

## **General Disclaimer**

### **One or more of the Following Statements may affect this Document**

- This document has been reproduced from the best copy furnished by the organizational source. It is being released in the interest of making available as much information as possible.
- This document may contain data, which exceeds the sheet parameters. It was furnished in this condition by the organizational source and is the best copy available.
- This document may contain tone-on-tone or color graphs, charts and/or pictures, which have been reproduced in black and white.
- This document is paginated as submitted by the original source.
- Portions of this document are not fully legible due to the historical nature of some of the material. However, it is the best reproduction available from the original submission.

STUDY OF THE MODIFICATIONS NEEDED  
FOR EFFICIENT OPERATION OF NASTRAN ON  
THE CONTROL DATA CORPORATION  
STAR-100 COMPUTER

By Aerospace Division of  
Control Data Corporation

(NASA-CR-132644) STUDY OF THE MODIFICATIONS  
NEEDED FOR EFFICIENT OPERATION OF NASTRAN ON  
THE CONTROL DATA CORPORATION STAR-100  
COMPUTER (Control Data Corp.) 85 p HC \$4.75  
CSCL 09B G3/61

N75-24382

Unclas  
20732



Prepared Under Contract No. 6-74-490-H by  
CONTROL DATA CORPORATION  
Arden Hills, Minnesota

Subcontractor to Contract NAS1-12436  
NASA SYSTEM DESIGN STUDIES PROGRAM OFFICE  
MCDONNELL DOUGLAS ASTRONAUTICS COMPANY

## CONTENTS

<b>SUMMARY</b>	<b>1</b>
<b>INTRODUCTION</b>	<b>1</b>
<b>FEASIBILITY</b>	<b>6</b>
Machine-Independent Code	6
FORTRAN Compiler	6
FORTRAN Library	6
Vector Processing	7
Machine-Dependent Code	8
Loading and Execution	8
STAR Static Loader	9
NASTRAN Controller and Drop Files	10
Reflection of Linkage to Star General	11
NASTRAN Input/Output on STAR	12
STAR Operating System I/O	12
STAR FORTRAN I/O	14
NASTRAN Generalized I/O (GINO)	14
Transition Method	15
<b>STAR TECHNOLOGY</b>	<b>16</b>
Virtual Memory	17
Vector Processing	18
Paging	20
Estimate of NASTRAN/STAR-100 Paging Rate	23
Virtual Space Manager	26
Non-I/O NASTRAN/OS Interface	26
Virtual Space Mapping	26
Virtual Subfile Structure	27
I/O Connector Management	27
Exception Processing	28
<b>IMPROVED EFFICIENCY</b>	<b>28</b>
Matrix File Structure	28
Sparse Matrix Indexing Schemes	30
General Outline of Use of Double Matrix File	30
Ordering Matrix Coefficients	32
Statement of Magnitude of Improvement	33
Modes of Operation	33
NASTRAN Stand-Alone Option	33
The "Front-End" Concept	34
Interactive NASTRAN	36

**ORIGINAL PAGE IS  
OF POOR QUALITY**

Extended Features for Efficient Processing	37
Half-Word Operands	37
Vector Macro Instructions	39
String Operations	39
Alternate Hardware Configuration	39
Limitations of Size of Problem	40
<b>MAINTENANCE CONSIDERATIONS</b>	41
Conversion Handling	42
Maintenance Handling	44
Inter-Machine Handling	44
Operational Reliability	46
<b>MANPOWER ESTIMATES</b>	46
Operating System Modifications	47
Batch Processor	47
Loader	47
Debugging Aids	47
Compiler Modifications and Extensions	47
Non-Standard Returns Label	48
FORTRAN IV Library Routines	48
Program Card	48
Half-Word FORTRAN	48
Pseudo-Operating System Features	49
Virtual Space Manager (VSM)	49
Indexed Sequential Record Management	49
Physical Space Allocation/Deallocation	49
Machine-Independent Code Conversion	50
Machine-Dependent Code Conversion	50
NASTRAN Optimization	50
Record Management	50
Matrix File Structure	51
Half-Word Operands	51
Maintenance	52
Cost Breakdown	52
Basic Conversion	52
Matrix File Structure and Vector Processing	52
<b>CONCLUSIONS</b>	58
<b>GLOSSARY OF STAR OS TERMS</b>	59
<b>APPENDIX A SCALAR CPU TIMING COMPARISON BETWEEN</b>	
6600 AND STAR-100	63
<b>APPENDIX B STIFFNESS MATRIX DECOMPOSITION TIMING</b>	67
<b>BIBLIOGRAPHY</b>	81

## FIGURES

Figure 1.	Symbolic Breakdown of Virtual Space for NASTRAN	3
Figure 2.	Structure of NASTRAN Controllee File	10
Figure 3.	Typical Flow of "Block of File Open for Implicit I/O"	13
Figure 4.	Associative Word Format	17
Figure 5.	Schematic of Vector Operand Processing Units	19
Figure 6.	Register Mode - Sparse Linear Form ( $C=A+\Theta B$ )	24
Figure 7.	Schematic of Sparse Matrix ERT Step	31
Figure 8.	Half- and Full-Word Operand Formats	38
Figure 9.	General Flow of NASTRAN Subsystem Build	43
Figure 10.	Time Frame Necessary for Basic NASTRAN Conversion (Scalar NASTRAN)	56
Figure 11.	Vector Processing	57
Figure B-1.	Structural Finite Element Analysis	70
Figure B-2.	Algorithms	71
Figure B-3.	Programming Procedures Used	72
Figure B-4.	Control Data 6600 CPU Timing	73
Figure B-5.	CPU Time - CDC 6600 Full Machine	74
Figure B-6.	Formula Validation Comparisons with CDC 6600 Benchmarks	75
Figure B-7.	STAR-100 CPU Timing	76
Figure B-8.	Full Machine Utilization - CPU Time of STAR-100 vs. CDC 6600 (Variable Band Algorithm)	77
Figure B-9.	Partial Machine Utilization - CPU Time of STAR-100 vs. CDC 6600 (Variable Band Algorithm)	78
Figure B-10.	Full Machine Utilization - CPU Time of STAR-100 vs. CDC 6600 (Constant Band Algorithm)	79
Figure B-11.	Partial Machine Utilization - CPU Time of STAR-100 vs. CDC 6600 (Constant Band Algorithm)	80

## TABLES

Table 1.	Modules Requiring Prefaces for File Size Determination	16
Table 2.	Order Vector Sizes	30
Table 3.	NASTRAN-STAR-100 Adaptation	54
Table A-1.	6600/STAR-100 Scalar CPU Comparison	63

**STUDY OF THE MODIFICATIONS NEEDED  
FOR EFFICIENT OPERATION OF NASTRAN ON  
THE CONTROL DATA CORPORATION STAR-100 COMPUTER**

**By Aerospace Division of Control Data Corporation**

**SUMMARY**

NASA Structural Analysis (NASTRAN) computer program is operational on three series of third generation computers.

This study was conducted to determine the magnitude of the problem and difficulties involved in adapting NASTRAN to a fourth generation computer, namely, the Control Data STAR-100. The salient features which distinguish Control Data STAR-100 from third generation computers are hardware vector processing capability and virtual memory.

This study reveals a feasible method of transferring NASTRAN to Control Data STAR-100 system while retaining much of the machine-independent code. Further, basic matrix operations are noted for optimization for vector processing.

**INTRODUCTION**

This report presents an unbiased investigation of the feasibility of modifying NASTRAN to execute efficiently on the Control Data STAR computer. The objectives of this study were as follows:

1. Identify areas in NASTRAN which easily lend themselves to conversion to STAR or which could cause problems in conversion to the STAR computer, and describe the areas in each of these categories.
2. Determine the areas of NASTRAN which will be affected by either (or both) virtual storage or the pipeline processor, describe these areas, and discuss any improvements that might be made.

**ORIGINAL PAGE IS  
OF POOR QUALITY**

3. Determine the areas of NASTRAN where modifications are needed to improve efficiency and where significant benefits could be expected from using new strategies or algorithms, and describe these areas and their benefits.
4. Determine if the above changes can be accomplished so that the efficiency of NASTRAN can be improved with little or no increase in the number of computer-dependent subroutines, and explain the purpose of all needed new computer-dependent subroutines.
5. Estimate the time and cost involved in designing, coding, and implementing each of the modifications identified above.

At the outset, several assumptions and choices were made that did not unnecessarily restrict the scope of the study, but did allow a fruitful concentration of effort. These guidelines were:

- Begin with level 15.7.8 NASTRAN
- Convert all of NASTRAN to STAR
- Assume batch processing mode
- Assume multiprogramming STAR environment
- Use standard STAR software
- Assume paging drum station in configuration

These guidelines underlie all of the analysis and recommendations presented in this report. The interesting alternatives to these assumptions and their consequences are mentioned throughout the report as exceptions or as alternate design strategies in the section describing Improved Efficiency.

**ORIGINAL PAGE IS  
OF POOR QUALITY**

Based on the stated conclusions and the recommendations in later sections, the initial conversion of NASTRAN can be described. The design retains NASTRAN's existing modularity to allow for later efficiency improvements. NASTRAN will be executed as a subsystem out of one virtual space. Figure 1 shows the functional breakdown of this virtual space.

Reserved by STAR Operating System	Virtual Bit Address $2^{48}-1$
Virtual Files	Virtual Bit Address $2^{47}-1$
Dynamic Space	
I/O Buffers	
Open Core	
Blank Common	
Error Processing Information	
Labeled Common	
NASTRAN Modules and Data Bases Internal to Modules	
Indexed Sequential with Actual Key Option I/O Package	
Disk Space Allocation/Deallocation (Prologue and/or Epilogue for Each Module)	
Virtual Space Manager (i.e., Job Mode Monitor)	
Page Zero (Register File)	Virtual Bit Address 0

Figure 1. Symbolic Breakdown of Virtual Space for NASTRAN

All of NASTRAN will be executed as a single controllee file. This greatly simplifies the STAR OS/NASTRAN interfaces and allows the natural modular structure of NASTRAN to predominate. In other words, we will dispense with the link structure of the present NASTRAN systems.

To centralize the direction of the orderly use of virtual space, a virtual space manager (VSM) will be developed. The primary responsibility of VSM will be to control and record the association of virtual space and disk space and to ensure that virtual space is not used in conflicting ways, e.g., that debugging information does not destroy a virtual file.

A fundamental precept of the STAR Operating System is that all used virtual space has real disk space associated with it; and furthermore, this disk space must be assigned at file creation time.

There is a system imposed limit of 18 open files for a given task. Some of these files are dedicated to special uses, e.g., controllee file, drop file, standard input file. This limitation will require development of a technique to distribute the 19 permanent files and 35 scratch files in current NASTRAN usage among the available STAR files. A prolog and/or epilog to each module is necessary to perform this association and disassociation of data blocks and disk file space. Clearly, all data set sizes cannot be exactly specified but reasonable bounds can invariably be determined for them with some specified degree of confidence. A recovery procedure can be defined so that occasionally when data set sizes do exceed the specified sizes, the NASTRAN run can continue with only time devoted to stacking and unstacking (pooling) of various data sets.

The existing concepts concerning input/output within NASTRAN remain valid and shall be retained and extended. This gives enormous flexibility in tuning the subsystem, e.g., the existing GINO calling sequence can be retained and

yet allow the module developer the choice of using virtual I/O or explicit I/O. The concept of open core shall be applied to virtual space. By experimenting with the size of open core, the effects of running NASTRAN in a mode assuming essentially infinite core may be compared with the effects of running NASTRAN in a more traditional way with a smaller fixed amount of real memory.

All NASTRAN files (or data blocks) will be assigned virtual space. This allows the NASTRAN subsystem to be checked out very early in the development stage, because of the ease of handling virtual files. The responsibility for the management of virtual files rests, in large part, with the STAR Operating System. For most types of files, virtual I/O cannot compete with explicit I/O, so an option will be made for the module writer to declare whether a file should be used explicitly or implicitly.

On the other hand, for fairly small (i.e., fits within core/drum system) truly randomly accessed files, virtual I/O is far superior to explicit I/O in the STAR system. Aside from the usual sequential file manipulation, the module writer shall have the following I/O options:

- a) In-core files (always virtual)
- b) Indexed sequential subset (either virtual or explicit)

Proceeding this way allows one to take maximum advantage of preceding work while retaining modularity and transportability. One's resources may be concentrated on areas of potentially high payoff and one has real flexibility in attacking these areas.

## FEASIBILITY

The objective of Study Phase 1 was to identify areas in NASTRAN which easily lend themselves to conversion to STAR or could cause problems in conversion to the STAR computer and to describe the areas in each of these categories. The study concludes that the conversion of NASTRAN to the STAR computer is feasible. STAR FORTRAN contains FORTRAN IV as a subset with only minor exceptions.

The major conversion effort is concentrated in the design and coding of machine-dependent routines. The load and execution process will be easier to use and maintain with standard software instead of the cumbersome linkage editor. The execution time I/O and system interface routines pose the most challenge to the conversion process.

### Machine-Independent Code

This section identifies problems with conversion of NASTRAN machine-independent code. The areas of concern are the FORTRAN compiler per se, the FORTRAN library, and vector processing.

#### FORTRAN Compiler

The STAR FORTRAN compiler encompasses the NASTRAN defined subset of FORTRAN IV with one exception: the use of the ampersand symbol (&) in a calling sequence to signify a nonstandard return label. FORTRAN IV specifies the symbol is to be a dollar sign (\$). Specifically, the nonstandard returns and multiple entry features of STAR FORTRAN are compatible with IBM/UNIVAC format. Thus the use of multiple-entry point driver decks can be eliminated.

#### FORTRAN Library

Mathematical subroutines defined in FORTRAN IV which do not appear in STAR FORTRAN are:

ERF (apparently unused in NASTRAN)

GAMMA (apparently unused in NASTRAN)

**ALGAMMA** (apparently unused in NASTRAN)

**COTAN**

Machine indicator tests defined in FORTRAN IV which do not appear in STAR FORTRAN are: **SLITE, SLITET, SSWITCH, OVERFL, and DVCHK.**

Other subroutines which do not appear in STAR FORTRAN are: **EXIT, DUMP, and PDUMP.**

**Vector Processing**

There are a number of routines which are presently machine independent, which should be modified to take advantage of the vector processing capabilities of STAR-100. These routines are:

**DMPY**

**FBS1**

**FBS3**

**SDCOM1**

**SDCOM3**

**MPY3T**

**MPYL**

**MPLLT**

**MPYLZZZ**

The machine-independent routines **FBS2, FBS4, SDCOM2, and SDCOM4** are probably not necessary to convert to machine-dependent code. These are the double-precision counter parts of **FBS1, FBS3, SDCOM1, and SDCOM3**, respectively. This is based on the assumption that 47 bits of mantissa is adequate for arithmetic computation in NASTRAN.

## Machine-Dependent Code

Converting machine-dependent code is less related to language compatibility and more to system interface compatibility. Thus routines like MAPFNS, PAKUNPK, and MPYQ should convert readily to STAR.<sup>†</sup> However, GINO and loader packages which interface to the operating system present some technical compatibility problems. These areas of NASTRAN require not only rewriting, but redesign for a different environment.

## Loading and Execution

It is grossly estimated that all of NASTRAN executable code and labeled common areas could be contained in a controllee file consisting of 250,000 to 500,000 54-bit words. In this case, the option of running NASTRAN as one long controllee file, as opposed to a series of controller-controllees, in analogy with the present NASTRAN link structure would be superior for two reasons: less dependence on STAR OS and the fundamental substructure of NASTRAN becomes the natural one, i.e., the functional module rather than the artificial link structure.

Investigation into the possibility of executing as a series of controller-controllees uncovered a number of problems with this concept. First of all the controller-controllee relationship is a linear one; that is, it is not capable of handling a tree structure. Furthermore, the chain of controller-controllees is limited to four (this limitation could in principle be removed fairly easily<sup>††</sup>). The operating system limitation of 18 open files per controllee requires that a subfile structure be imposed upon the operating system file structure. Each controllee is executed out of its own virtual space. This reduces intercontrollee communication to messages via operating system calls and pool files.

---

<sup>†</sup> There is no assembly level compatibility between Control Data 6600 COMPASS assembler and STAR-100 META Assembler; these routines must be completely recoded.

<sup>††</sup> The maximum number of controllees permitted in a chain is not a parameterized system constant.

## STAR Static Loader

The mechanics exist for much flexibility in arrangement of code and common blocks within the STAR loader. However, there are some limitations on the quantity of such options. For example, one can use only 12 private library files. For smaller systems, this appears to be adequate. For NASTRAN this will enforce a less than natural arrangement of libraries for the generation of a controllee file.

Another limitation is the present LOAD card buffer of 2048 64-bit words, which is probably insufficient for NASTRAN. The options GRSP, GRLP, GROS, GROL for arrangement of subroutines and common blocks are very powerful; and heavy use of them will be made to ensure minimum working set sizes.<sup>†</sup> However, these options are very wordy and alternate methods for filling an enlarged LOAD card buffer must be made available.

There is an additional overhead which affects the size of a controllee file which should be recognized; this is error processing information. Except in unusual conditions it will not cause NASTRAN execution interferences. However, it will occupy a sizable portion of disk space. The number of 64-bit words which are devoted to this for each subroutine are  $8 + 3$  times the length of the external/entry table. Assuming a nominal one entry point and two externals per subroutine the length of the external/entry table is nine words and the number of error processing information words for the subroutine becomes 35. Further assuming, we have 1000 subroutines in a controllee file, the space devoted to error processing information is 35,000 64-bit words.

---

<sup>†</sup> Initially the method of grouping subroutines will take advantage of present analysis by examining the LINK tree structure of the various NASTRAN systems.

## NASTRAN Controllee and Drop Files

A comparison of Figures 1 and 2 illustrates there is not a one-to-one correspondence between virtual space used (or to be used) and disk space for the controllee file. This results because information that is as yet unspecified (e.g., blank common) may be carried in a compact way simply as an address range. This address range becomes a "map" entry in the "minus" page.

The minus page saves information pertinent to the execution of the controllee file for the operating system. The disk copy of the controllee file is unaltered. Pages which must be changed during the course of execution, such as minus page, labeled common, or data bases, are recorded on the "drop file." The drop file is also used to hold modified virtual pages which are not associated with any other virtual file. Thus the concept that all used virtual space must have real disk space associated with it remains intact.

<b>Error Processing Information</b>
<b>Labeled Common Areas</b>
<b>Data Bases for Subroutines</b>
<b>Relocated Code</b>
<b>Page Zero</b>
<b>Minus Page (OS Control and Virtual Space Maps)</b>

Figure 2. Structure of NASTRAN Controllee File

## Rejection of Linkage Editor Concept

The concept of a linkage editor was developed to overcome disadvantages associated with an early version of the loader for the Control Data 6400/6600 systems. These disadvantages were:

- Only two levels of overlay provided beyond the root segment.
- An explicitly called overlay segment, consequently requiring a known overlay structure when the program is coded.
- Overlay segment entered at only one point thus limiting downward calls.
- No facility to explicitly position named common blocks.
- Loading of overlay segments accomplished from a sequential file, thus causing unnecessary search time.

These disadvantages do not exist with the STAR Loader; specifically:

1. There is essentially an unlimited number of overlay levels (or ways of arranging the executable code) available within a controllee file.
2. The programmer describes the controllee file structure to the loader after the program is coded. The loader builds a linked executable file which can be executed many times without going through the loader. The system provides the virtual I/O to get the necessary pages into main memory on a demand basis.
3. Complete communication throughout the controllee file is maintained.
4. Loader directives may be used to explicitly position subprograms and named common blocks.
5. Map entries in the minus page of the controllee file provide rapid correlation between virtual addresses and logical disk addresses. Consequently, pages of the controllee file residing on disk are immediately available to the system paging algorithm.

## **NASTRAN Input/Output on STAR**

**This section explains the standard STAR software supported I/O methods, the methods necessary for NASTRAN operation, and a transition method between them.**

### **STAR Operating System I/O**

**The operating system allows for four basic types of I/O: user-managed implicit, system-managed implicit, buffered explicit, and unbuffered explicit.**

**User-managed implicit means implicit I/O which goes to/from a user-defined file. System-managed implicit means I/O activity to/from the drop file (which includes the recording of all used virtual space that does not correspond to virtual space recorded in a user-defined implicit I/O file). Buffered/unbuffered explicit I/O is simply the traditional method of directly ordering I/O to take place between specific central memory and peripheral addresses.**

**The operating system supports three basic peripheral storage devices: drum, disk, and magnetic tape.**

**The operating system supports the two basic hardware defined page sizes: small pages (512 64-bit words) and large page (65,536 64-bit words).**

**Full generality between the four I/O types, the three peripheral device types, and the two page sizes is neither desirable nor supported.**

**The rules governing the viable combinations and the major ramifications are stated as follows:**

- 1. Buffer sizes for explicit I/O may be either 1 through 24 small pages or one large page.**
- 2. Only explicit I/O may be used for reading/writing magnetic tape.**
- 3. The drum may only be used for implicit I/O of small pages.**

4. For large pages, implicit I/O occurs only between the disk and central memory, i.e., the paging drum is not used with large pages.
5. Disk files cannot be extended, i.e., space for a disk file must be allocated at file creation time. (The disk file space may be reduced but not increased.)
6. There is a limit of 18 active (i.e., open) files.

A typical usage pattern for a block of an existing read/write file is depicted in Figure 3.

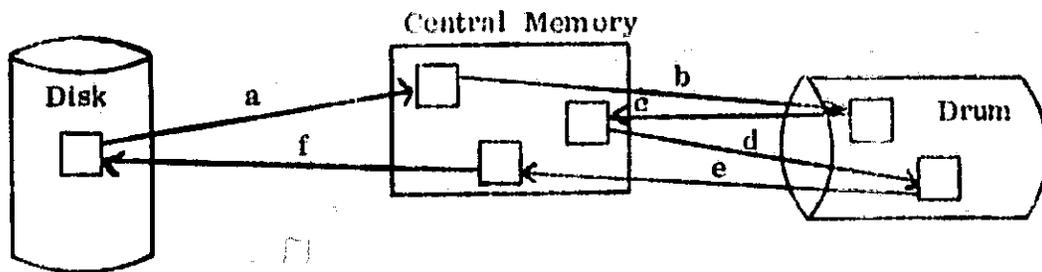


Figure 3. Typical Flow of "Block of File Open for Implicit I/O"

Paths such as b - e - d in Figure 3 may be repeated many times and are essentially beyond the control of the programmer. This figure shows the extra system overhead path e - f which occurs when using the paging drum for implicit I/O of small pages.

## STAR FORTRAN I/O

FORTRAN uses sequential record manager (SRM) as the interface with STAR Operating System.

When using unformatted I/O, the STAR record structure is used in blocking/unblocking the file.

STAR FORTRAN conventions require that all files using FORTRAN I/O be defined at compile time on a program card. This definition includes whether it is a tape or disk file.

BUFFER IN/OUT statements always use explicit I/O whether or not the file is to reside on tape or disk.

READ/WRITE statements (either formatted or unformatted) which refer to tape files always use explicit I/O.

READ/WRITE statements (either formatted or unformatted) which refer to disk files always use implicit I/O. An examination of Figure 2 again shows that for large sequential files this method of I/O is extremely inefficient.

FORTRAN (via SRM) places each defined implicit file in increments of 10,000,000 (hexadecimal) starting at virtual bit address 100,000,000 (hexadecimal).

## NASTRAN Generalized I/O (GINO)

Besides the usual sequential I/O, NASTRAN requires random access to two basic structures: tables and matrices.

Although direct access I/O is referred to in some of the NASTRAN maintenance literature, an examination of GINO documentation and code indicates that only indexed sequential I/O is presently used in NASTRAN.

Furthermore, it is stated that all GINO files may be accessed by record. Additionally, matrices may be accessed by subrecords of arbitrary length called strings. The purpose of the string manipulation is to allow the processing of matrices directly from GINO buffers.

A timely and cost effective conversion of NASTRAN to STAR requires that initially the calls to GINO from the functional module level remain the same.

### Transition Method

There are several important differences between NASTRAN and STAR I/O techniques. In NASTRAN there are hundreds of data blocks allocated for over 50 files, while the STAR Operating System provides less than 15 files for such allocation. In NASTRAN, files are open-ended while STAR Operating System requires allocation of file space at open time. In NASTRAN there are provisions (GINO) for random access methods employing indexed - sequential files with actual key processing options (SAVPOS, FILPOS), while STAR Operating System employs a simple sequential record manager. Thus, the significant I/O problems to overcome are briefly stated as:

The preallocation of optimal space for NASTRAN files

The many-to-one association of NASTRAN data blocks to STAR Operating System files

The provision for random access of NASTRAN data block substructures

The first two problems are handled by a preface to each NASTRAN module which analyzes the requirements for file space. Refer to Table 1 for modules requiring prefaces. Precise limits on space generally will not be possible, but with fairly simple logic one can develop estimates with a high degree of confidence which should be sufficient for most cases. It is suggested that the method of random I/O processing implemented be indexed-sequential (integer key) with option for actual key processing. This would initially be implemented for virtual files only. This allows a good version of NASTRAN to be implemented quickly without any "throw-away" code being necessary. Explicit I/O on random physical files will almost certainly improve performance and is considered necessary in any optimization effort.

**TABLE 1. MODULES REQUIRING PREFACE FOR FILE SIZE DETERMINATION**

BMG	GP4	PLTTRA	SMA3
CASE	GPWG	PRTMSG	SMPYAD
CEAD	MCE1	RANDOM	SMP1
DDR1	MCE2	RBMG1	SMP2
DDR2	MERGE	RBMG2	SOLVE
DECOMP	MPYAD	RBMG3	SSG1
DPD	MTRXIN	RBMG4	SSG2
DSMG1	OFF	REIG	SSG3
DSMG2	PARTN	SADD	SSG4
FBS	PLA1	SCE1	TA1
FRRD	PLA2	SDR1	TRD
GKAD	PLA3	SDR2	TRNSP
GKAM	PLA4	SDR3	VDR
GP1	PLOT	SMA1	XYPLOT
GP2	PLTSET	SMA2	XYTRAN
GP3			

#### STAR TECHNOLOGY

In Phase 2 of the study the impact of fourth generation computer technology on NASTRAN operation was analyzed. The two primary features studied were: virtual storage and its hardware/software implementation on STAR-100 and pipeline processing as it affects scalar and vector instructions on STAR-100.

The limited memory of third generation computers gave rise to many techniques for executing large problems with large data areas. One of the most successful applications of these techniques was the NASTRAN program which used extensive overlay structures to reduce the core requirements for executable code and used open core concepts along with packing/unpacking and spill to secondary storage devices for handling of large quantities of matrix data. From this perspective, virtual memory is just an extension and incorporation of third generation applications technology into fourth generation hardware and operating systems software.

## Virtual Memory

The method of mapping virtual memory into real memory in STAR-100 is accomplished by a combination of hardware/software features.

A collection of associative words provide the necessary linkage between real and virtual memory as shown in Figure 4 which is a functional diagram of an associative word. This collection of associative words is referred to as the page table.

Absolute Page Address	Usage Code	Lock	Virtual Page Identifier
-----------------------------	---------------	------	----------------------------

The usage code is defined as follows:

<u>Usage Code</u>	<u>Definition</u>
0	End of Page Table
1	Null associative word
2	512 - word page has not been referenced by the CPU
3	65K - word page has not been referenced by the CPU
4	512 - word page has been referenced by the CPU
5	65K - word page has been referenced by the CPU
6	512 - word page has been altered by the CPU
7	65K - word page has been altered by the CPU

Figure 4. Associative Word Format

The page table physically consists of a 16-register associative memory and the space table. The space table is a part of main core memory. The principal ramification of this is that "hidden" memory conflicts may occur during the course of execution of a program due to space table searches.

The page table is maintained as a push-down list (with the 16 associative registers at the top) by the hardware. If a hit is made, the referenced page address is automatically placed in the first associative register and the rest of the addresses moved down with the contents of associative register 16 becoming the first word in the space table. If a hit is not made (i.e., the referenced page is not in memory) the other addresses are again rippled downward and a null word is placed in the first associative register and an access interrupt is generated.

Rather than maintaining a page table for each active user, the STAR Operating System has adopted a global paging policy, which in effect removes the least recently used page in memory when a page replacement must be made.

### Vector Processing

Figure 5 is a schematic drawing of the four main vector processing units. They are:

- Pipe 1 Processes most arithmetic operations
- Pipe 2 Processes divides, square roots, binary-to-BCD conversion and supports Pipe 1 with some vector and vector macro operations
- String Processes bit, bytes, and decimal operations
- Logical Processes insert, extract, and logical operations.

The term vector processing includes more than the process of transforming data as it passes through the processing unit. The other major function in vector processing includes the storage and retrieval of data to/from main memory.

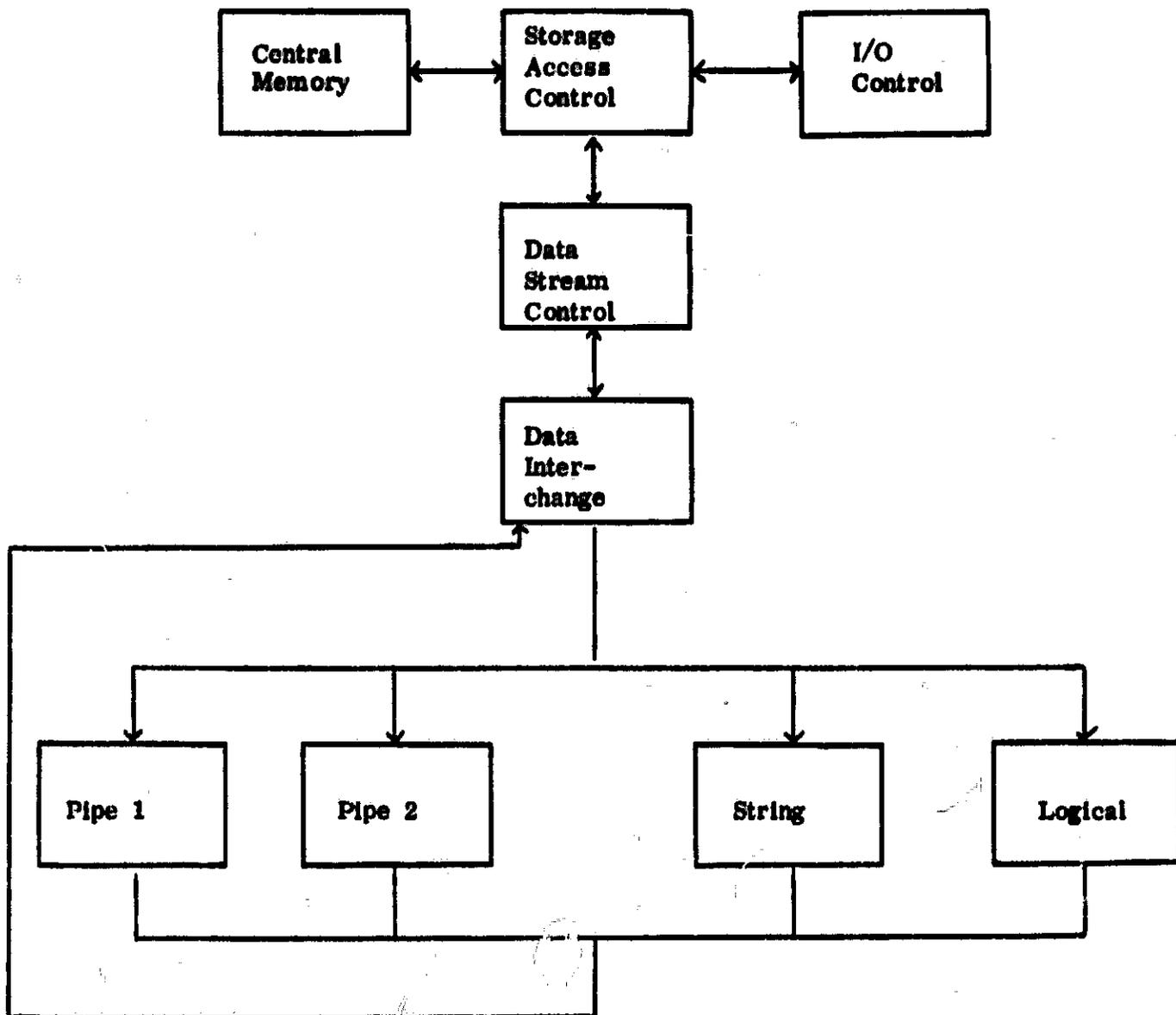


Figure 5. Schematic of Vector Operand Processing Units

**This process (streaming) is made feasible by the phased banks of main memory which allows the overlap of storage and retrieval functions, such that average storage/retrieval time for a group of consecutively stored items is much less than the actual time for one item.**

**In general, due to the arbitrary position of the relevant operands, there is a phase misalignment. The misalignment must be corrected before operands are sent to vector processing units to ensure correct results. This is handled by a complex method of buffering input and resultant operands.**

**This process can add significantly to the overhead of a vector instruction. (Overhead is commonly called vector start-up time.)**

### **Paging**

**In STAR, paging is a method used for associating virtual memory with real memory. Therefore, paging strategies and techniques are inextricably tied to consideration of virtual storage.**

**Few direct user controls over paging policy decisions are allowed. However, the user can influence paging policy by:**

- **Group option in controllee file construction**
- **Choice of explicit I/O for data files**
- **ADVISE command to attempt to override demand paging**
- **Choice of page size**

**The first two controls have been previously discussed.**

**In a stand-alone environment the ADVISE command provided by the STAR operating system to provide for prepaging and dispensing of old pages is a powerful tool for virtual file management. However, in a multiprogramming environment its use has been ineffective. Choice of page size is notable, because of the inherent conflict between the ideal page size for general purpose processing as opposed to page size for heavy vector processing. As an aid to understanding the problem and appreciating some of the difficulties involved, it is well to take**

a closer look at a typical vector instruction on STAR-100. Consider the following restricted vector operation:

$A = B + C$  where  $A, B, C$  are vectors conformable for vector addition in the usual mathematical sense: i.e., the  $i$ th element of  $A$ ,  $A_i = B_i + C_i$  for  $i = 1, \dots, n$ ;  $1 \leq n < 65,536$ .

The length of time this operation takes is dependent upon at least the following items:

- Stream rate; i.e., memory bandwidth in bits/cycle

- Operand size; i.e., 32 or 64 bits per word

- Length of vectors  $A, B, C$

- Relative bank position of  $A, B, C$

- Length of time operands are in pipe

- Small overhead for instruction initiate or reinitiate

- Bank phase relative to  $A, B, C$  upon initiation or reinitiation

- Number of pages that  $A, B, C$  occupy

- Distribution of  $A, B, C$  over occupied pages

- Page size of occupied pages (small page = 512 64-bit words, large page = 65,536 64-bit words)

- Position of associative words within page table

  - In associate register

  - In space table

  - Not in page table (page itself on drum)

  - Not in page table (page itself on disk)

- Paging policy

- External interrupts

If we attempt to maximize vector throughput over purely virtual time, we will choose page size of 65,536 to the detriment of everything else.

In trying to maximize vector throughput we must assume paging takes place and that there is a stiff penalty involved in occupying main memory when not in control of the CPU.

Apparently the only real freedom we have is in choice of page size, distribution of A, B, C over pages, and paging algorithm.

In summary, several major factors influence the page size determination problem in a scalar virtual machine, namely:

- Code organization
- Compression
- Transport time
- Page replacement algorithm

The introduction of the vector capability (via pipeline processing) into the hardware architecture creates additional factors which influence page size, namely:

Cost of halting a vector instruction to replace a page

Cost of restarting a vector instruction after a page fault

Vector length

These considerations lead to the seemingly strange choice of page sizes in STAR-100 of 512 64-bit words or 65,536 64-bit words and demonstrate an inherent conflict in choice of page size for scalar and vector processing.

Little practical experience has been obtained with large page sizes. Fortunately, the page size choices are flexible and easily modified since page size and placement are LOADER options.

With our assumption of NASTRAN executing in a multiprogramming environment, it is recommended that all NASTRAN code, data blocks, and buffers initially be small pages with open core fixed at one large page.

Undoubtedly experience will show a more nearly optimal arrangement which may be dependent upon problem size.

### Estimate of NASTRAN/STAR-100 Paging Rate

In an attempt to estimate paging rates which will be encountered during mathematical manipulation of data, we have chosen a sparse linear form operation as a composite representative of NASTRAN operations. The particular implementation we have chosen has the following characteristics (refer to Figure 6 for functional flowchart):

1. Given two sets of compressed values and two sets of corresponding indices and a multiplier, we form one set of compressed values and one set of corresponding indices, i.e.,  $C = A + \Theta B$  where  $A$ ,  $B$ ,  $C$  are sparse vectors,  $\Theta$  is a scalar, and  $I_A$ ,  $I_B$ ,  $I_C$  are the vectors of 16-bit indices corresponding to  $A$ ,  $B$ ,  $C$  respectively.
2. We assume a register mode implementation.
3. Each and every non-zero element requires a 16-bit index.
4. We use full-word arithmetic.

This method was chosen because it gives some estimate, however crude, of overhead involved in packing and un-packing vectors.

This method is only weakly dependent upon the number of vectors being used (i.e., initialization overhead is low). Therefore, extrapolation of the expected value of execution time used per iteration to the expected value of execution time per page of data can be made.

There are essentially three paths through this routine:

1.  $I_A(i_A) < I_B(i_B)$
2.  $I_A(i_A) = I_B(i_B)$
3.  $I_A(i_A) > I_B(i_B)$

where  $i_A$  refers to the  $i_A^{\text{th}}$  component of vectors  $A$  and  $I_A$  and  $i_B$  refers to the  $i_B^{\text{th}}$  component of vectors  $B$  and  $I_B$ .

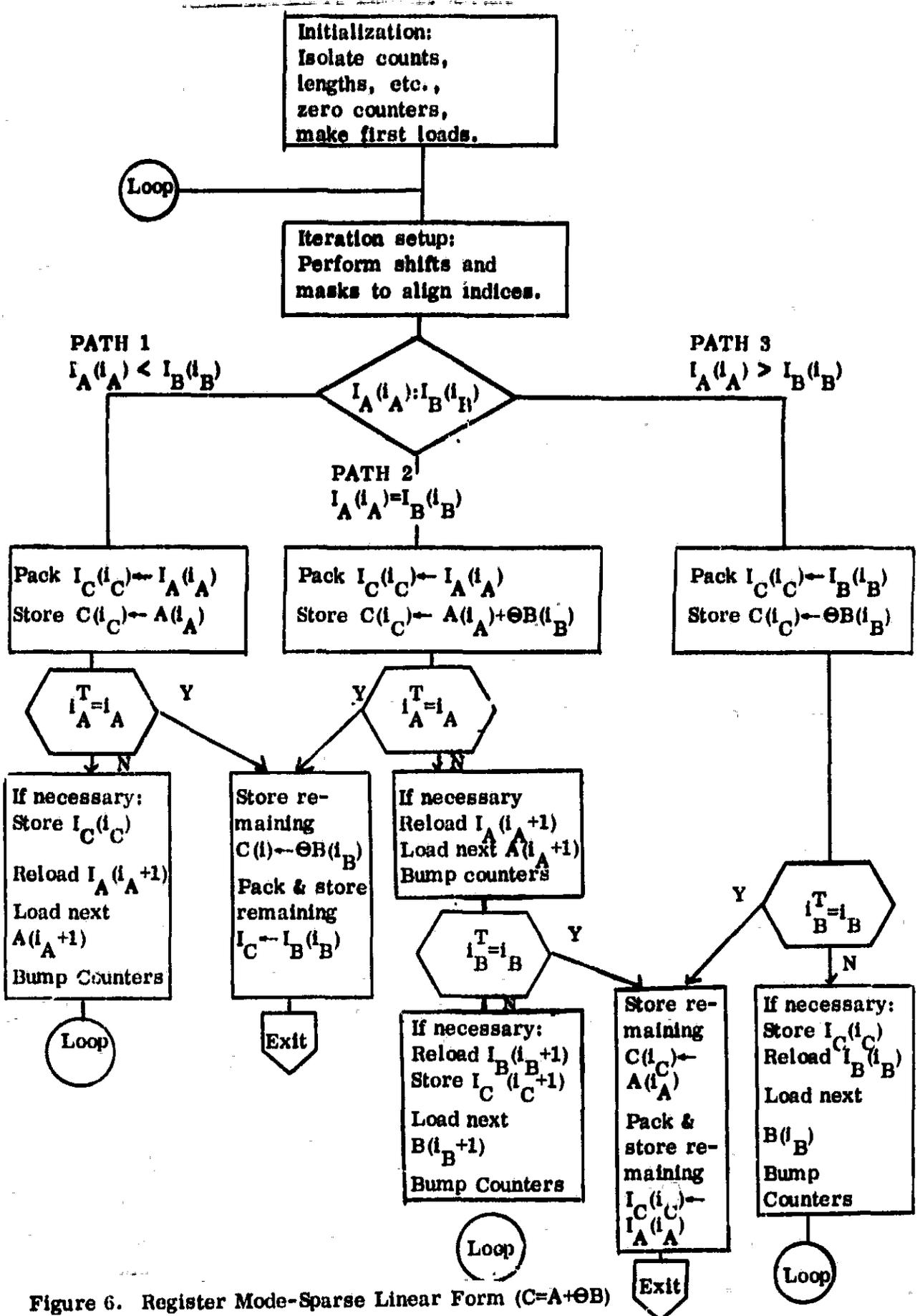


Figure 6. Register Mode-Sparse Linear Form (C=A+ΘB)

Hand timing of roughed-in code for these three paths yield:

Path 1 - 155 cycles

Path 2 - 157 cycles

Path 3 - 159 cycles

Within the limits of accuracy of our hand timing, we may assume that each path takes a nominal 160 cycles. At 40 nanoseconds per cycle this yields 6.4 microseconds per iteration (i.e., per resultant  $C(I_C)$  and  $I_C(I_C)$  values). Therefore, for 512 iterations we have 0.0033 seconds of expected compute time.

Let us further assume that path 2 is taken with a probability of  $2/3$ , and paths 1 and 3 each with probability of  $1/6$ .<sup>†</sup> This assumption means that the expected number of resultant elements (and indices) grow by  $1/3$ . Thus, when processing 512 iterations we have used  $3 \frac{1}{3}$  pages for values and  $3/4 + (1/4 * 1/3)$  pages for indices for a total of  $4 \frac{1}{6}$  pages of data in 0.0033 seconds.

This yields a nominal paging rate of 1250 pages/second. The random paging rates which can be sustained are on the order of 175 pages/second on the paging station (Control Data 865 drum) and 20 pages/second on the storage station (Control Data 844 disks). Explicit I/O and double buffering with large pages will alleviate this imbalance somewhat with respect to sequential files and the storage station. Using large pages from an 844 disk, one could reasonably expect a paging rate of 1.6 large-pages/second or approximately 100,000 64-bit words/second. This is equivalent to 195 small (but not random) pages/second.

---

<sup>†</sup> These assumptions seem reasonable as one does not typically deal with randomly scattered matrix elements but with strongly interacting rows and columns, most notably the banded matrices.

## Virtual Space Manager

The foregoing information leads to a proposal to develop a virtual space manager (VSM) to provide for the orderly transition of execution states in the NASTRAN subsystem. As such, the VSM functions more as a message switch and data traffic control than as an executive. It ensures, for example, that files do not inadvertently destroy one another, but it does not decide which module should be executed next. VSM is a single routine with many entry points and a collection of tables. There are five principal functions for which VSM is responsible:

- Provide the non-I/O NASTRAN/OS interface
- Maintain a map of virtual space
- Assign logical disk address to virtual files and maintain the minus page of the virtual files
- Manage I/O connectors
- Process exceptions

### Non-I/O NASTRAN/OS Interface

VSM provides a convenient location for centralizing and simplifying calls to system functions which are not provided via FORTRAN. These calls facilitate debugging and allow for the collection of performance measurement data. Typical of these calls are:

Send message to controller

Get message from controller

Miscellaneous system functions (number of page faults and I/O times)

### Virtual Space Mapping

VSM has the responsibility to provide a valid virtual address for NASTRAN files (operating system subfiles) when they are requested. In order to do this and to provide useful debugging information, a map of virtual space is maintained which includes all NASTRAN data blocks, dynamic space, code, data base, and the sequential record manager (SRM) provided locations.

### Virtual Subfile Structure

As previously mentioned, there are only a limited number of disk files available to a task (such as the execution of NASTRAN subsystem). Furthermore, a virtual file may contain no more than 40 noncontiguous pieces of virtual space. This leads to the subfile concept where each noncontiguous piece of virtual space becomes associated with a NASTRAN file or data block. For several reasons virtual files cannot be totally preallocated to data blocks; or conversely, data blocks cannot be assigned to virtual files. The reasons are:

1. Each tape is a file and the number of tapes may vary; therefore, number of files may vary.
2. There are too many data blocks even with 40 per virtual file with 13 available virtual files.
3. Reasonable size estimate on many data blocks may only be made at module execution time.

VSM, when informed of the data block requirements by a module prologue/epilogue (or disk space allocation/deallocation), will update the virtual files' bit maps and the records of data blocks and disk space.

### I/O Connector Management

I/O connectors provide the crucial link between physical file space, whether on disk or tape/or whether virtual or physical type files, and the executing program. A number of these connectors may be pre-empted for the life of the NASTRAN task. VSM notes these in order to avoid them. Others are constantly being created and destroyed while proceeding from one module to another module. It is these which are allocated and released by VSM.

### Exception Processing

A unified and flexible approach to the handling of abnormal conditions such as those indicated by the hardware data-flag register must be developed. This will require close coordination with the techniques used in the STAR FORTRAN library and the STAR operating system debug package. Data-flag register indicated conditions include:

Job internal timer

Breakpoint

Arithmetic faults

Some logical conditions (e.g., no true condition on vector logical "and")

Some search conditions (e.g., element not found)

### IMPROVED EFFICIENCY

Study Phase 3 was concerned with determining where modifications are needed to improve efficiency, and where significant benefits could be expected from using new strategies or algorithms.

As with third generation systems, a most important consideration for effective operation is keeping the CPU busy. This can be interpreted as getting extremely compact representation for the data base and managing and manipulating this data base in a manner which tends to localize references to specific portions of the data base. That is, locality must be exploited in data reference strings as well as in execution reference strings.

### Matrix File Structure

For the initial conversion effort a file structure similar to that used for matrices as described for IBM System/360 should be used. This enables one to quickly adapt the present NASTRAN to STAR-100 in a reasonably effective way. A better structure would be obtained by dividing the matrix files into two separate files. One file would contain all the control information such as column, row position, number of coefficients in a string, etc. The other

file would contain only matrix coefficients. This often enables one to operate directly on the coefficients without intermediate reorganization of the coefficients for efficient pipe-line processing. To see this, it is necessary to look closer at some of the hardware organizational constraints on streaming.

In central memory we have the following significant boundaries:

- Bit boundary
- Byte boundary
- Half-word boundary
- Full-word boundary
- Small page boundary
- Large page boundary

These boundaries are increasingly exclusive. By this is meant that a full-word boundary is also a half-word boundary and a small page boundary is also a full-word boundary but a half-word boundary is not necessarily a full-word boundary, etc. With one exception these boundaries must be rigorously adhered to. The exception is that for instructions only, full-word instructions may start on half-word boundaries. This is not true for data or in any other case.

When using vector instructions on STAR-100, the coefficient of each vector must be contiguous in memory and on appropriate half- or full-word boundaries.

When using the sparse vector instructions on STAR-100, the non-zero coefficient must be contiguous and on appropriate boundaries as with vector instructions. Additionally a sparse vector has associated with it an order vector which contains one bit for each element (whether zero or non-zero) of the sparse vector.

Table 2 shows that for a sparse matrix without discernable structure the order vectors alone could take an inordinate amount of space. Indeed, for a 16K x 16K matrix about 4 million 64-bit words would be required for order vectors alone.

Table 2. ORDER VECTOR SIZES

Row Size	Full Words for Each Order Vector
16K = $2^{14}$	256
8K = $2^{13}$	128
4K = $2^{12}$	64
2K = $2^{11}$	32
1K = $2^{10}$	16

#### Sparse Matrix Indexing Schemes

The indexing scheme (i.e., the method of associating a specific coefficient of a set with a particular coordinate) should attempt to take advantage of matrix structure. If the elements are truly randomly scattered, then a 16-bit index per element is efficient on STAR until the matrix density increases to about 6.25%. At that point the order vector scheme becomes more efficient.

#### General Outline of Use of Double Matrix File

In principle the use of two files for matrix reduction is quite simple. The many considerations, which can be taken into account, make the details of implementation very complex. For example, suppose one has a primary representation of a sparse matrix by columns and it is desired to make an elementary row transformation (ERT) on this matrix which results in a second sparse matrix. A step of forward transformation or elimination is shown in Figure 7.

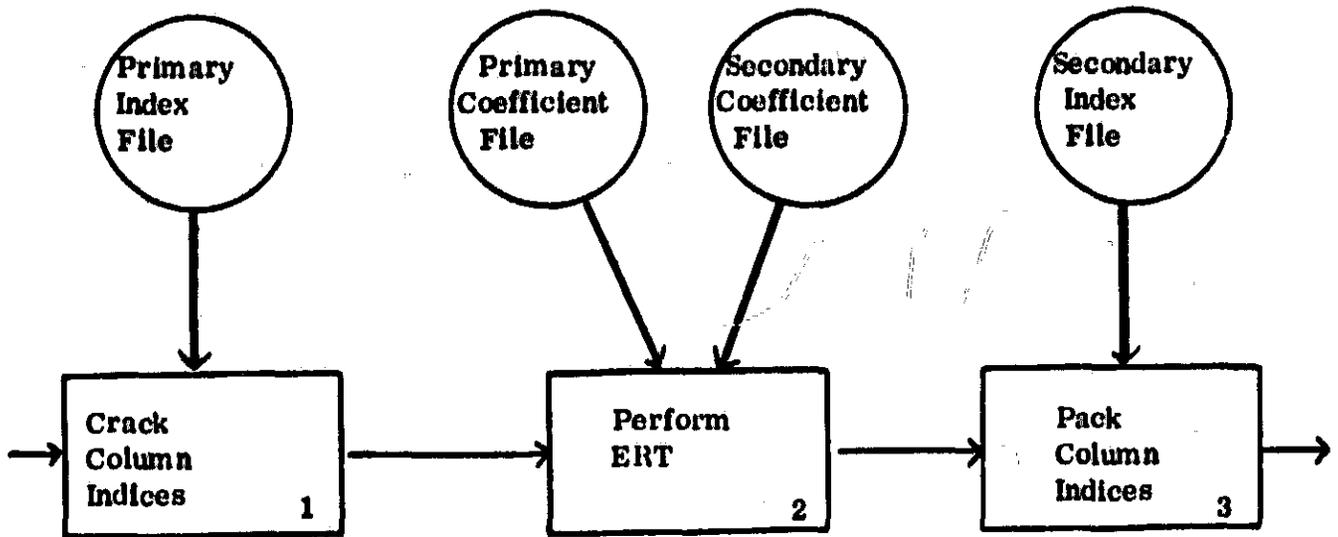


Figure 7. Schematic of Sparse Matrix ERT Step

Whether we consider this as a step in L-U Decomposition or Gauss-Jordan is immaterial at this level of discussion.

Mathematically we have  $EA_p = A_s$  where  $E$  is the identity matrix except for one column,  $A_p$  is the primary matrix and  $A_s$  is the transformed or secondary matrix. Represented in Figure 7 is just one step of this total transformation, i.e., we are performing the commonly used vector operation often referred to as "linear form":

$$Vs_j = V'_{pj} + \Theta E_c \quad \text{where } V'_{pj} \text{ is the } j\text{th}$$

column of the matrix  $A_p$  except that one element has been set to zero,  $\Theta$  is the scalar formed by making use of the element of  $V_{pj}$  before it was set to zero, and  $E_c$  is the column of  $E$  which differs from the identity matrix. Now, because the vectors  $Vs_j$ ,  $V_{pj}$ ,  $E_c$  are assumed sparse, we have much flexibility in details of executing the linear form. For example:

- a) If  $V_{pj}$  and  $E_c$  are very sparse, we may choose to make the computation in scalar mode,
- b) We may elect to perform the computation in sparse vector form,
- c) We may hold the vector  $E_c$  in expanded form and perform the computation with  $V_{pj}$  in compact form,
- d) If  $V_{pj}$  and  $E_c$  are relatively dense, we may elect to expand both  $V_{pj}$  and  $E_c$  and perform the computation over full vectors.

### Ordering Matrix Coefficients

In essence, one has four alternative ways of holding the compact coefficients of a matrix: randomly, by submatrix, by rows, or by columns.

Of course, it is possible to have variations and combinations of these four basic methods.

It may appear strange that random ordering would ever be useful. This method is used when there is a high percentage of common coefficients in a matrix. In this case the index file assigns the same value to different matrix coordinates.

Submatrix storage can be useful for matrix multiplication, but is not the best for equation solving.

It makes little difference, in the general case, whether we store matrices by rows or by columns. In cases where specific matrices are consistently used as pre-multipliers or as post-multipliers, it may be advisable to use row storage in the first case and column storage in the second case.

For the general case, it is recommended that the coefficient be stored by columns, because of the following reasons:

No other method has a clearcut advantage.

It has closer compatibility with existing NASTRAN storage techniques.

It is the commonly assumed method in the literature.

### Statement on Magnitude of Improvement

Performance comparisons with other machines were not undertaken for this study.

Appendix A compares scalar CPU performances on STAR-100 with CPU performances of a Control Data 6600.

Appendix B compares positive-definite, symmetric matrix decomposition CPU performance between Control Data 6600 and STAR-100. Full use of vector operations on STAR-100 was made in this comparison.

### Modes of Operation

Several alternate modes of operation provide some interesting options for the development of NASTRAN on the STAR computer.

### NASTRAN Stand-Alone Option

A basic design consideration is whether NASTRAN should be run "stand-alone" on STAR-100 or run in a multiprogramming environment. In order to run stand-alone, an attempt must be made to make full use of the drum/core system. In order to use demand paging (and depend upon multiprogramming to make efficient use of the system) then enough resources must be left over so that multiprogramming can occur usefully.

STAR Operating System is a multiprogramming operating system. This implies that STAR Operating System carries many overhead items that are unnecessary in uni-programming. It also uses more management of virtual resources than is desirable for stand-alone operations. In a stand-alone environment, the ADVISE command of STAR Operating System provides for prepaging and releasing old pages which is a powerful tool for virtual file management. (In a multiprogramming environment its use has not been effective). But the ADVISE command alone is not enough control over the core/drum system for uni-programming. The addition of explicit I/O to the drum would improve this situation.

In a previous section a reasonable estimate for a paging rate when running NASTRAN was given. This number was 1,250 pages/second. When comparing this number with the estimated paging rates for the storage station and paging station, (20 and 175 respectively), one concludes that the STAR-100 CPU would remain idle most of the time when running NASTRAN "stand-alone". The theoretical maximum paging rate attainable with the 865 drum is 660 pages/second.

If the STAR operating system were modified to support large pages and explicit I/O on the drum, then effective paging rates on the order of 500 small pages/second could be obtained.

#### The "Front-End" Concept

Another method of implementation of NASTRAN would involve transferring something less than the whole of NASTRAN from Control Data 6600 to STAR-100. That is, let STAR do what STAR does best and leave the rest on the 6600. The operations transferred to STAR could range all the way from selected matrix operations to selected functional modules.

A link between STAR-100 and the 6600 is currently planned to have the following characteristics: powerful physical connections (5 megabit lines) and unit record station functional software capability.

This implies that the operations selected for STAR should have modest input requirements, heavy compute and/or internal I/O requirements, and again modest output requirements.

Further, it seems that to be effective, a running program (NASTRAN) on the 6600 must be able to direct that a file be sent to STAR and either continue processing or go into RECALL until a file is received from STAR and then continue processing.

The STAR-100 System is a general purpose system which is capable of high performance on scalar as well as vector operations. If a significant functional feature such as dynamic analysis is to be converted to STAR, many of the basic tools, features and extensions to incorporate the rest of NASTRAN should be developed. These features and extensions include:

- Virtual space map maintenance
- Virtual file address assignment
- Minus page maintenance
- I/O connector management
- Exception processing
- Random access record management
- Space allocation/deallocation
- Symbolic debug capability
- Program card changes

The nominal expected performance of the presently envisioned Control Data 6600-STAR link (which does not include double buffering) is as follows:

80 msec	CIO read of one page
50 msec	Five data link transfer per page
<u>50 msec</u>	<u>Service Station write of one page</u>
180 msec	Transfer of one page of data from Control Data 6000 to STAR

This is equivalent to a transfer rate of six pages/second.

Now suppose it is desired to solve a 20,000 row problem with structural matrix characteristics as follows: real, symmetric, and banded of semi-bandwidth of 1000.

This results in a matrix file of approximately 20 million 60-bit floating point numbers or 37,000 small pages. At a transfer rate of six pages/second this requires an elapsed time of 6,150 seconds or 103 minutes to cross the link.

A problem has been encountered in attempting to transfer the 20 million word file to STAR; namely, the transfer is to the STAR service station which is presently planned to be configured at the Langley Research Center with two 844 disk drives. STAR operating system does not support files extending across pack boundaries. So we are limited to a file containing at most 23,000 pages which is considerably less than the 37,000 pages required for the supposed 20 million word file. In order to solve this problem, provisions must be made for dividing the large file into more manageable segments.

Before this file is usable by an equation solver on STAR, two more processes must take place on the file:

It must be transferred (via central memory) to the STAR storage station.

The 60-bit floating point numbers in Control Data 6000 format must be converted into the 64-bit STAR equivalent.

If large data files are to be transferred, specialized techniques using double buffering, large buffers, automatic substructuring, and floating point conversions should be developed.

Therefore, the remaining conversion effort, to get all of NASTRAN on STAR-100, while involving significant volumes of code, would not require further system type extensions.

#### Interactive NASTRAN

Interactive NASTRAN means a NASTRAN system which gives the structural analyst the ability to direct the problem solution while at the interactive console. This could be implemented by making commands analogous to DMAP and rigid formats available to the analyst at the console. Also it would be highly desirable to interact with graphics terminals in order to quickly and conveniently analyze graphical output. Graphics interactives would require a new set of commands to be added to NASTRAN.

This approach will almost certainly be taken in any major new structural analysis program. Whether or not this is a reasonable goal for the NASTRAN conversion effort is unclear. The usefulness of interaction of the user with DMAP sequences without corresponding interactive graphic capability is doubtful. The interactive graphics capability represents a significant hardware/software cost outlay which is not in present system plans. Therefore, this true interactive mode for NASTRAN execution is currently unrealistic.

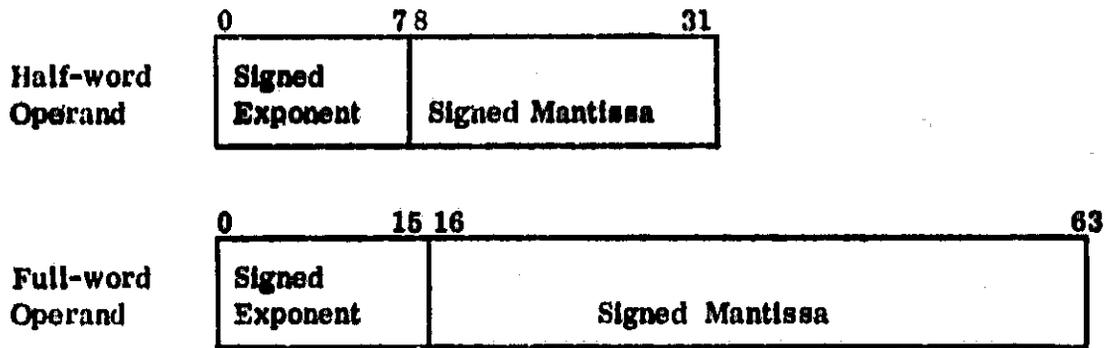
For development and maintenance work on NASTRAN, the interactive mode of operation will be used almost exclusively for controlling task execution. That is to say a user may gain flexibility in initializing NASTRAN execution by using the interactive mode of operation. It does not mean that a user without knowledge of the internal workings of NASTRAN could usefully and successfully interact with NASTRAN once it is in execution.

#### **Extended Features for Efficient Processing**

The STAR-100 has an extensive repertoire of instructions and options to instructions which give the software designer ample opportunity to enhance performance. In particular, some areas which have great potential are: 32-bit word operand size, vector macro instructions, and string operations.

#### **Half-Word Operands**

For every floating point arithmetic operation on STAR, whether scalar, vector, sparse vector, or vector macro, there is an option to use half-word (32-bit) operands or full-word (64-bit) operands. Furthermore there are instructions which convert half-word operands to full-word operands and other instructions which convert full-word operands to half-word operands. Figure 8 shows that a half-word carries 23 significant bits while a full-word carries 47 significant bits.



**Figure 8. Half- and Full-Word Operand Formats**

The stream rate for half-word vector operands is twice that of full-word vector operands. More importantly, but not as easily quantified, is the savings in core memory and auxiliary storage and the increased transfer rate between them when dealing with half-words instead of full-words.

Thus, for performance it becomes extremely important to analyze the amount of significance required at various states of computation. Most raw engineering data can be accurately represented in 23 bits. It is recognized that 23 bits of significance is not universally acceptable in stiffness matrix generation. However for many problems it is acceptable.

Error monitoring is relatively inexpensive and very accurate in the solution of special systems of linear equations. For real, symmetric, positive-definite matrices an attempt should be made to do both the decomposition and back substitute in half-word arithmetic. Automatic procedures could be invoked to resolve the system using full-word arithmetic when it becomes apparent that half-word accuracy is insufficient.

Again, on the output side, 23 bits of significance would normally suffice for graphical devices and reports.

This leads to the conclusion that the following important areas be investigated for possible optional use of half-word operands for representation of matrix coefficients:

- Matrix generation modules
- Report generation modules
- Data recording functions (checkpoint/restart)
- Dense matrix operations for small matrices
- Matrix operations on well-conditioned matrices

#### Vector Macro Instructions

STAR has an unusual set of powerful instructions which manipulate operands in a complex manner. Among these are the following types of operations:

- Dot products
- Sums
- Products
- Transmit lists
- Matrix transpose

If used with care, these operations have potential in simplifying and clarifying coding sequences. This has great usefulness in program maintenance. Presently it is unclear whether their use will improve performance. As the timing information becomes more stable and if some of the instruction micro-code sequences are changed, their utility may be more accurately assessed with respect to improved code performance.

#### String Operations

Included in these instructions are the generalized binary and decimal arithmetic operations, search operations and logical operations on bit strings. These operations have particular utility in data conversion, editing, and sorting operations.

#### Alternate Hardware Configuration

Throughout this report a drum paging station has been assumed to be present in the STAR-100 system.

It is not necessary to explicitly have a paging station in the configuration. A storage station with Control Data 844 or 819 disk drives may fulfill the paging functions completely. As was noted previously, this is always the case for large pages even with a paging drum station.

A storage station made up of 819 disks is referred to as a high capacity disk (HCD) station. The random paging rate which can be sustained on the HCD station is on the order of 100 pages/second.

For the processing of large sequential virtual files, efficiency is improved by having a storage station take on the paging station function. This is due to the elimination of the extra overhead paths of getting pages of the virtual file in and out of the core/drum system.

#### Limitations on Size of Problem

The question of how large a problem can be solved with NASTRAN on STAR-100 is a formidable one.

There are many dimensions to this question which include:

- Theoretical vs. practical limits
- Maximum file size
- Mean-time between failures and recovery procedures
- Available secondary storage space

In principle, the limitations may be extended almost indefinitely.

STAR operating system allocates a 16-bit field for definition of mass storage file length in small pages. This limits a single operating system file to 33.5 million 64-bit words.

STAR operating system does not allow files to reside on more than one disk pack. This restriction further limits the maximum file size to 12.5 million 64-bit words on an 844 disk. Similarly this limits a file size to 30 million 64-bit words on an 819 disk.

Treating an adaption of the existing real-symmetric matrix decomposition within NASTRAN as representative of limiting conditions allows one to examine another dimension of the problem. To be technically operable, there need only be enough real memory data area available to contain two rows of terms of length B where B is the semi-band width of the banded portion of the real-symmetric matrix and B terms for each of the currently active columns; i.e.,

$$(n + 2) B \leq \text{real memory data area}$$

where n is the maximum number of active columns. As this inequality approaches equality, the paging rate is high. For low paging rates and the corresponding high performance, the following inequality should be limiting:

$$(n + \frac{B}{2}) \leq \text{real memory data area}$$

As an example, assume the real memory data area available is 400,000 words. Further assume that n is approximately equal to B. To remain in the low paging region, the semi-band width B should be less than 632.

#### MAINTENANCE CONSIDERATIONS

The sheer size of NASTRAN presents a problem above and beyond the correctness of theory and engineering technology. The logistics of physical and mechanical control over one quarter of a million lines of source code in over 1300 source decks, object decks, and an executable file are formidable. Carefully integrated methods have evolved from experience with current NASTRAN operation that define the format of deliverable files and specify utility programs and procedures for handling them. This question of how to "handle" NASTRAN on STAR will be analyzed during the conversion, maintenance, and exchange stages.

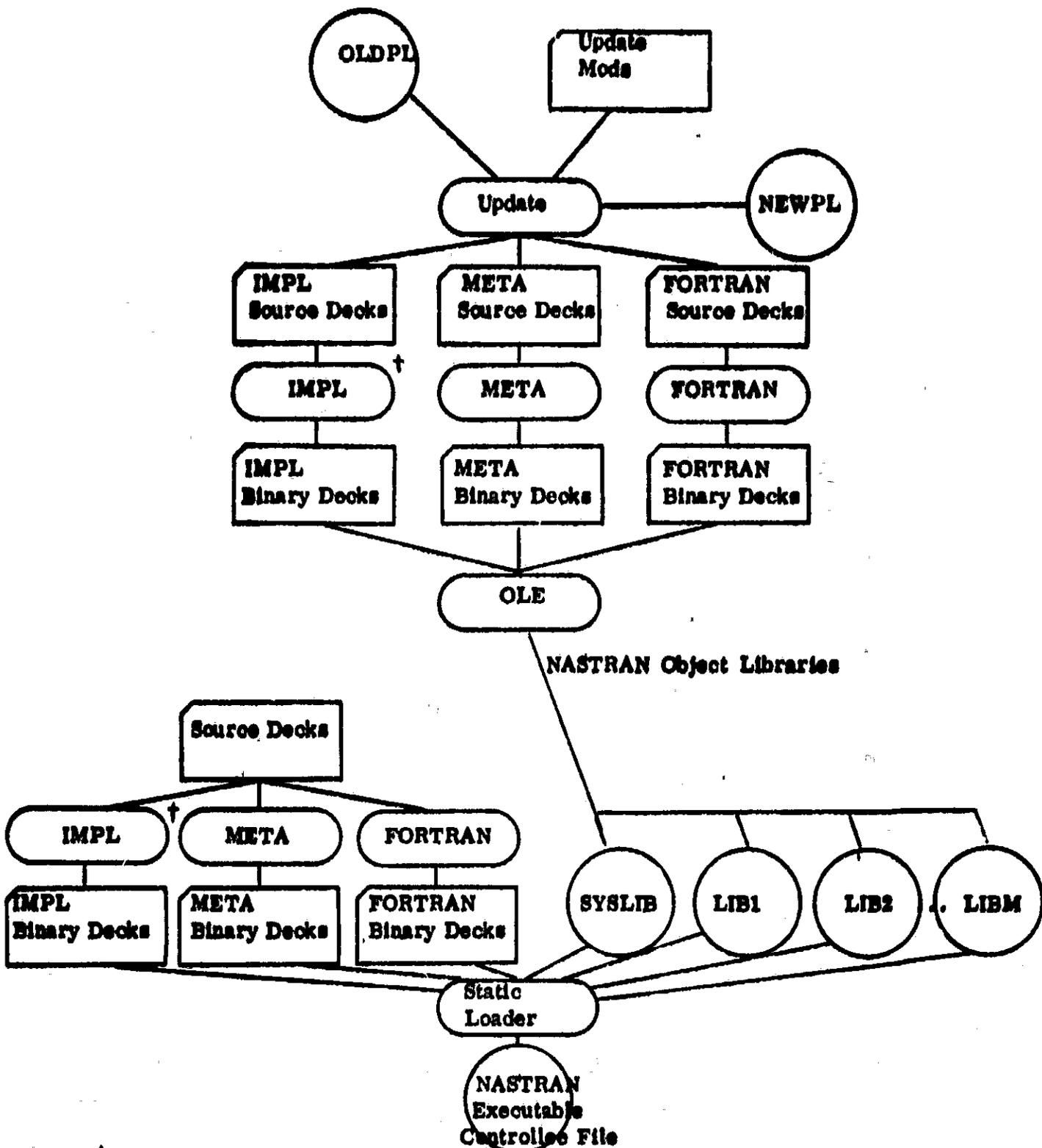
## Conversion Handling

The initial conversion task will be to get the NASTRAN machine-independent source code onto STAR. A small modification to the NASTRAN utility source conversion program will convert Control Data 6000 source to IBM 360 source with an ampersand (&) instead of a dollar sign (\$) in nonstandard return calling sequences. This converted source can then be placed on a STAR update program library in a format compatible with STAR FORTRAN input. Changes to this library would be hand-coded and applied using the UPDATE utility program. No significant modifications to this file are envisioned except for renaming of open-core common blocks and data block/file association and disassociation.

Machine-dependent source code must be written from scratch and also maintained on an update library. Aside from the difficult task of generating technically correct and error free code, there should be no major "handling" problem with the machine-dependent code.

The decision to execute NASTRAN as one controllee file eliminates many handling problems associated with Control Data 6000 versions. There is no necessity to maintain or execute the linkage editor or the XBOOT routines on the front of the absolute file. There is no more LINKLIB or SUBSYS decks to maintain. The STAR loader will handle input from several libraries and an object library editor (OLE) can perform the functions of the NASTRAN utility MERGEX.

Thus during conversion of NASTRAN to run on the Control Data STAR computer, most of the "handling" problems will be in the source code area with the object library editing and loading being accomplished with standard STAR software. The general flow for this procedure is depicted in Figure 9.



† It is not anticipated that IMPL will be used in NASTRAN subsystem but the possibility exists none the less.

Figure 9. General Flow of NASTRAN Subsystem Build

## Maintenance Handling

Maintenance involves debugging of error conditions and checking out corrective code on STAR itself with problems of compatibility and swapping of fixes between different NASTRAN machines left to the next section on transportability. In this phase, indeed in all aspects of operation except actual execution, the interactive mode of STAR operation will be a boon to the NASTRAN systems programmer. Each step in code check procedure can be initiated and completed as a separate task, and debugging will no longer be the time consuming procedure it now is with overnight batch turnaround.

The usual maintenance tasks would generally be handled much as they are on current levels of NASTRAN. Changes can be made to the source code by way of update alters, the changed decks recompiled and reloaded to produce a checkout version of the system against which test problems can be run to verify the correctness of the code fixes. Update, object library editor, and loader are all standard STAR software that can be employed in these procedures.

The NASTRAN utility program COMPARE is FORTRAN coded and easily converted to STAR. It would be of general use to compare and verify the implementation of alters or to discover what has changed when one is presented with a clean new version of an old deck.

Thus routine maintenance of the STAR version of NASTRAN on the STAR machine presents no significantly new or additional problems in handling procedures, and indeed may be easier with the use of interactive procedures and the lack of link edit/SUBSYS tasks.

## Inter-Machine Handling

The problem of intermachine compatibility hinges on the degree of sameness in the machine-independent code. If the code can be made to look exactly alike, it would be convenient to number the source statements alike and have a universal update program to update the source code from the same alters on all four

NASTRAN machines (IBM System/360, UNIVAC 1108, Control Data-6000, Control Data STAR-100). Such an ideal situation does not exist, but the current situation of slight differences in the machine-independent source code is manageable.

In this case, if one sticks to the initial conversion with only slight modifications to existing machine-independent code, then one can apply many of the same techniques and procedures used to handle inter-machine maintenance that are in existence on current third generation NASTRAN machines. For short SPR fixes involving few changes in code, alters from any of the NASTRAN machines can be hand punched and visually verified into a format for updating on any other NASTRAN machine. For more extensive changes, both new decks and the alters that produced them are desirable. Then with the help of the compare utility, the user can decide whether to install alters into his decks or reinstall his alters in the new decks. Of course there is no simple, mechanistic approach to the machine-dependent routines since the source code varies so radically on each of the NASTRAN machines. A bug found and fixed on one machine needs to be tested and traced on the others. Any major improvements or additions to machine-dependent code involves additional re-writing of code on all machines.

Progressing beyond the initial conversion to where future versions of NASTRAN might have a redesigned and radically different machine-independent source code to take special advantage of STAR architecture, one arrives at a contradiction in terms. If the code is different, then it isn't machine-independent. What one has is two versions of NASTRAN with more machine-dependent code than independent code. One has really returned to the pre-NASTRAN mode of maintaining separate versions of a program on separate systems. Eventually a decision must be made as to whether compatibility between third and fourth generation systems is as important as exploiting fourth generation technology.

## **Operational Reliability**

**A large conversion effort to a new and radically different environment is bound to reduce the reliability of NASTRAN for some time in relation to the reliability now enjoyed on the Control Data 6000.**

**On the STAR-100 System, with its speed and an interactive debugging capability, the turn around time for bug fixes should be very much less than that experienced on Control Data 6000 series with comparable level of personnel competence.**

**An estimate of the mean-time-between-failure for the STAR-100 System is unavailable.**

## **MANPOWER ESTIMATES**

**The estimates and suggestions contained herein are in no way to be interpreted as commitments on the part of Control Data Corporation to produce within these guidelines or even to develop them at all. They are simply frank assessments of what might be done and at what cost as seen within the limited confines of this study.**

**The effort to convert NASTRAN to STAR-100 may be grouped according to the following categories:**

- Operating system modifications**
- Compiler modifications and extensions**
- Pseudo-operating system features**
- Machine-independent code conversion**
- Machine-dependent code conversion**
- NASTRAN optimization**
- Maintenance**

## **Operating System Modifications**

Operating system modifications should be made by the standard operating system group to ensure continual support and maintenance of these modifications.

### **Batch Processor**

The batch processor should be extended to recognize files which exist for the duration of a job (i.e., across several tasks). The names of the files are unique only to that job, and the files are automatically purged upon job termination unless overt action is taken to prevent purging. This greatly simplifies batch control card sequences and prevents unintended file proliferation.

### **Loader**

The loader must be extended such that it is possible and relatively convenient to group by various arrangements, more than 1000 routines and common blocks. This constitutes no basic new features for the loader.

### **Debugging Aids**

Symbol table output from FORTRAN is provided for the purpose of symbolic debugging. The debug features of STAR operating system should give the user the capability of symbolic debugging.

A memory dump utility which allows the user to conveniently dump the controllee and/or drop files is a basic requirement lacking from the presently defined debugging aids.

### **Compiler Modifications and Extensions**

Modifications and extensions to the STAR FORTRAN compiler per se should be made by the standard compiler group to ensure continual support and maintenance. STAR FORTRAN library routines could come from a variety of sources.

### Non-Standard Returns Label

Either the compiler could be modified to accept "\$" as well as "&" to signify a non-standard return label or the NASTRAN source code could be preprocessed to accommodate this minor discrepancy. It is recommended here that the existing NASTRAN utility source conversion program be changed to modify the machine-independent code rather than modifying the compiler.

### FORTRAN IV Library Routines

These routines may be divided into three classes:

Those which have an exact equivalent in STAR FORTRAN

Those which are unused in NASTRAN

Those without an exact equivalent

Those routines which have an exact equivalent with exception of routine name should have the name changed in the outboard preprocessor or cataloged in the library with new names. Those unused in NASTRAN may be ignored in the conversion effort. Those without an exact equivalent in STAR FORTRAN must be provided.

### Program Card

The STAR FORTRAN compiler should be modified to provide some file device flexibility without necessitating recompilation, i.e., the user should have device destination as a run-time option to decide which files are to physically exist on tape and which files are to be assigned to disk. The user presently does not have this option.

### Half-Word FORTRAN

As a convenience in exploiting the 32-bit arithmetic capability of STAR, it is desirable to allow the definition of 32-bit operands to be analogous to 64-bit operand usage presently used in STAR FORTRAN. There have been two different proposals made for this implementation, namely:

- Integrate a half-word TYPE into the present compiler.
- Build another compiler analogous to the present one, but using 32-bit as the basic operand.

### **Pseudo-Operating System Features**

These three features, virtual space manager, indexed sequential record management, and physical space allocation/deallocation are classified as pseudo-operating system features, because with careful design they would have universal utility for large general purpose programs.

#### **Virtual Space Manager (VSM)**

The virtual space manager is responsible for the following six functions:

- Non-I/O NASTRAN/OS interface
- Virtual space map maintenance
- Virtual file address assignment to logical disk space
- Minus page maintenance
- I/O connector management
- Exception processing

#### **Indexed Sequential Record Management**

A subset of indexed sequential record management is recommended to simplify the conversion effort and yet run efficiently. This subset is:

- Indexed sequential (integer keys, actual keys) for virtual files
- Indexed sequential (integer keys, actual keys) for physical files

#### **Physical Space Allocation/Deallocation**

Upon entry to a module, it is necessary (as is presently done in NASTRAN via DMAP at least) to declare which files are needed and whether they are input, scratch, or output files. This information must now be augmented with directions for determining a reasonable size for all scratch and output files.

Thus, each of the approximately 85 modules must have a specially tailored prologue to supply reasonable data block sizes.

### **Machine-Independent Code Conversion**

The conversion of machine-independent code involves editing over 1200 NASTRAN source decks written in a subset of FORTRAN IV. This includes:

- Modification to Source Conversion Program to convert from Control Data source code to IBM source code with ampersand (&) instead of dollar sign (\$).
- Processing of Control Data NASTRAN source files to produce pseudo-IBM source file.
- Processing of pseudo-IBM source file to produce tape file physically written and blocked for STAR compatibility.
- Creation of an UPDATE tape of STAR machine-independent NASTRAN source codes.

### **Machine-Dependent Code Conversion**

Several routines associated with linking and loading have been eliminated. New routines associated with pseudo-operating system modules have already been mentioned. The remaining current machine-dependent routines require a straightforward translation of the language or the function.

### **NASTRAN Optimization**

Optimization opportunities exist throughout most of NASTRAN, however, the areas of greatest potential are:

Record management

Matrix file structure and the primitive matrix operations

Selective use of half-word operands.

#### **Record Management**

The indexed sequential record management scheme presently used in NASTRAN, but extended to include virtual and physical files, is believed to be effective for STAR-100. An optimization effort would involve choosing a file

type (physical or virtual), buffer sizes, and NASTRAN file distribution over STAR operating system files. If there is a significant usage of random files on the order of 500,000 to one million words in size, it would be advisable to investigate the implementation of hashing techniques for random access such as the direct access method.

#### Matrix File Structure

With careful attention to design, the matrix file structure can be changed to a dual-file concept with only minimal impact outside of the primitive matrix operations themselves. Assuming an adaptation of existing algorithms without significant new mathematical development, the routines which should be completely recoded are as follows:

BLDPK	GFBS	PARTN
CDCOMP	GMMATS	PLAMAT
DECOMP	INTPK	SADD
DMPY	INVERS	SDCOMP
ELIM	MERGE	SOLVER
FACTOR	MPYAD	TRANSP
FBS	PACK	UNPACK
		UPART

#### Half-Word Operands

Within the present software structure there are two options for implementation of half-word operand processing, namely through STAR FORTRAN special calls or through assembly language subprograms. Although much more inconvenient than coding in FORTRAN IV, the cost of implementation this way may be less than the cost of explicitly adding this feature to FORTRAN. However, it should be possible to amortize the FORTRAN extensions over several projects. Specific areas which, without doubt, could be implemented using half-word operands have not been identified.

## Maintenance

Maintenance tasks can be split between the conversion phase and the ongoing maintenance phase as follows:

- Preparation of initial libraries and development of procedures for use during conversion
- Procedures for development of new and corrective code on STAR and its distribution to other NASTRAN machines
- Integration of new and corrective code developed on other NASTRAN machines.

## Cost Breakdown

Table 3 illustrates the effort involved and alternatives possible in developing NASTRAN for STAR-100 as described in this study. This presentation is based on features rather than specific routines. Estimates of the effort involved are displayed. It is unrealistic to present a routine by routine breakdown at this time. This could come only after considerable detailed design effort is expended.

## Basic Conversion

Figure 10 is a graphic representation of a reasonable work allotment for the basic NASTRAN conversion effort. This corresponds to the items labeled "Vital" in Table 3 and refers to a scaler version of NASTRAN on STAR-100. This figure shows that the conversion could be done in an elapsed time of nine months with a peak manpower loading of eight people.

## Matrix File Structure and Vector Processing

There are four basic families of matrix operations which are effected by matrix file structure and which may be vectorized in varying degrees:

- **Matrix composition**
- **Matrix Decomposition**
- **Matrix Multiply - add and transpose**
- **String utilities**

**These operations should be valid for either single-precision real or complex values of the coefficients.**

**The graphs in Figure 11 show a reasonable manpower loading and elapsed time for vectorizing these time-consuming operations in NASTRAN. The volume of code involved in vectorizing the important operations appears moderate. The difficulty involved in interfacing with other more casual users of matrix files is unexplored and accounts for the large manpower difference in the graphs. It should be noted that system checkout of vectorizing code can not begin until the basic conversion is complete.**

TABLE 3. NASTRAN - STAR-100 ADAPTATION

Feature	Category	Reason	Man Months
Batch Processor	OS Mod	Convenience	6 <sup>†</sup>
Loader	OS Mod	Vital	1 <sup>†</sup>
Symbolic Debug	OS Mod	Convenience	12 <sup>†</sup>
Virtual Memory Dumps	OS Mod	Convenience	1 <sup>†</sup>
Nonstandard Returns Label	Compiler Mod/ Ind Code Conv	Convenience	1 <sup>†</sup>
FORTRAN IV Library	Compiler Mod/ Ind Code Conv	Vital	6
Program Card	Compiler Mod	Convenience	3 <sup>†</sup>
Half-Word FORTRAN (1)	Compiler Ext/ NASTRAN Opt.	Convenience	100 <sup>†</sup>
Half-Word FORTRAN (2)	Compiler Ext/ NASTRAN Opt.	Convenience	64 <sup>†</sup>
Non-I/O NASTRAN/OS	Pseudo OS	Convenience	2 <sup>††</sup>
Virtual Space Map Maintenance	Pseudo OS	Vital	3 <sup>††</sup>
Virtual File Address Assignment	Pseudo OS	Vital	4 <sup>††</sup>
Minus Page Maintenance	Pseudo OS	Vital	4 <sup>††</sup>
I/O Connector Management	Pseudo OS	Vital	6 <sup>††</sup>
Exception Processing	Pseudo OS	Vital	12 <sup>††</sup>
Index-Sequential Virtual	Pseudo OS	Vital	8 <sup>††</sup>
Index-Sequential Physical	Pseudo OS	Performance	4 <sup>†††</sup> 10 <sup>††</sup>

Feature	Category	Reason	Man Months
Allocation/Deallocation	Pseudo OS	Vital	10 <sup>††</sup>
Machine-Independent Code Conversion	Ind. Code Conv.	Vital	8
Machine-Dependent Code Conversion	Dep. Code Conv.	Vital	4
Record Management	Pseudo OS/NASTRAN Opt.	Performance	Needs Study
Matrix File Structure	NASTRAN Opt.	Performance	30 - 60
Half-Word Operands	NASTRAN Opt.	Performance	Needs Study
Library Preparation and Procedures	Maintenance	Vital	1

Total of Vital Tasks

67 Man Months

† These features should be done by STAR-100 Development Division of Control Data Corporation.

†† Pseudo OS features could be generalized with careful planning - estimates here do not include that case.

††† This assumes that Index-Sequential-Virtual is done. At least one of these must be done.

Months 0 1 2 3 4 5 6 7 8 9

Manpower

2



**FORTRAN IV library and exception processing.**

2



**Virtual space manager (without exception processing and non-I/O NASTRAN/STAR interface).**

2



**Allocate/deallocate and independent code conversion.**

1



**Indexed sequential virtual file manager.**

1

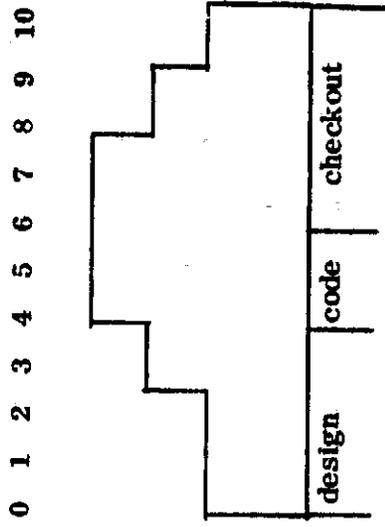


**Loader, machine-dependent code conversion, Library preparation**

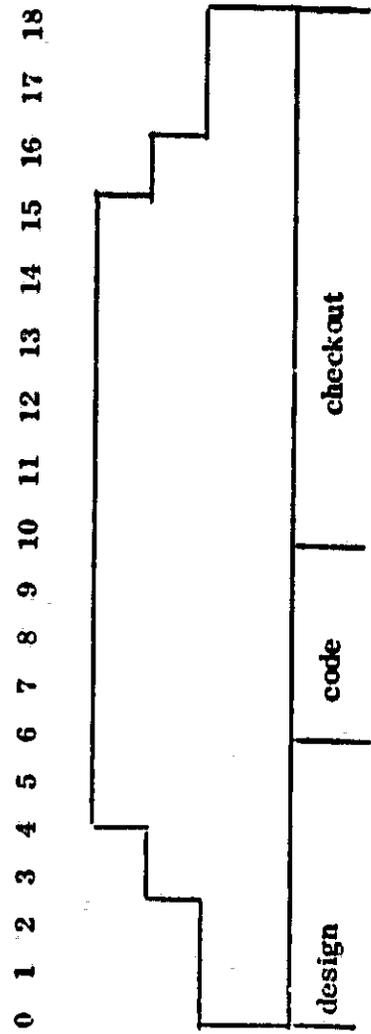
**NOTE: This does not include external documentation or extensive quality assurance testing.**

**Figure 10. Time Frame Necessary for Basic NASTRAN Conversion (Scalar NASTRAN)**

**Optimistic Estimate**



**Pessimistic Estimate**



**Figure 11. Vector Processing**

## CONCLUSIONS

This study concludes that NASTRAN can be converted to run on the Control Data STAR computer. The findings are that:

- NASTRAN machine-independent source statements are readily convertible
- FORTRAN compiler and library are compatible
- NASTRAN can execute as one controllee file with module groupings
- File allocation and input/output activity must operate under control of a virtual space manager
- The basic conversion<sup>†</sup> effort is estimated to be six man-years

---

<sup>†</sup> The basic conversion effort produces only scalar code for NASTRAN.

## GLOSSARY OF STAR OS TERMS

**Controllee file** - The file which contains NASTRAN executable code and initialized data areas. This file is never modified during execution.

**Drop file** - The file which contains all modified controllee file pages and all used pages which were not assigned by the user to a virtual file.

**Explicit I/O** - Input/output which occurs under the direction of the user through the use of virtual memory buffers which are "locked-down" in real memory.

**GRLP** - A STAR loader directive which allows the user to group routines and/or common areas in a user defined arrangement on a large page boundary.

**GROL** - A STAR loader directive which allows the user to group common areas in a user defined arrangement on a large page boundary. No minus page map entries are generated. This is the responsibility of the user at execution time.

**GROS** - A STAR loader directive which allows the user to group common areas in a user defined arrangement on a small page boundary. No minus page map entries are generated. This is the responsibility of the user at execution time.

**GRSP** - A STAR loader directive which allows the user to group routine and/or common areas in a user defined arrangement on a small page boundary.

**IMPL** - The implementation language in which the central operating system is written.

**I/O Connector** - An input/output connector is a four-word block used by the operating system to establish a link between the program and its I/O device. The operating system uses an I/O connector to keep track of a specific file's activity and a program's use of that file. Each time a program issues a create or open file request, an I/O connector is created.

**LINK** - Collection of one or more segments which comprise a logical subdivision of the program.

**LINKLIB** - Collection of subprograms which are used to satisfy unresolved externals in NASTRAN.

**META** - The assembly language used in coding STAR-100 central memory programs.

**Minus Page** - All virtual files have a minus page preface; it consists of 512 word segment (small page containing information to control program execution and data access).

**OLE** - Object library editor. The STAR loader uses library files as a source of modules to complete unsatisfied externals. OLE is used to create and modify such libraries.

**Pool Files** - Files which are accessible to more than one user, but which are not universally accessible.

**RECALL** - The state of a program when it has released control of the central processor until a fixed time has elapsed or until a requested function is complete.

**Sparse vector** - A vector with a high incidence of zero elements.

**SUBSYS** - Procedure for generation of functional links in NASTRAN.

**Uni-programming** - The execution of only one program at a time.

**Virtual I/O** - Input/output which occurs (perhaps unknown to the user) when virtual memory referenced exceeds the available real memory or when virtual memory referenced has not been associated with real memory.

**XBOOT** - A bootstrap program which writes subsequent programs as an indexed file on disk and then passes control to an entry point which initiates execution of the problem program.

## APPENDIX A

### SCALAR CPU TIMING COMPARISON BETWEEN 6600 AND STAR-100

The following information was extracted from the Control Data Corporation Standard Benchmark Library. Several points are noteworthy:

- 6600 computer system timings are actual
- STAR-100 timings are computed
- STAR FORTRAN code is unoptimized
- Only scalar code was generated by STAR FORTRAN
- Virtual addressing was in effect on STAR-100

Table A.1 presents the results of a comparison of ten problems. Brief descriptions of these problems follow the table.

Table A.1. 6600/STAR-100 SCALAR CPU COMPARISON

Problem	6600 CP Seconds	STAR-100 CP Seconds	Ratio
1	11.507	6.050	1.9020
2	13.297	12.600	1.0553
3	10.058	5.353	1.8790
4	7.465	4.000	1.8663
5	7.065	3.600	1.9625
6	6.721	4.740	1.4179
7	6.309	5.020	1.2568
8	4.726	3.200	1.4769
9	12.531	4.250	2.9485
10	9.841	3.300	2.9821

Average Ratio	1.87470
Standard Deviation of Ratio	0.65047
Maximum Ratio	2.9821
Minimum Ratio	1.0553

- |                   |  |
|-------------------|--|
| 1) Name           | ADVAN  |
| Source Statements | 69   |
| Language          | FORTRAN  |
| Purpose           | CPU Performance  |
| Description       | Dynamics code which tests 3-dimensional subscripts. This kernel <sup>†</sup> presents significant opportunity for subscript evaluation optimization. This kernel is the solution of three simultaneous partial differential equations for a plane rectangular object.  |
| 2) Name           | BIG1LP   |
| Source Statements | 18   |
| Language          | FORTRAN  |
| Purpose           | CPU Performance  |
| Description       | This kernel forms a dot product of two vectors with a dimension of 200. It tests the efficiency of small loops.  |
| 3) Name           | GIBSON   |
| Source Statements | 32   |
| Language          | FORTRAN  |
| Purposes          | CPU Performance  |
| Description       | Written to produce code on the Control Data 3600 to approximate the Gibson Mix - an instruction mix (i.e., values are calculated by multiplying the execution times for selected instructions based on their (supposed) frequency of use). These products are summed to give the "GIBSON VALUE" for that machine. The mix is for CPU only; |

---

<sup>†</sup>The word kernel throughout this appendix refers to the fact that the test was completely in core and no I/O was involved.

no I/O is considered. However, timing comparisons from actual problems on many computer systems show the relative speeds as indicated by the Gibson Mix are also valid for throughput.

- |                   |   |
|-------------------|---|
| 4) Name           | LINK  |
| Source Statements | 29  |
| Language          | FORTRAN   |
| Purpose           | CPU Performance   |
| Description       | This kernel extracts elements from a data array via a function call and sums those elements. It tests the efficiency of function linkage. |
| 5) Name           | SCI-1   |
| Source Statements | 7   |
| Language          | FORTRAN   |
| Purpose           | CPU Performance   |
| Description       | This is a subroutine which generates a Hilbert Matrix. Element (I,J) of the matrix is set to $1/(I+J-1)$ .                                |
| 6) Name           | SCI-2   |
| Source Statements | 19  |
| Language          | FORTRAN   |
| Purpose           | CPU Performance   |
| Description       | This is a subroutine which inverts the Hilbert Matrix generated by SCI-1.   |

- 7) **Name** SCI-3  
**Source Statements** 9  
**Language** FORTRAN  
**Purpose** CPU Performance  
**Description** This kernel forms the product of the Hilbert Matrix generated by SCI-1 and the inverse of the Matrix generated by SCI-2.
- 8) **Name** SINCOS  
**Source Statements** 40  
**Language** FORTRAN  
**Purposes** CPU Performance  
**Description** This kernel converts a set of coordinates from polar to Cartesian form. It evaluates the performance of SIN and COS functions.
- 9) **Name** SORTEST  
**Source Statements** 53  
**Language** FORTRAN  
**Purpose** CPU Performance  
**Description** This kernel creates 10000 random integers and sorts them in core using a shell sort. It tests random branching.
- 10) **Name** TBLLNK  
**Source Statements** 52  
**Language** FORTRAN  
**Purpose** CPU Performance  
**Description** This kernel builds a series of chained table entries. It tests random branching.

## APPENDIX B

### STIFFNESS MATRIX DECOMPOSITION TIMING

The information contained in this appendix was extracted from a report by the Advanced Software Research Department of Control Data Corporation, December 1974.

Structural analysis consists of several steps as shown in Figure B-1. The most complex (in terms of arithmetic and data management) is the decomposition of the stiffness matrix. This phase consumes nearly all of the computing resources (memory, CPU time, I/O time) compared with the other steps as the problem size increases. This is also illustrated in Figure B-1.

#### Algorithms

The stiffness matrix requiring decomposition is almost always a symmetric positive definite matrix (i. e.  $Q^T A Q > 0$ ). The algorithms used are Gauss elimination (in symmetric form) or one form of Cholesky decomposition (with or without square roots) as shown in Figure B-2. The original ordering of nodes in a structure is renumbered to reduce the matrix bandwidth. The "effective bandwidth" (designated by the mnemonic NBAND) depends upon the programming (numerical) algorithms used in implementing the mathematical algorithm. The width of each row is the number of elements from the first non-zero position up to (but not including) the diagonal. A classical constant band algorithm requires the effective bandwidth to be equal to the largest row width. A variable band algorithm (often called profile method) sets the effective bandwidth equal to the average row width. The effective bandwidth is the key to the computational assets required. We have broken the analysis into two categories:

1. Variable Band Algorithms - The effective bandwidth has been set to  $1.5 \sqrt{N}$
2. Constant Band Algorithm - The effective bandwidth has been set to  $4 \sqrt{N}$

Where  $N$  is the number of equations to be solved. There are in-between cases such as the active column approach of NASTRAN. Also for the two algorithms, there is considerable deviation from the averages of  $1.5 \sqrt{N}$  and  $4 \sqrt{N}$ .

The specific programming procedures used are shown in Figure B-3.

### CPU Timing Analysis

The section explains the details involved in the calculation of CPU time requirements for the Control Data 6600 and STAR. A formula is derived for each computer and formula verification data via benchmarks is illustrated.

The Control Data 6600 CPU timing formulas are shown in Figure B-4. The 6600 CPU time, unlike STAR, is contributed to by swapping and mapping factors reflecting CPU time used for data management functions (since the entire problem will not fit into 6600 central memory). The breakdown of the 6600 CPU time by function is shown in a tabular form in Figure B-5. Figure B-6 shows the validation with IMP<sup>†</sup> benchmarks.

The STAR timing formulas are shown in Figure B-7. The formulas are derived using a simulation program for the STAR-100. The base data for individual instruction timings come from the STAR Advanced Development Laboratory Report of April, 1974. "STAR-100 Timing Document" (through Revision 3). These timings have been verified on STAR except for the micro-code type of complex instructions using bit patterns, etc.; none of these instructions were used in the benchmark codes. The simulator accounts for memory conflicts, pipe conflicts, register conflicts, branches to various superword positions in and out of stack, etc., in the timing analysis.

---

<sup>†</sup> IMP is a software system for the direct or iterative solution of large differential and/or algebraic systems. Copyright D. M. Brandon, Jr., March 1972.

Figure B-8 shows STAR and 6600 CPU times for the variable banded algorithm based on full machine utilization.<sup>†</sup> The analysis for both constant and variable banded algorithms was done for full utilization of both machines (uni-programming) and partial<sup>†</sup> utilization of both machines. Both Gauss and Cholesky algorithms were evaluated for STAR and both FORTRAN and COMPASS code were evaluated for the 6600. Figure B-9 shows the STAR and the 6600 CPU time for the variable banded algorithm based on partial machine utilization. Figure B-10 shows STAR and the 6600 CPU time for the constant band algorithm based on full machine utilization. Figure B-11 shows STAR and 6600 CPU time for the constant band algorithm based on partial machine utilization.

---

<sup>†</sup> Full or partial utilization is defined as a real core memory data area of the following sizes:

STAR-100	Full utilization	920,000	64-bit words
STAR-100	Partial utilization	200,000	64-bit words
CDC 6600	Full utilization	60,000	60-bit words
CDC 6600	Partial utilization	12,000	60-bit words

**STEPS (STATIC, LINEAR-ELASTIC)**

- Data input
- Geometry check
- Cartesian coordinate transformation
- Global coordinate transformation
- Stiffness matrix generation
- Matrix decomposition
- Forward substitution
- Back substitution
- Stress recovery
- Data output

**TIME REQUIRED (LARGE PROBLEM - FULL CDC 6600 MEMORY USED)**

	<u>CPU TIME</u>	<u>I/O TIME<sup>†</sup></u>
Decomposition	7,639	32,477
Total <sup>††</sup>	10,090	32,654

<sup>†</sup> Not actual I/O time, but SCOPE 3.3 accounting algorithm.  
<sup>††</sup> Includes all steps above.

Figure B-1. Structural Finite Element Analysis

## ORIGINAL PROBLEM

$$\bar{\bar{A}}x = \bar{B}$$

where:

$\bar{\bar{A}}$  = Stiffness matrix

$\bar{B}$  = Load vector

$\bar{X}$  = Displacement vector

## DECOMPOSITION

$$\bar{\bar{A}} = \bar{\bar{L}}\bar{D}\bar{\bar{U}}^T$$

## FORWARD AND BACK SUBSTITUTION

$$\bar{\bar{L}}\bar{Z} = \bar{B}$$

$$\bar{\bar{U}}\bar{X} = \bar{D}^{-1}\bar{Z}$$

where:

$$\bar{\bar{U}} = \bar{\bar{L}}^T$$

## CHOLESKY DECOMPOSITION

- Dot products used to find  $\bar{\bar{L}}$

## GAUSS DECOMPOSITION

- Row operations used to find  $\bar{\bar{U}}$

Figure B-2 Algorithms

## BOTH MACHINES

- Storage of original and decomposed stiffness matrix by rows only in cumulative lists
- Variable banded (profile) method used for timing
- Separate storage of diagonal vector D (actually  $D^{-1}$ )
- Full "shortcuts" code employed:
  - Singleton diagonal elements
  - One element in row prior to diagonal
  - Dot products and row operations performed with shortest length on row combinations
- Cumulative indexing of  $\bar{A}$  and  $\bar{U}$  (or  $\bar{L}$ ) with indices for:
  - First non-zero position in row
  - Diagonal position in cumulative list
- Full error monitor code
- No data packing (full word usage)

## STAR

- Virtual memory used.

## Control Data 6600

- ABDM (Arithmetic Block Data Manager of IMP) used

Figure B-3. Programming Procedures Used

$$T_{\text{CPU}}^{6600} = T_{\text{ALG}}^{6600} + T_{\text{MAP}}^{6600} + T_{\text{SWAP}}^{6600}$$

where:

$T_{\text{ALG}}^{6600}$  Time due to arithmetic or algorithm (If the algorithm were written to operate entirely within core memory, this would be the only factor in  $T_{\text{CPU}}^{6600}$ ).

$T_{\text{MAP}}^{6600}$  Time due to mapping (finding page location for data on disk and in core buffers).

$T_{\text{SWAP}}^{6600}$  CPU time due to execution of swap routines and FORTRAN, 6RM and CIO calls.

$$T_{\text{ALG}}^{6600} = a_4 \cdot N + a_5 \cdot N \cdot \text{NBAND} + (a_6 / 2) \cdot N \cdot \text{NBAND}^2$$

$$a_4 = 4.0 \cdot 10^{-5}$$

$$a_5 = 2.7 \cdot 10^{-5}$$

$$a_6 = 3.0 \cdot 10^{-6}$$

$$T_{\text{MAP}}^{6600} = a_7 \cdot N + a_8 \cdot N \cdot \text{NBAND}$$

FORTRAN

COMPASS

$$a_7 = 4.0 \cdot 10^{-3}$$

$$2_7 = 4.0 \cdot 10^{-3}$$

$$a_8 = 5.0 \cdot 10^{-4}$$

$$a_8 = 2.0 \cdot 10^{-4}$$

$$T_{\text{SWAP}}^{6600} = a_9 \cdot \text{NSWAP}$$

$$a_9 = 2.1 \cdot 10^{-2}$$

where:

$N \equiv$  Number of equations

$\text{NBAND} \equiv$  Effective semi-bandwidth

$\text{NSWAP} \equiv$  Number of page swaps

Figure B-4. Control Data 6600 CPU Timing<sup>†</sup>

<sup>†</sup> Based on variable banded algorithm code (profile method) with full error recognition (60-bit arithmetic).

**VARIABLE BAND ALGORITHM**  
**(NBAND = 1.5  $\sqrt{N}$ )**

Number of Equations	Time (Seconds)							
	Algorithm <sup>1</sup>		Mapping		Swapping <sup>2</sup>		Total	
	FORTRAN	COMPASS	FORTRAN	COMPASS	FORTRAN	COMPASS	FORTRAN	COMPASS
100	.07825	.07825	1.1500	0.7000	0	0	1.2282	.77825
250	.36362	.36362	3.8750	2.1500	.02100	.02100	4.7596	2.5346
500	1.2822	1.2822	10.250	5.3000	.06300	.06300	11.595	6.6452
750	2.7514	2.7514	18.375	9.1500	0.1260	0.1260	21.252	12.027
1000	4.6225	4.6225	27.500	13.400	0.2100	0.2100	32.332	18.232
3000	37.020	37.020	135.00	61.200	1.1340	1.1340	173.15	99.354
5000	98.780	98.780	285.00	126.00	2.4570	2.4570	386.24	227.74
10000	378.40	378.40	790.00	340.00	111.28	111.28	1279.7	829.68
20000	1463.6	1463.6	2200.0	928.00	1057.6	1057.6	4721.2	3449.2

- <sup>1</sup> The FORTRAN code was written so as to compile into the optimal machine code.
- <sup>2</sup> The swapping CPU time is the same for FORTRAN and COMPASS since the same COMPASS I/O macros and GRM/CIO calls are used (however the swapping I/O time differs).

Figure B-5. CPU Time - CDC 6600 Full Machine

### FULL MEMORY UTILIZATION

EXAMPLE	N	N <sub>0</sub> BAND <sup>2</sup>	ACTUAL		FORMULA	
			T <sup>6600</sup> <sub>CPU</sub>	NSWAPS	T <sup>6600</sup> <sub>CPU</sub>	NSWAPS
1	50	6.25•10 <sup>4</sup>	1.49	3	1.2	0
2	100	2.5•10 <sup>5</sup>	3.55	3	3.4	1
3	250	1.56•10 <sup>6</sup>	14.0	3	13.8	4
4	500	6.25•10 <sup>6</sup>	42.9	20	40.7	12
5	750	1.41•10 <sup>7</sup>	90.1	93	78.9	63

### PARTIAL MEMORY UTILIZATION

EXAMPLE	N	N <sub>0</sub> NBAND <sup>2</sup>	ACTUAL		FORMULA	
			T <sup>6600</sup> <sub>CPU</sub>	NSWAPS	T <sup>6600</sup> <sub>CPU</sub>	NSWAPS
1	50	6.25•10 <sup>4</sup>	.752	3	1.3	2
2	100	2.5•10 <sup>5</sup>	2.26	6	3.5	6
3	250	1.56•10 <sup>6</sup>	31.2	989	35.9	1053
4	500	6.25•10 <sup>6</sup>	152.9	4493	127.5	4144
5	750	1.41•10 <sup>7</sup>	281.2	9457	273.2	9316

Figure B-6. Formula Validation Comparisons with CDC 6600 Benchmarks<sup>†</sup>

<sup>†</sup> FORTRAN code of IMP Version 2.0.

$$T_{\text{CPU}}^{\text{STAR}} = a_1 \cdot N + a_2 \cdot N \cdot \text{NBAND} + (a_3 / 2) \cdot N \cdot \text{NBAND}^2$$

**VECTOR MODE**

**GAUSS<sup>†</sup>**

$$a_1 = 1.89 \cdot 10^{-5}$$

$$a_2 = 2.01 \cdot 10^{-5}$$

$$a_3 = 6.00 \cdot 10^{-8}$$

**CHOLESKY<sup>†</sup>**

$$a_1 = 1.89 \cdot 10^{-5}$$

$$a_2 = 1.50 \cdot 10^{-5}$$

$$a_3 = 1.60 \cdot 10^{-7}$$

where:

N = Number of equations

NBAND = Effective semi-bandwidth

**Figure B-7. STAR-100 CPU Timing<sup>††</sup>**

† Crossover point between Gauss and Cholesky is at an effective bandwidth of about 100.

†† Based on variable banded algorithm code (profile method) with full error recognition code (64-bit arithmetic).

**VARIABLE BAND ALGORITHM**  
**(NBAND = 1.5  $\sqrt{N}$ )**

Number of Equations	Time (seconds)			
	STAR (FORTRAN)		CDC 6600 (CHOLESKY)	
	GAUSS	CHOLESKY	FORTRAN <sup>†</sup>	COMPASS
100	0.032715	0.02619	1.2282	.77825
250	0.12427	0.10155	4.2596	2.5346
500	0.35743	0.30051	11.595	6.6452
750	0.67007	0.57628	21.252	12.027
1000	1.0299	0.90062	32.332	18.232
3000	5.6065	5.3605	173.15	99.354
5000	12.433	12.539	386.24	227.74
10000	37.089	40.689	1279.7	829.68
20000	112.57	135.89	4721.2	3419.2

† The difference in these times is due to mapping, not the arithmetic in the inner loop.

Figure B-8. Full Machine Utilization-CPU Time of STAR-100 vs. CDC 6600  
 (Variable Band Algorithm)

**VARIABLE BAND ALGORITHM**  
**(NBAND = 1.5  $\sqrt{N}$ )**

Number of Equations	Time (seconds)			
	STAR (FORTRAN)		CDC 6600 (CHOLESKY)	
	GAUSS	CHOLESKY	FORTRAN	COMPASS
100	0.032715	0.02619	1.2702	0.82025
250	0.12427	0.10155	4.3856	2.6606
500	0.35743	0.30051	11.994	7.0442
750	0.67007	0.57678	21.987	12.762
1000	1.0299	0.90067	33.424	19.324
3000	5.6065	5.3605	457.87	384.07
5000	12.433	12.539	1177.7	1187.4
10000	37.089	40.689	4339.4	3889.4
20000	112.57	135.89	16307.	15035.

Figure B-9. Partial Machine Utilization-CPU Time of  
 STAR-100 vs. CDC 6600 (Variable Band Algorithm)

**CONSTANT BAND ALGORITHM**  
**(NBAND =  $4\sqrt{N}$ )**

Number of Equations	Time (seconds)			
	STAR (FORTRAN)		CDC 6600 (CHOLESKY)	
	GAUSS	CHOLESKY	FORTRAN	COMPASS
100	.08709	.07469	2.7502	1.5520
250	.35107	.32035	10.862	6.1366
500	1.3634	1.3345	31.601	18.251
750	2.6060	2.6346	59.856	35.331
1000	4.0498	4.2010	94.844	57.044
3000	24.392	28.236	743.38	546.28
5000	55.013	67.703	2035.2	1612.2
10000	170.49	230.09	6839.7	5639.7
20000	537.25	798.85	23914.	20524.

Figure B-10. Full Machine Utilization-CPU Time of STAR-100 vs. CDC 6600 (Constant Band Algorithm)

**CONSTANT BAND ALGORITHM  
(NBAND =  $4\sqrt{N}$ )**

Number of Equations	Time (seconds)			
	STAR (FORTRAN)		CDC 6600 (CHOLESKY)	
	GAUSS	CHOLESKY	FORTRAN	COMPASS
100	.08709	.07469	2.8570	1.6570
250	.35107	.32035	14.558	9.8326
500	1.3034	1.3345	87.401	74.111
750	2.6060	2.6346	185.37	160.85
1000	4.4098	4.2101	318.28	260.48
3000	24.392	28.236	2597.7	2400.5
5000	55.013	67.703	6946.1	6523.1
10000	170.49	230.09	27004.	25804.
20000	6734.0	6995.6	105150.	101760.

Figure B-11. Partial Machine Utilization-CPU Time of STAR-100 vs. CDC 6600 (Constant Band Algorithm)

## BIBLIOGRAPHY

Coffman, E. G.; and Denning, P. J.: *Operating Systems Theory*.  
Prentice - Hall, 1973.

Control Data Corporation: *Control Data STAR Computer System FORTRAN  
Language Reference Manual*. Pub. No. 60386200, June 1974.

Control Data Corporation: *Control Data STAR Computer System Operating  
System Reference Manual*. Pub. No. 60384400B, January 1974.

Control Data Corporation: *Control Data STAR-100 Computer System Preliminary  
Instruction Execution Timing Manual*, Pub. No. 60440600, July 1974.

Control Data Corporation: *Math Science Library*. Pub. No. 60327500A, Vol. 6,  
March 1971.

Control Data Corporation: *STAR-100 Computer System Hardware Reference  
Manual*. Pub. No. 60256000, May 1973.

Denning, P. J.: *The Working Set Model for Program Behavior*. *Comm.*  
*ACM* 11, May 1968, pp. 323-333.

Douglas, F. J., ed.: *The NASTRAN Programmer's Manual*. NASA SP- 223,  
1974.

Field, E. I.; Johnson, S. E.; and Stralberg, H.: *Structural Mechanics Computer  
Programs*. University Press of Virginia, 1974, pp. 1019-1042.

Geurtin, R. L.: *Programming in a Paging Environment*. *Datamation*,  
February 1972, pp. 48-55.

Heuer, G. A.: *Matrix Algebra and Related Problems*. Control Data  
Technical Report TR 53.

Hohn, W. C.; and Jones, P. D.: *The Control Data STAR-100 Paging Station*,  
*National Computer Conference*, 1973, pp. 421-426.

IBM Corporation: *IBM 7090/7094 IBSYS Operating System Version 13  
FORTRAN IV Languages*. File No. 7090-25 GS28-639C-4, November 1968.

Lambiotte, J. J., Jr.; and Bolgt, R. G. : The Solution of Tridiagonal Linear Systems on the CDC STAR-100 Computer. Institute for Computer Applications in Science and Engineering (ICASE), July 1974.

MacNeal, R. H., ed.: The NASTRAN Theoretical Manual. NASA SP-221(01), 1972.

McCormick, C. W.; and Redner, K.H.: Study of the Modifications Needed for Effective Operation of NASTRAN on IBM Virtual Storage Computers. The MacNeal-Schwendler Corporation, Los Angeles, October 1974.

McCormick, C. W., ed.: The NASTRAN User's Manual. NASA SP-222(01), 1972.

McKellar, A. C.; and Coffman, E.G.: Organizing Matrices and Matrix Operations for Paged Memory Systems. Comm. ACM 12, March 1969, pp. 153-165.

Noor, A. K.; and Fulton, R.E.: Impact of the CDC-STAR-100 Computer on Finite-Element Systems. Presented at Sixth National ASCE Conference on Electronic Computations, August 1974.

Schnock, Paul B.: The Myth of Multiprogramming. Software - Practice and Experience, Vol. 4, 1974, pp. 59-62.