N92-24325

NASTRAN INTERNAL IMPROVEMENTS FOR 92 RELEASE

by

Gordon C. Chan
NASTRAN Maintenance Group
UNISYS Corporation
Huntsville, Alabama

SUMMARY

The 1992 NASTRAN release incorporates a number of improvements transparent to users. The NASTRAN executable has been made smaller by 70 percent for the RISC base Unix machines by linking NASTRAN into a single program, freeing some 33 megabytes of system disc space that can be used by NASTRAN for solving larger problems. Some basic matrix operations, such as forward-backward substitution (FBS), multiply-add (MPYAD), matrix transpose, and fast eigensolution extraction routine (FEER), have been made more efficient by including new methods, new logic, new I/O techniques, and, in some cases, new subroutines. Some of the improvements provide ground work ready for system vectorization. These are finite element basic operations, and are used repeatedly in a finite element program such as NASTRAN. Any improvements on these basic operations can be translated into substantial cost and cpu time savings. This paper also discusses NASTRAN in various computer platforms.

NASTRAN SINGLE LINK

The 91 NASTRAN RSIC base Unix version was released as a single link program. (The multi-link version can also be built.) The shell program that controls NASTRAN's execution was modified so that it can run NASTRAN under single- or multi-link environments automatically. The success of the single link Unix version (previously referred to as NASTRAN superlink) prompts the conversion of the 92 VAX/VMS NASTRAN into a single link program. The VAX single link program is extremely useful in debugging the Unix RISC version since the latter has extremely poor system diagnostics and provides no error trace-back. The biggest advantage of the single link version, and particularly in a small workstation environment, is that it needs only 25 percent of the disc space to hold the executable program as compared to the multi-link version, and 33 megabytes of disc space is returned to the system that can be used by NASTRAN for solving larger structures. It is also faster by almost 70 percent to build the single link NASTRAN as compared to the complete multi-link version. In program execution, the single link has no overhead for link-switching, and data saving and recovering between links are not needed. The single link versions, both in RISC base machines and VAX/VMS, appear to be running faster

since the program could be executing several "links" ahead of the screen printout if NASTRAN is executed interactively. For purposes of consistency, the single link program issues the link numbers as if the program were generated in multiple links. Presently, there are no single link versions for IBM, CDC, and UNIVAC machines.

NASTRAN IN VARIOUS MACHINE PLATFORMS

COSMIC/NASTRAN is supported in five computer platforms: IBM, CDC, UNIVAC, VAX/VMS, and DEC/ULTRIX (a RISC base Unix machine). However, the COSMIC/NASTRAN is continuously improved aiming towards a unified environment. ANSII standard FORTRAN 77 is used; and most of the machine dependent items are removed if possible. Some source codes are modified or re-written with system vectorization in mind. A far-reaching plan is initialized in the boot-strap (BTSTRP) subroutine that is used to set up machine-dependent constants for the five machines that COSMIC supports. The 92 COSMIC/NASTRAN has expanded the machine-dependent constant table in BTSTRP for 18 major computers and two dummies. A user, or a third-party organization, can install NASTRAN to a new platform currently not supported by COSMIC. Also, the users or third parties using the new computer platform can talk to one another since the machine type has been pre-arranged. The machine-constant table in BTSTRP is arranged for the following computers:

> DUMMY, IBM, UNIVAC, CDC, VAX, DEC/ULTRIX (RISC base Unix), SUN, IBM/AIX, HP, SILIC.GRAPHICS, MAC, CRAY, CONVEX, NEC, FUJITSU, DATA GENERAL, AMDAHL, PRIME, 486, DUMMY

BTSTRP sets up the machine constants correctly only for the first six machines. The first DUMMY is set up for IBM 7094, which has long been obsolete, and therefore can be reused for any machine not on the list. The last DUMMY is intended for the same purpose. The constants for the remaining 14 machines are dummies or guess-values. Therefore before moving COSMIC/NASTRAN to any computer platform, one must first re-supply the correct machine dependent constants in BTSTRP for that machine. The machine constants are well defined in the BTSTRP subroutine, such as NBPW, number of bits per word; NBPC, number of bits per character; NLPP, number of lines per printout page, etc.

There are also a few machine-dependent constants outside BTSTRP that need to be set locally. These few constants are scattered in about eight or nine NASTRAN subroutines. For example, some of these constants involve convergent criteria used only locally, and therefore not at the same level of importance as those in BTSTRP. To locate these few machine-dependent constants has been made easy in the 92 NASTRAN release. If the first word of labeled common /MACHIN/ in a subroutine is MACHX, not just MACH, there are machine dependent constants that require fixing.

The main NASTRAN program in the single link environment is called NASTRN. In the

multi-link environment, each of the 15 links is a complete program by itself, and the main programs are NAST01, NAST02, NAST03,..., NAST15. There is something very special in NASTRN and in NAST01 in the 92 NASTRAN release. If the DEBUG flag in NASTRN or NAST01 is changed to 1, the so-called <LINK 1> portion of the single link NASTRAN, or the regular LINK 1 in the multi-link program, will go through a series of machine compatibility checks. The results will echo back to the user. If some things, or some parameters, are set incorrectly, NASTRAN will stop. For example, if the FORTRAN OPEN statement of a new computer platform is in byte count for the record length, RECL, and the user sets the corresponding constant in BTSTRP to word count, an error diagnostic will appear.

Due to COSMIC policy, the NASTRAN Maintenance Group has never tried to move NASTRAN outside the five designated computer domains. However, it was demonstrated by a user in 1991 that the DEC/ULTRIX version required only two minor changes to move NASTRAN to a SiliconGraphics machine; and those two changes have been incorporated back into the 1992 COSMIC/NASTRAN release. The NASTRAN Maintenance Group in UNISYS welcomes any contribution from the users or third parties, concerning the migration of NASTRAN to different computer platforms.

INTERNAL IMPROVEMENTS

The PROFILER, a performance analyzer of the VAX/VMS machine, was used to identify the major time consuming elements in several typical NASTRAN runs. This was followed by a major effort to look into the logic, computing mechanism, methods of calculation, order of execution, paging, vectorization etc. of the time consuming areas, and to search for improvements. Most of the time consuming elements are basically standard matrix operations, and they are the essential elements of a finite element program. Their treatments are either "textbook standard", or "company proprietary". Generally speaking, the original NASTRAN developer did a very fine job in these areas. Further improvements are not easy, and cannot be treated lightly or as short projects. Indeed, several of the 1992 internal improvements were made in periods of several weeks, not days. Some improvements were made on top of previous improvements or newly written subroutines. Sometimes a simple line of improvement may require days of careful study, thorough understanding of the program algorithms, and the theoretical treatment of the subject. The final improvements in the 92 NASTRAN release are quite satisfactory. Several slow moving areas are speeded up 25 to 30 percent, and in some areas, three to five times faster. Appendix A tabulates some of the test results. Most of the new internal improvements of the 92 NASTRAN release can be removed by DIAG 41.

The following sections of internal improvements apply only to large matrix operations. Usually the matrices are many times larger than the available computer core memory can hold at one time. Many of the matrix operations must be done by parts, or in a number of passes.

IMPROVED FORWARD-BACKWARD SUBSTITUTION

The forward-backward substitution (FBS) is used for matrix inversion, load-solution, eigenvalue iteration, and many other applications that follow matrix decomposition. If the number of columns on the solution side of the equation is large, FBS can be very time consuming. It could easily take 10 to 20 times longer to go through FBS than to do the matrix decomposition.

In NASTRAN, the driver for FBS is the subroutine FBS. The actual FBS computation takes place in FBS1, FBS2, FBS3, or FBS4 for real single precision, real double precision, complex single precision, and complex double precision calculation respectively. If FBS requires more than seven passes, the new improvements will automatically kick in. The improvements are in four new subroutines, FBSI, FBSII, FBSIII, and FBSIV, similarly arranged as the subroutines FBS1/2/3/4. The new improvements include reduced I/O operations, large data blocks, and new row-and-column matrix multiplication. The new FBSI/II/III/IV are about 30 to 50 percent faster than the original FBS1/2/3/4, as tested in COSMIC's VAX/VMS machine.

THE FEER METHOD

Seventy to seventy-five percent of the cpu time used by the FEER method (the fast eigenvalue extraction method, with real tridiagonal reduction) is actually spent in the subroutine FNXTVC (double precision version) or FNXTV (single precision version). The main time consumer in FNXTVC/FNXTV is the forward-backward substitution operation. Unlike the FBS module, the open core in FEER method is not fully used, and particularly not in FNXTVC/ FNXTV subroutines. The improvements in FNXTVC/FNXTV include reduced I/O operations, full utilization of the core space, and new row-and-column matrix multiplication. The new improvements in FNXTVC/FNXTV alone produce impressive results – reducing the FEER method cpu timing by 30 to 200 percent, as tested on the VAX/VMS machine, and on a CRAY (tested on RPK/NASTRAN).

COSMIC/NASTRAN sometimes gives negative values for the rigid body eigenvalues. (They should be zeros.) Sometimes the negative values could be quite large (-1.E+5 range) particularly on the IBM machine. The explanation for this strange behavior, and the solution to the problem, may or may not match. On the solution side, since the rigid body frequencies are zeros, the 92 COSMIC/NASTRAN FEER method, by default, will set them to zeros. The second solution option is to reinforce certain key areas of computation in FNXTVC/FNXTV by quad-precision (real*16) for the 32-bit word machines, and by double-precision for the 60- or 64-bit word machines. This second option gives good results and moves the rigid body frequencies down to 1.E-6 to 1.E-12 range. However, it takes 2 to 3 times longer to compute.

To activate quad-precision calculation in a 32-bit word machine or double-precision in a 60- or 64-bit word machine in FEER method, one only needs to replace the BCD word "FEER" on the third field of the EIGR bulk data card by "FEER-Q". Replacing "FEER" by "FEER-X"

will prohibit the substitution of zeros for the rigid body frequencies.

Now to explain what happens to produce the negative eigenvalues. The criterion for orthogonality convergence EPSILON is 1.E-28 for the 32-bit word machines using double precision computation, and 1.E-24 for the 60- and 64-bit word machines using single precision computation. In FNXTVC/FNXTV, the accumulated sum of a mode shape is compared to EPSILON. The accumulated sum at the end of a do loop is the difference of two very close numbers. Therefore this very small numeric difference is a function of the number of digits a computer word can hold. Most 32-bit word machines with double precision calculation, and most 60- or 64-bit word machines with single precision, are limited to 14 to 16 digits per word, and therefore down to only 1.E-14 to 1.E-16 numeric accuracies. Here, the mantissa as well as the exponent of a data word is important! Since all five computers supported by COSMIC exhibit this accuracy problem, it becomes meaningless to do an orthogonality check by comparing the result to EPSILON, which is another 10 to 14 decades smaller. To home in to the size of EPSILON could end up producing big numeric errors. (Just a guess here.) Since the mantissa of an IBM machine word, in double precision, is smaller than a VAX, IBM seems to produce big negative rigid body eigenvalues more often than the VAX. Similarly the 60-and 64-bit machine using single precision computation could be worse than IBM.

In the 92 NASTRAN release, an attempt is made to avoid the above dilemma. The orthogonality convergence criterion is based on a ratio instead of the finite difference of two very close numbers. However, presently there is not enough test data to verify that is a good fix.

NEW LOGIC FOR MATRIX TRANSPOSE

Matrix transpose of a matrix which is too big to reside completely in the computer core memory is not an easy task. It can be done, but it may use up lots of cpu time. Lots of I/O may be involved here, and perhaps a high percentage of system paging if a virtual machine is used. The problem here is how to do the matrix transpose in seconds instead of minutes, or in minutes instead of hours. A good algorithm here is a treasure, and quite often, it becomes a "company proprietary" product.

The out-of-core matrix transpose in NASTRAN is handled by the subroutine TRNSP. The algorithm there is amazingly powerful. The only drawback is that it uses up to nine scratch files. The scratch files are supplied by the calling routines. The more scratch files passing over to TRNSP, the bigger matrix transpose TRNSP can do. There is no check in TRNSP of the actual scratch files requirement. Again, there are lots of I/O, data packing, and unpacking involved.

If only 1/50th of the matrix can be loaded into the computer core memory space at one time, TRNSP will complete the transpose task in 50 passes. A new matrix transpose subroutine TRNSPS has been written for 92 COSMIC/NASTRAN. If the passes exceed seven, TRNSP will switch over to TRNSPS automatically, if and only if DIAG 41 is not turned on by the user.

TRNSPS uses only one scratch file. The I/O department and the data packing and unpacking are greatly reduced. The new TRNSPS is two to four times faster than the original TRNSP.

MPYAD, MPY4T, AND MPYDRI

Matrix multiplication and addition are basically the most important and most widely used tools for a finite element program. If the matrices are small and can reside completely in core, multiply-add has no problem; and three simple do-loops will complete the job. Again, if the matrices are bigger than the computer core can hold, matrix multiplication and addition have to be done by parts, and there will be many passes, many I/O operations, and many row- and column-packings and unpackings. The problem can become I/O bound, and lots of cpu time will be spent on getting and saving the intermediate results. The situation is further complicated in NASTRAN in that the matrices can be of different types (single precision or double precision; real or complex), or of different forms (rectangular, square, diagonal, identity that may or may not exit, lower or upper triangular, row vector, or symmetric), or transpose and non-transpose matrices.

There are five multiply-add (MPYAD) methods in NASTRAN, two methods for the non-transpose case (MPY1NT and MPY2NT) and three for MPYAD with transpose (MPY1T, MPY2T and MPY3T). NASTRAN selects internally the best method to use based on the cpu time requirement of each method that fits best for a given matrices-and-core environment. The cpu time requirement is a function of size and shape (rows and columns), form and density of the matrices, and core space. The cpu time requirement is also a function of the relative sizes and shapes of the matrices. That is, matrix A may be very big and cannot reside in core, while matrix B is small, and matrix B may or may not be loaded entirely into the available core space. Or vice versa. All five MPYAD methods are written in assembly languages for four out of five COSMIC supported machines, except VAX.

Matrix transpose is supposedly very slow. The programmer manual recommends matrix transpose be done via MPYAD with transpose, and an identity matrix. This turns out not to be very efficient. Two test matrices A and B, 5166x5166 each, using the best of MPY1NT or MPY2NT, can be multiplied together in 470 cpu seconds (on the VAX machine), while A-transpose times B, using the best of MPY1T, MPY2T, and MPY3T requires 6680 cpu seconds. That is 12 times longer! If matrix A is transposed first by TRNSP routine (TRNSPS is not used), then followed by MPYAD without transposing, the total cpu time can be cut to half. The only problem here is that at least one extra scratch file is needed to save the transpose file, and in many instances, there is no extra scratch file available.

MPY1NT and MPY1T share much common logic, and operate quite similarly. The same holds true for the MPY2NT and MPY2T pair. MPY3T is a third method for the transpose case. The best method for the test matrices A and B above, with transpose, is MPY2T. One would think that since NASTRAN stores a matrix by column, and that a column of matrix A transpose

is a row of A, the row-and-column multiplication (matrix A in row and matrix B in column) should be very fast, very smooth, and very convenient. One could almost feel and touch the natural flow of the multiplication algorithm. But what one can feel or imagine, is not what a computer sees. The row-and-column multiplication produces only one element in the resulting matrix. There are more than 26 million (5166^2) double precision elements to go. Anyway, the fact is that MPY2T takes 12 times longer than MPY2NT.

Several methods, several logics, and several new algorithms were developed and tested to beat the clock set by MPY2T. The ultimate goal is to match the MPY2NT performance if possible. Finally, after many trials, a fourth method, MPY4T, was developed based on the scheme similar to MPY2T, except that matrix B (and matrix C, to be added if it is present), and the resulting matrix D are processed by column instead of by element. Of course, the logic in MPY4T becomes much more complicated and the open core must be rearranged. But MPY4T is three to five times faster than MPY2T. In the 92 NASTRAN release, MPY4T will be automatically substituted for MPY2T, unless DIAG 41 is turned on by the user. MPY4T is written in FORTRAN, and it is machine independent.

The original MPYAD does not take advantage of certain types of matrices. For example, the transpose of matrix A which is symmetric, need not go through the transpose route. (This is already checked in the 91 release.) A new subroutine, MPYDRI, is added to the 92 release to handle special cases involving diagonal matrix, row vector, and identity matrix.

The correct handling of these special matrices expedites the MPYAD process by manyfold.

NEXT IMPROVEMENTS UNDER CONSIDERATION

The internal improvements in the 1992 NASTRAN release tackle a few important, and often-used, basic finite element tools with satisfactory results. However, there are many more areas in NASTRAN that can be explored. There are still several areas involving FBS that have not been touched. The matrix decomposition process could be improved. All the complex computations involving complex FBS, complex decomposition, complex FEER method, and more, are targets for the next improvements. Nevertheless, the internal improvements in 1992 NASTRAN release represent the beginning of an extraordinary effort to bring NASTRAN up to par.

APPENDIX A

Demo problem D03012A was used in most of the following tests. The D03012A demo produced a double precision KGGX matrix of size (5166 x 5166) and a KAA (2380 x 2380), double precision. The trailer of the KGGX matrix in some cases had to changed from "symmetric" to "square" so that NASTRAN did not take the symmetric route processing the modules under investigation. Tests were done on a VAX/VMS machine, unless stated otherwise. In most cases, HICORE is 350,000 words.

FBS test: KAA

1991 Version	1992 First Version	1992 Second Version
5644 cpu seconds	3508 cpu seconds	3120 cpu seconds

FEER method

1991 VAX Version	1992 VAX Version		
	GINO Improvement	Open Core Not Used	Open Core Used
1043 cpu seconds	978 cpu seconds	907 cpu seconds	763 cpu seconds
1991 CRAY Version	1991 CRAY Version Plus Changes – Open Core Used		
45.7 cpu seconds	21.8 cpu seconds		

MPYAD, KGGX(transpose) * KGGX + KGGX

DIAG 41 On	With MYADT (Obsolete)	MPYAD With New TRNSPS	MPYAD With New MPY4T	If Symmetric Matrix Allowed
6681 cpu secs.	3871 cpu secs.	2114 cpu secs.	1358 cpu secs.	469 cpu secs.