request should be the maximum of the following in bytes:

1. Statics problems - 175,000 + 48G
2. Dynamics problems - 150,000 + 32G
3. Plotting operation - 200,000 + 32G
4. FBS and MPYAD operations - $150{,}000 + \frac{B}{RPM}$ or 150,000 + BN

where G = total number of grid points

B = number of bytes in largest unpacked column used in FBS or MPYAD

N = maximum number of columns on right-hand side in FBS or matrix in MPYAD

R = allowable paging rate in pages per second

M = time for unit multiply/add operations in seconds

P = number of bytes per page

For small problems the amount of real memory will be controlled by the first or second expression above, depending on whether the problem to be solved is a statics problem or a dynamics problem. If plotting operations are requested for small problems, the third expression will probably control the real memory request. Large problems usually involve significant MPYAD or FBS operations and the fourth expression will usually control the real memory request. If the Inverse Power Method of eigenvalue extraction is used, the 32G term for dynamics problems should provide space for at least two double precision vectors. The above working space allows for half track GINØ buffers on 3330 disk files and a modest amount of code to be paged out during execution. In order to avoid thrashing, the working space must always be less than the total amount of real memory on the machine. The wall clock time for execution may be very long if the working space is a large fraction of the available real memory.

CONCLUSIONS

1. The minimum change required to make NASTRAN operational under virtual storage operating systems is to modify the procedure for acquiring address space.

2. The performance of NASTRAN can be improved on virtual storage machines if both the address space and the working space are considered by the program.

3. For small problems, the real memory actually used by NASTRAN at any time will usually be less than 200K bytes.

4. The maximum amount of real memory required for the efficient solution of large problems will be smaller than on real memory machines.

5. The current NASTRAN code can be easily modified to give good performance with virtual storage operating systems.

6. If NASTRAN is improperly used with virtual storage operating systems, the performance will be very bad.

7. Hardware paging cannot be exclusively used for large problems because the available paging devices cannot support the resulting paging rates.

BIBLIOGRAPHY

Anon., NASTRAN Programmer's Manual, NASA SP-223, September 1972.

Anon., IBM System/370 System Summary, IBM GA22-7001, May 1973.

Anon., IBM System/370 Principals of Operations, IBM GA22-7000, January 1973.

Anon., IBM System/370 Model 145 Functional Characteristics, IBM GA 24-3557, September 1974.

Anon., IBM System/370 Model 158 Functional Characteristics, IBM GA 22-7011, April 1973.

Anon., IBM System/370 Model 168 Functional Characteristics, IBM GA 22-7010, July 1972.

Anon., IBM Virtual Machine Facility/370: Introduction, IBM GC 20-1800, March 1974.

Anon., IBM Virtual Machine Facility/370: Planning and System Generation Guide, IBM GC 20-1801, March 1974.

Anon., ØS/VS Virtual Storage Access Method (VSAM) Planning Guide, IBM GC 26-3799, June 1973.

Anon., ØS/VS1 Planning and Use Guide, VS1 Release 3, IBM GC 24-5090, December 1973.

Anon., ØS/VS2 Planning and Use Guide, VS2 Release 1, IBM GC 28-0600, September 1972.

Anon., Introduction to ØS/VS2 Release 2, IBM GC 28-0661, March 1973.

Anon., IBM Virtual Machine Facility/370: Release 2 Guide, IBM GC 20-1815, March 1974.

Anon., ØS/VS2 Planning Guide for Release 2, IBM GC 28-0667, November 1973.

BIBLIOGRAPHY (Continued)

Anon., ØS/VS2 System Programming Library: Initialization and Tuning Guide, IBM GC 28-0681, February 1974.

Anon., ØS/VS2 Auxilliary Storage Management Logic, Release 2, IBM SY 35-0009, January 1974.

Anon., ØS/VS2 VIØ Logic, Release 2, IBM SY 26-3834, September 1974.

"The aeronautical and space activities of the United States shall be conducted so as to contribute . . . to the expansion of human knowledge of phenomena in the atmosphere and space. The Administration shall provide for the widest practicable and appropriate dissemination of information concerning its activities and the results thereof."

—NATIONAL AERONAUTICS AND SPACE ACT OF 1958