N91-20509

Experiences in Porting NASTRAN® to Non-Traditional Platforms

Gregory L. Davis
Robert L. Norton
Jet Propulsion Laboratory

Summary

The 1990 UNIX version of NASTRAN was ported to two new platforms that are not supported by COSMIC: the Sun SPARC workstation and the Apple Macintosh using the A/UX version of UNIX. This paper summarizes the experiences of the authors in porting NASTRAN, and makes suggestions for users who might attempt similar ports.

Introduction

Historically, NASTRAN has been supported on only the largest, most capable mainframe computers. For many years the computers supported by COSMIC were the CDC, IBM, and UNIVAC mainframes. In the late 1970s various manufacturers introduced what became known as minicomputers. These computers offered capable performance at much lower cost than traditional mainframe computers. After the very successful DEC VAX minicomputer was introduced, NASTRAN was ported to it. Over the last ten years the widespread use of VAX minicomputers has extended the use of NASTRAN to many new sites, and VAX leases now amount to over half of all NASTRAN leases. The introduction of small office-environment VAXes has allowed consultants and departments to bring NASTRAN nearly to the engineer's desk.

As the cost of computer hardware has decreased, the workstation market has emerged. Workstations offer the performance of minicomputers at a cost and size that allows single-user computers. The market has seen a variety of proprietary operating systems grow and then falter; the dominant operating system for workstations is now clearly UNIX. For the user this trend has been very helpful, allowing the user to concentrate on the proper hardware solution without having to also select the operating system. One significant advantage for the hardware manufacturer is the ability to concentrate on developing high performance hardware without having to divert resources into operating system development.

As UNIX workstations have become pervasive, COSMIC has released a new version of NASTRAN designed to be portable enough to run on a variety of these workstations. The first release of this version was designed for the DEC ULTRIX operating system and retained many of the non-standard FORTRAN extensions that are used in the VAX version. Later releases have moved closer to standard FORTRAN. Experiences with porting NASTRAN to new UNIX workstations have allowed the removal of certain impediments.

The rapid development of hardware has not been the exclusive province of workstations. Since the early 1980s microcomputers or personal computers have also shown amazing growth in capability. While the early 8-bit microcomputers were almost useless for finite element analysis, some work could be done on the 16-bit microcomputers of the middle to late 1980s. With the introduction of high speed 32-bit microcomputers, the boundary between workstations and microcomputers has become blurred. The cost of workstations has dropped enough that low-end workstations are cheaper than high-end personal computers, while the performance of high-end personal computers approaches the performance of workstations.

Many people have wrestled with the definitions of workstations and personal computers. Rather than focus on hardware to establish the difference, it makes more sense to look at the differences from the user's point of view. One big appeal of the personal computer has always been the vast array of software available. Few engineers would want to do without the personal productivity software they now routinely use. The vast volume of personal computers along with the relatively small number of display devices allows the development of niche software to go with the high volume software (e.g. word processors, spreadsheets). Probably the strongest feature of workstations is the robustness of UNIX. While it is trivial to write a program to crash a personal computer, it is much more difficult to crash UNIX.

Naturally enough, most engineers don't want to choose only the personal productivity software of the personal computers or only the robustness of UNIX — they want both on their desktop at the same time. Thus hardware manufacturers are producing computers that run both UNIX and traditional personal computer operating systems. There are many computers using Intel architecture that run UNIX and MS-DOS programs. Apple has available A/UX (their version of UNIX), which also runs regular Macintosh software and can even run MS-DOS software in emulation mode. The workstation hardware manufacturers are countering with Reduced Instruction Set Computers (RISC) that also run MS-DOS in emulation. One manufacturer has even announced a laptop RISC machine that runs UNIX, MS-DOS, and Macintosh software.

One clear winner has emerged from the confusion of operating systems and computer architecture — the end user. We now have available an amazing, almost paralyzing set of options. For the NASTRAN community this revolution means that "NASTRAN for the masses" is at hand. We have $10,000 desktop computers that are at least as capable as the multi-million-dollar mainframes that were used at the dawn of NASTRAN twenty years ago. Manufacturers have recently announced portable UNIX computers that are fully capable of running NASTRAN. Now the individual engineer can not only have NASTRAN at the desk, but also can carry NASTRAN to the work!

General Porting Comments

The sheer size of NASTRAN is one of the biggest obstacles to porting. The 1990 VAX version has 84 machine-dependent subroutines (0.3 MBytes) and 1695 machine-independent subroutines (13.3 MBytes), for a total of 1779 subroutines (13.7 MBytes). This size has always created problems for NASTRAN, and it typically pushes the boundaries of the computer and operating system capabilities.

Although the VAX version has been all FORTRAN, a number of VAX extensions to FORTRAN have been used. UNISYS has been trying to eliminate as many extensions as possible, but a number of extensions to FORTRAN are still used. Following is a summary of the extensions used, along with some suggestions for porting:

1. Some non-standard variable types are used: `REAL*4`, `REAL*8`, `INTEGER*2`, `INTEGER*4`, and `LOGICAL*1`. These extensions are often supported, but if not they can be easily changed.

2. Hexadecimal constants are used, and the required form of the hexadecimal constants may vary from one compiler to another. The hexadecimal edit descriptor `Z` and the octal edit descriptor `O` are used in the `FORMAT` statement.

3. Some non-standard functions are used: `IAND`, `IOR`, `IEOR`, `ISHFT`, `JMOD`, and `NOT`. All of these except `JMOD` are used in bit manipulation.

4. In-line comments are used, with ! signifying the beginning of the comment.

5. Hollerith constants are used in `DATA` statements.

6. The alternate `RETURN` specifier is used with & to indicate the statement label. Change the & to * to meet the standard.

7. `READONLY` is used in a file `OPEN` statement in subroutine `DSXOPN`.

8. File names in `READ` and `WRITE` statements are stored in arrays (using Hollerith constants) rather than using `CHARACTER` variables. These references should be changed to use `CHARACTER` variables.

9. Variable names exceed the 6 characters permitted by the standard.

10. `DISP=` rather than `STATUS=` is used in several `CLOSE` statements.

11. `TYPE=` rather than `STATUS=` is used in an `OPEN` statement.

12. The `%LOC` function is used to return the location in memory where a variable is stored.

13. Lower case source code is used.

14. Subroutines `CPUTIM`, `TDATE`, and `WALTIM` are used to get the cpu time, date, and all clock time from the system. The calls from these subroutines to get the system level information will be different for each new port.

Most of the extensions can be worked around. The truly significant extensions are the use of the %LOC and the non-standard functions. All of the above extensions are located in the machine-dependent subroutines, identified by the .MDS extension on the VAX. All the machine-independent routines, identified by the .MIS extension, compiled on the Sun and the Macintosh with no changes at all.

Sun Porting Experiences

The 1990 UNIX release of NASTRAN was shipped to JPL from UNISYS on a TK50 tape, where it was read onto a VAX ULTRIX machine and copied over to a Sun 4/390 using FTP. The ensuing porting and debugging process fell into three main stages.

Stage 1 consisted of fixing initial, fairly obvious incompatibilities between the Sun and VAX FORTRAN compilers. The machine-dependent subroutines were initially screened for the incompatibilities listed above in General Porting Comments. After all subroutines were compiled, the 15 executable NASTRAN links were generated. Gordon Chan of UNISYS was frequently consulted at this stage of the process and he provided invaluable assistance.

Stage 2 consisted of modifying the ancillary UNIX shell scripts used to drive the executable NASTRAN links. The script problems originally became apparent in trying to run sample problem D01000A.NID, when the proper UNIX links could not be established. XQT and @XQT are well-written UNIX shell scripts to provide a friendly user interface for running the NASTRAN program; however, these had to be modified to properly represent the user directory structure and to properly establish the UNIX links between the rigid format and the alter files.

Stage 3 consisted of debugging the executable links. Problems in execution became immediately apparent when trying to run sample problem D011A.NID. The first problem was eventually traced to bit shifting operations in subroutine KHRFN1: see point 2 under Recommendations to Users for details. A second problem in execution was traced to subroutine INTPK in link 4. This was inadvertently repaired by relinking link 4 with INTPK included twice in the link statement. Link ordering does become crucial! This ad hoc fix was then applied to all NASTRAN links containing INTPK. These repairs finally permitted the successful execution of test problem D01011A.NID on the Sun computer.

Macintosh Porting Experiences

The first major challenge with the Macintosh version was getting the source code downloaded to the Macintosh from the VAX. The only connection was via a 9600-baud local area network. Kermit was used to automatically download all the subroutines, which took about 10 hours. The UNIX versions of the machine-dependent subroutines were obtained via FTP from the Sun computer.

The FORTRAN compiler supplied with A/UX does not have the extensions required to properly compile the machine-dependent subroutines, so a third-party FORTRAN compiler sold by NKR Research, Inc. of San Jose, California was selected. NKR proved to be very helpful during this project, providing useful advice and compiler updates on a timely basis.

The organization of the files on the Macintosh took a couple of tries to get right. A/UX allows the use not only of the usual UNIX editors, `vi` and `ed`, but also of Macintosh graphical user interface editors, such as `TextEditor` (supplied by Apple with A/UX), `QUED/M` (a commercial editor), or `Alpha` (a shareware editor). Unfortunately, since the Macintosh file system does not adequately handle directories with large numbers of files, the source files cannot be stored together in one directory. The UNIX file system does cope with large directories, but the Macintosh editors use the Macintosh file system to open the files. The source files were put into 26 directories corresponding to the first letter of the subroutine name. In this way the largest directory had only 253 files.

The next hurdle was using the UNIX `ar` utility to create the library of object files. The VAX and other UNIX systems put all the object files together in one library. This library is then used as input to the linker to form each of the 15 executable files. The ar utility supplied with A/UX could not load all the object files into the library. After about 1400 files, it produced an error message when additional files were to be added to the library. In addition to the error in creating the library, it took one hour to load the object files into the library. To avoid the library problem all the object files were copied to a single directory. Since no Macintosh programs would be used in this directory, the weakness of the Macintosh file system did not matter. To link the executable files, a list of all the subroutines used in a link was generated on the VAX and used as input to the A/UX linker.

Recommendations to UNISYS

As the current maintenance contractor to COSMIC, UNISYS has done a splendid job in producing the UNIX version of NASTRAN. UNISYS has spent several years reducing the number of non-standard extensions to FORTRAN used in the code and has ported NASTRAN to several UNIX platforms.

There is a fundamental tension between the desire to produce a truly generic version which can be ported to new UNIX platforms relatively easily and the desire to optimize the code for a particular platform. The various proprietary versions of NASTRAN will probably continue to be more efficient than the generic version on any given platform, and some users will always complain. However, it is in the best interests of COSMIC and UNISYS to place the emphasis on portability. As the hardware manufacturers continue their rapid performance improvements, it seems to make more sense to upgrade the hardware than to "tweak" the code for improved performance.

From our experience in these ports of NASTRAN, we have several suggestions for UNISYS:

- NASTRAN is, of course, a rather old code, and FORTRAN has seen many changes since the FORTRAN IV that was used in the beginning. FORTRAN 77 introduced features that could simplify the code and also help the reading and maintainability of the code. The FORTRAN 90 that is currently being reviewed will introduce even more radical changes. UNISYS should move toward the use of structured programing. While it is possible to carry this to extremes with overly deeply nested `IF` clauses, a gradual transition to the use of the `IF - THEN - END IF` rather than repeated `GO TO` statements would help readability. After FORTRAN 90 becomes approved and supported, constructs such as `DO WHILE` and `DO - END DO` would also be helpful. The 1990 NASTRAN release does not use `IF - THEN - END IF` anywhere.

- The bit handling features of the code should be modernized by using character variables. Character variables were not available in the FORTRAN compilers used when these routines were written and the available computer memory was meager, so non-standard bit handling techniques were used. Now that NASTRAN is routinely used on computers with several hundred to several thousand times as much memory as the 16k-word IBM 7094 and since the FORTRAN 77 compilers support character variables, it is time to eliminate the bit manipulation.

- Have a dedicated UNIX machine at UNISYS connected to the Internet, thereby greatly facilitating program development and user/vendor communications. Program fixes and enhancements could then be transmitted using FTP, and user/vendor messages could be transmitted through e-mail.

- Provide the UNIX NASTRAN source codes and related shell scripts on media other than the TK50 tape, which is VAX specific. Other common media on UNIX-based "mainframe" type machines are 1/4 inch tape cartridges and 8 mm cassette tapes. CD-ROMs would provide a wonderful distribution media, especially when the manuals become available in electronic form.

Recommendations to Users

Porting NASTRAN to other computer platforms is an ambitious undertaking. At the outset the authors counsel patience and perseverance — the very large amount of code will probably stretch the computer's and user's resources to the limit. The following general approach for porting the UNIX version of NASTRAN over to other platforms profits from our own experience and mistakes.

1. Copy the NASTRAN source code over to the host machine, renaming the files as appropriate for the host's FORTRAN compiler. We highly recommend maintaining the `MDS/MIS` distinction in the directory structure — most of the coding incompatibilities will be in the `.MDS` routines. It is also a good idea to make a write-protected copy of *all* subroutines in the `.MDS` directory to preserve the capability to recover from inadvertent or incorrect edits during the debugging process. Develop a bookkeeping system to keep track of the large number of subroutines.

2. Initially screen the .MDS subroutines for coding incompatibilities with the host FORTRAN compiler. Prime candidates for compiler-dependent problems are listed in the above General Porting Comments. Comment any changes made for future reference.

   Bit shifting operations using the subroutine `khrfn1` need to be examined. This may or may not be a problem depending upon the convention for ordering the position in a character variable. Specifically, the character position of the VAX word is numbered left to right; the corresponding Sun and Macintosh word is numbered right to left. The current code assumes the VAX convention. This problem may arise in the `.MIS` subroutines `XSEM01-15`, which are the main drivers for each executable link.

3. Compile the source code. If the compiler has an option to produce a symbol table for a debugger, enabling it will prove very handy later. The `.MIS` subroutines should compile uneventfully; the `.MDS` subroutines may still have additional bugs. Debug any new errors and comment any changes for future reference. *Successful compilation is no guarantee of successful linking or successful execution.*

4. Upon completion of (3), archive the object modules using the supplied shell scripts to form the main library. If building the library exhausts the usable memory, subdivide the libraries into smaller, more manageable units or place all the object modules in one subdirectory and do without a library.

5. Upon completion of (4), build the 15 executable links using the supplied makefiles. Libraries containing certain intrinsic functions, or those supporting the VMS extensions, may have to be explicitly included in the link statement. Any unresolved cross-references among the subroutines will appear as errors here. Debug any new errors and comment the changes for future reference. *Successful compilation and linking are no guarantee of successful execution.*

6. Upon completion of (5), begin running the sample problems using the supplied shell scripts. Sample problems `D01000A.NID` and `D01001A.NID` are tests of LINK1, a good, simple initial test. At this stage, bugs will be more difficult to run down. The system debugging utility could prove invaluable here; however, there is one caveat: NASTRAN is so large that it may overload the symbol tables used by the debugger, giving incorrect error diagnoses. You then must resort to using strategically placed `WRITE` statements to debug.

7. After an error in a particular link is located, the following is a convenient way to test the fix. Initially, it is not necessary to rebuild the library; instead, the subroutine containing the prospective fix can be inserted directly into the link statement. Generate the appropriate makefile for the link being debugged based on the `makelink1` model supplied. Insert the debugged subroutine after `$(BLKDAT)` and before `$(LIB)`. *Linking is order dependent.* Regenerate the new executable link from this makefile. The fix can be tested by either rerunning the NASTRAN program from the beginning (LINK1) or having

saved the FORTRAN I/O files at the successful termination of the previous link, rerunning only the repaired link. If all is well, the library can then be rebuilt and all the links regenerated from the updated library.

Iterate through steps (6) and (7) until all of the sample problems run properly.

Implications for COSMIC

As it becomes easier to port NASTRAN to a wide variety of platforms, COSMIC is forced to deal with several difficult issues. The first of these issues is the question of how many versions of NASTRAN COSMIC should officially support. The present four versions could be drastically multiplied if COSMIC were to provide an official version for each of the hardware manufacturers that desires a port. One proprietary version of NASTRAN supports 15 different manufacturers, and some manufacturers require more than one version. This would be an intolerable burden for COSMIC and UNISYS. COSMIC's position is that only the current four versions will be supported, leaving the users, hardware manufacturers, and third-party software companies responsible for porting NASTRAN to other platforms.

This leads to the second issue. Once these new ports of NASTRAN have been accomplished, how does COSMIC control their quality? No one wants to see a situation where any number of people can make available new ports of NASTRAN and sell them without having some provision for quality control. The suggestion of the NASTRAN Advisory Group has been for COSMIC and UNISYS to work on an expanded suite of demonstration and validation problems. Only after a company certified that their port successfully passes this expanded suite would the company be allowed to advertise their port of NASTRAN. This is probably the best solution for now, but the policy might have to adjust over time.

Conclusions

The development of powerful desktop computers, both workstations and personal computers, combined with the UNIX version of NASTRAN has turned the dream of desktop NASTRAN into a reality. Enterprising users can do the port themselves, and third-party software companies will undoubtedly provide NASTRAN on a wide variety of computers. This development is a tribute to the original designers of NASTRAN, who provided such a robust program structure. This could well be the beginning of a new era of NASTRAN use, with the potential to provide an even better product, arising from the synergies of interaction between COSMIC and the new, expanded user community.

Acknowledgment

This paper presents research results carried out in part at the Jet Propulsion Laboratory, California Institute of Technology, under contract with the National Aeronautics and Space Administration (NASA).