

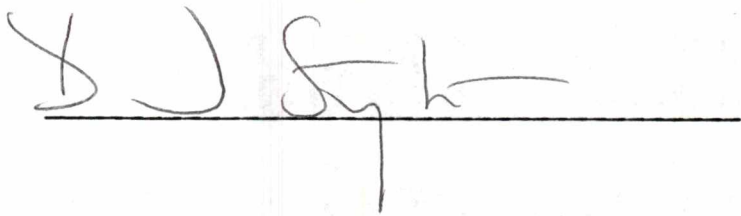
To the Graduate Council:

I am submitting herewith a thesis written by John B. Hampton III entitled "A FORTRAN-based MUMPS Interpreter." I have examined the final copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Computer Science.


Kevin C. O'Kane, Major Professor

We have read this thesis
and recommend its acceptance:





Accepted for the Council:


The Graduate School

A FORTRAN-BASED MUMPS INTERPRETER

A Thesis

Presented for the

Master of Science

Degree

The University of Tennessee, Knoxville

John B. Hampton III

August 1984

DEDICATION

For all her patience, love, and support throughout this long process, this thesis is for my wife Sue Bachmann. I couldn't have done this without her.

ACKNOWLEDGEMENTS

I first want to express deep appreciation to Dr. Kevin C. O'Kane of the University of Tennessee, Knoxville Computer Science Department for supplying me with this thesis project and providing me with good counsel and help along the way. Next I wish to thank Dr. Charles Pfleeger and Dr. David Straight for agreeing to be on my thesis committee, for being very helpful when the need arose, and for having been good educators while I was taking courses from them. The aid and support of the Computer Science Department was invaluable. I want especially to thank the departmental secretary, Ethel Wittenberg, whose unfailing help and good cheer have meant a great deal.

ABSTRACT

Computer systems--combinations of hardware and software--have evolved from the mid-1960's to the present time as the type and population of user communities have grown. Systems have changed in nature from requiring special technical expertise for their use to being more readily available to less-sophisticated audiences: i.e., being "user friendly." An early and successful attempt at a user-friendly system is the MUMPS system, consisting of a programming language, a data base manager, and an interactive operating system. Part of this study is an investigation of the MUMPS system historically, including present-day implementations, how the sytem has changed to meet new requirements of computer systems, and what ways MUMPS might yet evolve and improve.

An implementation of a MUMPS system written in the high-level RT-11 FORTRAN dialect of FORTRAN 66 was developed by Dr. Kevin O'Kane and others at the University of Tennessee, Knoxville in 1980. As part of the thesis project, this implementation was translated into a FORTRAN 77 dialect to run on Data General Corporation ECLIPSE series hardware. The translation process is given, and the two implementations are contrasted and compared. Suggestions for the further improvement of this FORTRAN-based MUMPS interpreter are given.

The result of this project is a working MUMPS interpreter running in a different hardware and operating system environment from its predecessor. Setting aside dialectical differences between FORTRAN 66 and FORTRAN 77, the translation was not especially difficult. This shows the validity of using high-level languages for systems development purposes, which is one of the criteria for the original FORTRAN-based MUMPS interpreter.

TABLE OF CONTENTS

CHAPTER		PAGE
1.	STATEMENT OF THE PROBLEM	1
2.	HISTORICAL OVERVIEW OF MUMPS	5
	History of MUMPS	5
	Comparison with BASIC.. . . .	10
	FORTRAN-based MUMPS Interpreter.	11
	Innovations and Improvements	15
3.	OPERATING SYSTEM INTERFACE	18
	AOS Overview	18
	AOS System Concepts.	19
	A Comparison Between AOS and RT-11	21
	Overview of FORTRAN 77	23
4.	A COMPARISON OF MUMPS INTERPRETERS	26
	DG MUMPS	26
	DSM-11	28
	Hitachi Micro MUMPS.	29
	Comparisons.	30
5.	DATA BASE CONSIDERATIONS	32

CHAPTER	PAGE
6. METHODOLOGY AND PROBLEMS WITH F77 MUMPS. . . .	35
Translation of LOGICAL*1 to CHARACTER	
Data Types	41
Conversion of Numeric Internal Representation	
to Character	43
Transportation of Sources From RT-11	
to AOS	45
Removal of RT-11-specific System Interfaces. .	46
Creation of FORTRAN Routines to Emulate	
RT-11 System Builtins.	47
7. SUGGESTED IMPROVEMENTS FOR FORTRAN MUMPS . . .	52
AOS-specific Enhancements.	52
General Enhancements	56
8. CONCLUSIONS.	63
LIST OF REFERENCES	65
APPENDICES	68
APPENDIX A. SOFTWARE TOOLS	69
APPENDIX B. DEVELOPMENT ENVIRONMENTS	72
APPENDIX C. BUILDING MUMPS	73
APPENDIX D. MUMPS BENCHMARKS	76
VITA	78

CHAPTER 1

STATEMENT OF THE PROBLEM

The main purpose of any computer system--hardware and software together--is utilization. The more work of all sorts, including program development and maintenance, that a system can allow, the better that system is. The MUMPS system, consisting of an interpreted language, an interactive operating system, and a data manager, is an early and good attempt to increase the capacity for work of a given set of hardware.

Early MUMPS systems were implemented in low level languages, which limited their portability from machine to machine as well as their understandability, which can cause difficulties in maintenance and enhancement of the systems. This project is the study of the translation of a MUMPS system written in a high level language (FORTRAN) to another FORTRAN dialect and to run under a different operating system. This high level language implementation of a MUMPS interpreter eliminates the above-mentioned shortcomings of any program written in a low level language: the program is easily portable to different machine architectures and operating systems, and the program can be understood quickly and easily, leading to modification and maintenance with a minimum of effort.

The following is an overview of topics covered in later chapters.

1. Histories of the MUMPS system in general and of this FORTRAN-based MUMPS interpreter in specific are given. Dr. Kevin O'Kane, of the Computer Science Department at the University of Tennessee, Knoxville, and others developed an interpreter for the MUMPS language in 1980. This interpreter was written in RT-11 FORTRAN, a dialect of FORTRAN 66. The development and runtime environment was a DEC PDP-11/03 with an LSI-11/02 processor running the RT-11 operating system. The resulting MUMPS interpreter is a stand-alone program for one or several users running under a single-job operating system. This describes the MUMPS system as it was originally developed.
2. Both host operating systems, as well as the two FORTRAN dialects are contrasted and compared. The original host system is the previously-mentioned PDP-11/03 running the RT-11 operating system. The target host system is Data General Corporation's Advanced Operating System (AOS) and the DGC dialect of FORTRAN 77 that runs on it. RT-11 is, for the purposes of this project, a single-user operating system with

minimal user support outside that given to development in programming languages. AOS is a multiuser, interactive operating system with extensive user support both in terms of program development and in terms of built-in system utilities requiring no programming. This "new" implementation of the MUMPS interpreter for AOS is a stand-alone program which allows users the full range of facilities available under AOS, including systems utilities, other language development, application programs, etc.

3. Several contemporary MUMPS interpreters are discussed, and trends among them are noted. One of these is DGC's MUMPS interpreter, which uses a separate data base management system rather than one integral to the interpreter itself. This shows a trend toward disintegration of functionality from the original idea of the MUMPS system. The concept of the MUMPS global variable facility is discussed, with possible new implementations noted.
4. The actual translation process, with specific technical solutions to certain problems encountered, is related and discussed. Included in this are the transportation of the

interpreter's sources to a DGC development environment, translation of the sources into a DGC FORTRAN 77 dialect, producing a running interpreter, and investigating the MUMPS interpreter under a different host environment. As a result of this investigation, ideas for further improvement of the FORTRAN-based interpreter came about, and these are given in the latter part of the thesis.

5. The appendices contain technical data of interest to the developer who wishes to modify or enhance the FORTRAN-based interpreter: software tools that were used, development environments, and macros used in building the AOS-based interpreter. Additionally, benchmark data comparing several MUMPS interpreters is given.

CHAPTER 2

HISTORICAL OVERVIEW OF MUMPS

The purpose of this chapter is to present an historical overview of the MUMPS language and of the FORTRAN-based MUMPS interpreter. MUMPS is compared with another interpreted language: BASIC. The goals of the FORTRAN-based MUMPS interpreter project are given and discussed. Innovations and improvements brought about in translating RT-11 MUMPS to AOS and FORTRAN 77 are discussed.

I. History of MUMPS

MUMPS (Massachusetts general hospital Utility MultiProgramming System) is both a general-purpose interactive programming language and a multiprogrammed time-sharing operating system in which the language is embedded. Together they are known as the MUMPS system, and this system has been implemented on many combinations of hardware or software. A list of vendors of these implementations is available from the MUMPS Users' Group (see List of References for address). Both the language and the operating system were developed in the Laboratory of Computer Science at Massachusetts General Hospital in the late 1960's for interactive data management applications on small computers, first the DEC PDP-9, then later the PDP-11 and PDP-15 (Walters 1982).

In the years 1960 - 1965 experiments had been done at the Laboratory with assembly language systems. These proved unsuccessful due to long development and debugging times and excessive turnaround times for even trivial program changes (Bowie 1976). In addition, assembly language was difficult for medical personnel to learn and use on their own. This led to one of the two dominant design objectives for MUMPS: "to provide a language environment where programming development was relatively easy and where programming modification could occur in a graceful and timely fashion." (Barnett 1981) Development of an operating system that met these criteria was imperative at that time, because operating system support for the developer on a mini-computer was practically nonexistent. There were no multi-user operating systems, which meant tight rationing of scarce computer resources, around-the-clock scheduling of computer time, and relatively low production. The single-user operating systems were extremely limited in terms of development tools: text editors, compilers or assemblers, system utility programs for file manipulation, backup and restore operations, etc.

The second design objective was to take the greatest advantage of available computer resources. Since computers and peripheral equipment were still relatively expensive, slow, and of low capacity, this was

very important. MUMPS emerged as "a dedicated language on a mini- to moderate-sized computer configuration." (Barnett 1981)

The main features of MUMPS are a comprehensive set of operators and functions for the manipulation of variable-length string data and a symbolically referenced, hierarchically structured global data base. These features were taken primarily from three other languages: JOSS, a time sharing system for the solution of numerical problems which was developed at the RAND Corporation; FILECOMP, developed by the General Electric Company; and STRINGCOMP, developed by Bolt, Beranek, and Newman, Inc. (Bowie 1976)

There have been many versions and implementations of MUMPS on different computers since the original. Each unfortunately was a distinct and incompatible dialect of the parent language (Walters 1982). By early 1979, for example, there were over 1000 installations worldwide using MUMPS systems on 26 different kinds of computers. These include not only the minicomputer class originally targeted by the MUMPS developers, but several mainframe and microcomputers as well (Greenes 1981). By 1972, the federal government--spurred in part by its considerable investment in MUMPS application programs--began to study the feasibility of a MUMPS standard. Under sponsorship of the National Bureau of Standards and the National

Center for Health Services Research, major users of the language cooperated to produce Standard MUMPS in 1975 (Walters 1982). In 1977, this became the current ANS MUMPS X11.1-1977, only the fourth computer language to be so accepted. Currently under review is the X11.1-1983 standard, which features new, commonly-accepted extensions to the language.

Salient features of the language are

- a. Machine independence. Despite the fact that there are many dialects and implementations of MUMPS, programs written in ANS Standard MUMPS are readily transportable in source form from machine to machine.
- b. General purpose function library. This includes many string manipulation functions not found in other languages such as FORTRAN and COBOL.
- c. A data management system which includes local variables that are stored in symbol space in the MUMPS machine workspace as well as global variables that are stored in a sparse array structure which is usually kept on disk due to its potential size. Variables may have any number of subscripts. Globals are denoted internally to the program by a circumflex ("^") before a normal MUMPS variable name. These

global variable structures give MUMPS so much of its power and simplicity of use.

d. The interpretive nature of MUMPS is positive in several ways:

1. MUMPS can be used in either immediate (type a command, get a response) or program (write several commands, then execute them as an entity) mode. This makes it quick and easy to learn the language, which was a goal of the original developers. Ideally, they wanted the physicians and other hospital personnel to be able to write their own application programs, since as users, they had the best ideas of what they wanted.

2. Interactive debugging is easy, since the language itself is used as a debugging tool. Breakpoints can be set and contents of variables can be typed out in immediate mode. Changes can be made directly in source code, and the effects of these changes can be seen immediately. Memory requirements are less in this environment as well, since a separate debugger is not necessary.

3. Program development is likewise simplified. Since the equivalents of a compiler and a linker as well as a text editor are integral parts of the MUMPS machine, everything needed for development is present and relatively simple to use. There is also no waiting for compiles or links. This reduces learner frustration by shortening the loop between trying a technique and finding either success or an error.

II. Comparison with BASIC

Perhaps the best way to describe MUMPS in terms of grammar, operators, and other linguistic structure is to compare it with another, more familiar interpreted language: BASIC. MUMPS is very similar to BASIC both in form and operation with respect to the assignment statement, arithmetic and string operators, loop control structures, and Boolean (IF) tests. Differences lie in MUMPS's global variable capabilities and non-strict data typing, which BASIC does not have. A great advantage that MUMPS has over BASIC is the ability to "DO ^filename." This command is, in effect, a "push" transferring control to another program or portion of a

program temporarily. When execution in the other program finishes, control is returned to the statement following the DO command. BASIC allows programs to be RUN from within other programs, but this is a permanent transfer of control. The GOSUB command in BASIC allows a "push/pop" transfer of control, but the subroutine must be present in the calling program. The MUMPS DO command can be used to transfer control to a program not resident in core, but on disk. Using the DO command is an easy way to incorporate well-defined programs performing discrete tasks into another program, thus shortening program development time and program size. MUMPS lacks the trigonometric functions of BASIC. BASIC requires line numbers; MUMPS allows labels, which is another distinct advantage.

III. FORTRAN-Based MUMPS Interpreter

The following deals with the history of the FORTRAN-based MUMPS interpreter, called RT-11 MUMPS hereafter in this thesis, which is the basis for this thesis project. Specifically addressed is how RT-11 MUMPS differs from other implementations.

During 1980 a group at the University of Tennessee implemented a single-user/multiterminal version of Standard MUMPS in FORTRAN to run on small PDP-11 computers under the RT-11 operating system (O'Kane 1980).

RT-11 FORTRAN is based on the FORTRAN 66 standard with extensions and is compatible with DEC-10 FORTRAN as well as other dialects of FORTRAN found on many mini- and micro-computers (O'Kane 1983). There were several goals for this implementation project, each of which is a difference from the majority of MUMPS interpreters. The goals are:

1. A system written in a high level language. Until recently, in most cases it has been the custom to do systems programming and development, including the writing of compilers and interpreters, in assembly languages native to the various target machines. MUMPS interpreters have been no exception: they have for the most part been written in assembly language, meaning that they are relatively difficult to understand and maintain, let alone convert to run on different systems. The MIIS dialect MUMPS interpreter, written in assembly language, will run on Data General NOVA and ECLIPSE and DEC PDP 11/35 machines. As these machines have totally different assembly languages, this means that two versions of the same interpreter must be maintained, updated, enhanced, etc. RT-11 MUMPS is different in that it is a multiple-machine-portable, high-level

language implementation of a MUMPS interpreter. It has the strengths of any high-level-language-based program: relative ease of portability, understandability, maintainability, the ability to be changed or enhanced quickly, etc. In the four years since its development, RT-11 MUMPS has been ported from its original PDP 11/03 under RT-11 to other operating systems running on the PDP 11 series (TSX+, RSX-11M, IAS, RSX-11D, RSX-11S) as well as other machines: the DEC-10 (under TOPS-10), the VAX under UNIX, and the IBM 360/370 series under VM/CMS (O'Kane 1983). These versions were able to retain a great deal of the original RT-11 code, based on FORTRAN 66, since this is also the basis for their FORTRAN compilers. Additionally there is the current implementation in FORTRAN 77 (another dialect of FORTRAN) on a totally different machine architecture: the DGC ECLIPSE.

2. "A version of MUMPS which could interface not only with multiple operating systems but with other language processors and [subsystems]." (O'Kane 1980)

In order to overcome certain inherent weaknesses in the MUMPS language (i.e., numeric

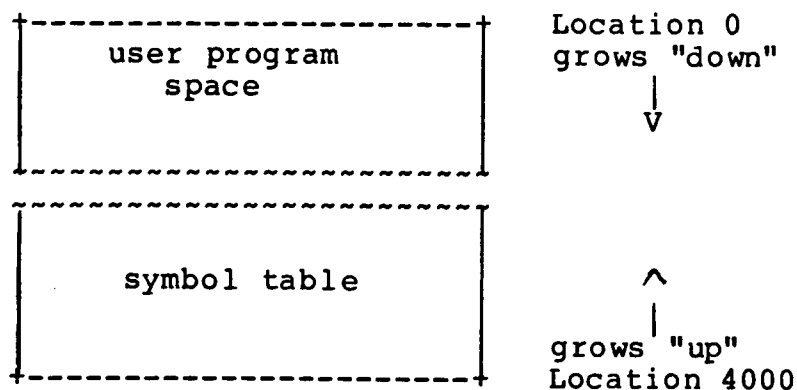
manipulation) or simply to be more useful, RT-11 MUMPS is designed to "utilize host operating system services whenever possible as well as conform to the host system program linkage conventions." (O'Kane 1980) This includes eventual usage of such things as DBM systems, statistical packages, networking, etc.

3. A system which could be compiled and run on a microcomputer with limited memory.

To date, most MUMPS systems have been developed to run on mini- to mainframe computers. With the current price reduction in microcomputers, a MUMPS interpreter running on micros would make available to a wide audience the multitude of MUMPS application programs from the past decade and a half. The educational value of such an interpreter both for medical and computer science environments would be great.

4. An efficiently-operating system. Even though RT-11 MUMPS is written in a high-level language, it compares very favorably with the UCD V2.20X assembly-language-based implementation in terms of speed and efficiency.

Following is an illustration of user memory layout in a typical MUMPS implementation.



The user program space includes immediate-mode commands as well as stored programs. In RT-11 MUMPS there is a resident MUMPS program whose function it is to accept and execute individual MUMPS lines as they are input by the user. This program is resident at all times in the "CLI" mode and is reloaded after MUMPS program execution. The program is:

```
W !,">" R x X x G +0
```

It first Writes the prompt (">") on a new line, then Reads a line of 132 characters or fewer from the input device into variable "x". It then eXecutes the MUMPS line in variable "x" and Goes to line 0 in the program space: itself.

IV. Innovations and Improvements

This project, which will be called F77 MUMPS, is different from the RT-11 MUMPS in three major ways. The first of these is that it is rewritten in FORTRAN 77.

There were several reasons for this choice: FORTRAN 77 is available and widely used on many different machines. Bringing the interpreter closer to a language standard allows more straightforward porting to other machines than would have been possible with the RT-11 FORTRAN version. Character variables and other features of FORTRAN 77 are more compatible with interpreter writing than features of other dialects of FORTRAN.

Closely coupled with the first difference is rehosting of the RT-11 MUMPS on DGC hardware under a DGC operating system. Given the task at hand as well as the work environment, this was inevitable. Adaptation of the interpreter to run on DGC systems gives the opportunity for further development and enhancement within the DGC product line. The closeness of AOS and AOS/VS allows the interpreter a migration path upward to the MV/10000: currently the most powerful DGC computer. In addition, there are relatively few implementations of the MUMPS interpreter in any form on DGC systems. Currently there are the MIIS dialect, DGC's own MUMPS, and the F77 MUMPS implementations.

The third difference is the removal of multitasking code and capabilities from RT-11 MUMPS. Traditionally, MUMPS has been implemented in a MUMPS machine: a totally dedicated computer in which only the MUMPS system is available to the ordinary user. Due to differences

between AOS and RT-11, which will be discussed in a later chapter, the decision to remove multitasking from the RT-11 version is a necessary and sound one.

An improvement but perhaps no innovation is the addition of documentation to several hundred lines of FORTRAN source. This documentation of the operation of the interpreter within the source code as well as documentation of the development process in this thesis can help other developers and users of the interpreter.

CHAPTER 3

OPERATING SYSTEM INTERFACE

This chapter gives an overview of the AOS and RT-11 operating systems, how F77 MUMPS interfaces with AOS, and a discussion of FORTRAN 77.

I. AOS Overview

The Advanced Operating System (AOS) is a Data General Corporation proprietary operating system designed to run on the ECLIPSE family of 16-bit processors. AOS is a timesharing, resource-sharing, multitasking, multiprocessing OS that supports a UNIX-like tree-structured disk directory structure on a variety of different hardware. Currently, AOS will run on machines that are near the extreme ends of the spectrum: from the larger, more powerful S/280 systems handling up to 2 meg of main memory and 32 terminals to the Desktop Generation Model 10 SP systems handling a 15 meg hard disk, 256 KB of main memory, and one terminal. This is a significant reason for the choice of AOS as the development and execution environment. The advantages are several:

- * F77 MUMPS can be hosted in many different machine environments with few or no changes,
- * MUMPS programs and files can be transported via existing DGC utilities to the same wide variety of machines, again with no changes,

- * One MUMPS interpreter can serve the needs of different users with different requirements both in terms of machines and costs. This is true in part due to the nature of F77 MUMPS. It is not a "MUMPS machine" per se, but a MUMPS utility running with other programs on AOS, allowing the use of all machine and operating system resources.

II. AOS System Concepts

Under AOS the quantum of system use is not the program but the process. A process is an entity which consists of a program in some phase of its execution, and which is allocated resources: memory, CPU time, devices, etc. Processes are referred to by Process IDentifiers (PID's), and there may be as many as 64 PID's allocated at one time on an AOS system. Each time a user logs on, for example, the user is allocated a PID for the CLI program; when the user wishes to run another program--a text editor, the MUMPS interpreter, etc.--the user is allocated an additional PID for each program. When the user stops a program, the PID is deallocated and returns to a common pool. Since a program may be overlaid and otherwise change in the course of its execution in terms of core-resident code, the process is a better concept of system use.

Subordinate to the concept of process is that of task. A task is an asynchronous flow of code execution through a process's logical address space. More than one task can be allocated under one process simultaneously. The MUMPS interpreter has only one task at present, but since AOS FORTRAN 77 supports multitasking, other tasks could be introduced. An obvious candidate would be a task to handle user I/O and increase throughput.

Address space is one of the resources allocated to a process. This space is divided into shared and unshared pages of 2 KB each in length. The pages may hold either code or data. If two users are running copies of the MUMPS interpreter at the same time, there will be only one copy each of the shared pages of the interpreter in main memory at one time, thus saving not only memory space but the I/O necessary for swapping as different processes are given control of the system. The limit of logical addressability on a 16-bit ECLIPSE is 32,768 words, or 32 pages. As the interpreter is larger than 32 pages in terms of total code, it must be divided into resident and overlaid code, each of which has its own area of shared and unshared pages.

Another resource allocated to a process is disk space and access. Disk space under AOS is ordered into directories, which may contain files and/or other directories. AOS controls file access in several ways.

With each disk file is associated an Access Control List (ACL) containing the owner of the file and the type of access given to other users: Execute, Read, Write, etc. A search list, which is a list of directories to be searched for files, is maintained for each user, and the search list is dynamically changeable while the user is logged on. If the user wants to execute a program, AOS looks for that program in the current directory. If the program is not found there, the search list is examined. The first occurrence of the program is the one executed. The ACL, the search list, and the user's home directory can be set up to restrict a user's access to the user's own copies of MUMPS programs, the global files, etc., while allowing one copy of the MUMPS interpreter to be used by everyone on the system.

III. A Comparison Between AOS and RT-11

RT-11 MUMPS runs under the RT-11 SJ/FB (Single Job or Foreground/Background) operating system. Of itself, RT-11 can support one job with the SJ monitor or two jobs with the FB monitor. Essentially, a program--or job--takes direct control of memory, peripheral devices, etc. It is the job's responsibility for opening device channels, scheduling use of devices and memory, and having a fairly intimate knowledge of the machine on a level not too far above that of device drivers. RT-11

does have built-in functions and capabilities in its system library, but these are language-oriented (for MACRO-11, an assembler, and FORTRAN IV) rather than their being there for a nonprogramming user.

As has been stated, AOS is a multiprogramming operating system, so it takes on the task of resource scheduling and allocation. Programs are not allowed direct access to peripherals unless these peripherals have been explicitly left out of the operating system's repertoire during system generation. Access to system resources is controlled on a per-user basis in part by an Access Control List (ACL) associated with each file (including directories and peripherals) in the directory structure. Furthermore, each user has a particular program which is run immediately after logon. This is normally the CLI program; but if one wished to restrict a user or group of users to run only one program, such as F77 MUMPS, this program could be set up as the initial program for these users. Under AOS there is the concept of system constants such as OUTPUT or INPUT for each task; the programmer can default to these without having to be concerned about device specifics. RT-11 puts a good deal of responsibility on the programmer to keep track of these things. With AOS the programmer can concentrate on the application and allow details to be defaulted if desired. A good descriptive term is that

AOS exhibits a higher degree of abstraction from the hardware that it is, after all, supposed to make available to the user than does RT-11. The programmer is still able to specify details and exercise a high degree of control with AOS, but this is not required. For example, under RT-11 the Job Status Word, which is a certain memory address, must be adjusted to change a device from line input to individual character input. To change the characteristics of a device under AOS, a CLI command, CHARACTERISTICS, can be used to change, for instance, baud rate, parity, device type, and whether the device produces 7-bit or 8-bit characters. RT-11 MUMPS requires an individualized system start file; F77 MUMPS will run under any AOS system.

IV. Overview of FORTRAN 77

FORTRAN 77, as described in the ANSI 1978 standard, is a version of the FORTRAN language that originated in 1954. FORTRAN 77 is in wide use today. The Data General Corporation version of FORTRAN 77 includes the full standard with extensions. Extensions used in this version of the MUMPS interpreter are:

1. Comments on the same line as a statement may be begun with an exclamation point (!).
2. Tabs may be used in source lines.

3. Quotation marks (") are used interchangeably with apostrophies (') in delineation of strings.
4. The %INCLUDE compiler directive allows inclusion of other source files in the current program. The %LIST directive allows suppression and restoration of compiler output to the listing file.
5. The standard specifies a four byte size for integer, real, and logical data types; however, DGC's FORTRAN 77 allows specification of integer and logical data types with size of two bytes. This is done via a compiler switch (/INTEGER=2) and/or via program type declaration, e.g. INTEGER*2 INTVAR.
6. Certain word and bit manipulation functions: BTEST.
7. Allowing character and noncharacter data within the same common block.

There are several significant differences between DGC's FORTRAN 77 and RT-11 FORTRAN. These differences will be expanded upon in a following chapter.

- * RT-11 FORTRAN relies on LOGICAL*1 arrays for character variables; FORTRAN 77 has a CHARACTER data type, along with functions and other methods for manipulation:

- a. Implicit substrings by use of the colon (:).

A(1:2) = "hi"

- b. Concatenation of strings by double slashes (//).

A = "a"//"b"

- c. The LEN function returns the typed length of a character variable.
- d. The CHAR(n) function returns the character whose ASCII code is n.
- e. The ICHAR(a) function returns the ASCII code for the character a.

All RT-11 "character" variables must have an extra byte for an ending null. FORTRAN 77 character variables do not require this.

- * RT-11 FORTRAN allows system subroutine calls with variable numbers of arguments. FORTRAN 77 requires correspondence of argument lists between subroutine calls and declarations.
- * RT-11 FORTRAN relies on certain RT-11 system routines that do not exist in AOS or in the FORTRAN 77 subroutine library. Examples are IPEEK, IPOKE, SUBSTR, etc. In other cases, such as the LEN function, a routine of the same name but different qualities exists.

CHAPTER 4

A COMPARISON OF MUMPS INTERPRETERS

This chapter compares and contrasts the FORTRAN MUMPS interpreters with the Data General (DG), Micro, and DEC (DSM-11) MUMPS interpreters. This is done not only in terms of functional capability but in terms of system philosophy, target user audience, and potential user environments. Since RT-11 and F77 MUMPS's have already been compared, this won't be repeated. A synopsis of the other systems will be given first, followed by a general comparison section.

I. DG MUMPS

Data General's MUMPS is an implementation of the full ANS 77 MUMPS language with Type A extensions. It conforms to the proposed 1983 ANSI standards defined by the MUMPS Development Committee. It runs under AOS on machines with the Character Instruction Set and under AOS/VS. The interpreter runs as a shared process, allowing simultaneous access by multiple users as well as use of all other AOS(/VS) facilities, utilities, languages, programs, etc. DG MUMPS is not, therefore, a classic MUMPS system. It is a much more flexible "utility" that allows development in other languages, use of existing system utilities, and use of a powerful,

interactive operating system. DG MUMPS was developed for wide use in the existing Data General user community, as it requires the DGC product INFOS II for global variable management.

DG MUMPS is even further disintegrated from the MUMPS machine in terms of data base management; it uses the DGC INFOS II file management system to implement global variables, which are called Named System Storage. INFOS files--and the MUMPS global variable files are no exception--can be referenced by several DGC languages and utilities, including COBOL, PL/I, Extended Business BASIC, and PRESENT. PRESENT is a utility capable of accessing and formatting data based on multiple conditions, sorting data, computing values, writing reports, and making line, bar, and pie charts. Both nonprogrammers and non-MUMPS programmers can use PRESENT's very high level language commands to manipulate a MUMPS data base without using the MUMPS language or interpreter (DG INFOS User's Manual). The use of INFOS also provides for networking through DGC's XODIAC network software, thus allowing users access to Named System Storage areas regardless of data location. Within the interpreter, the command ZGLOBAL specifies the current global variable area (DG MUMPS User's Manual).

II. DSM-11

"DSM-11 [Digital Standard MUMPS] is a multi-user data base management system that consists of an interactive, high-level programming language, a data base management facility, and a time-sharing executive." (DSM Technical Summary) It is a MUMPS system in every sense of the word: a dedicated system offering only MUMPS. It seems targeted for large installations where a MUMPS environment is desired: hospitals, MUMPS applications shops, medical research facilities, etc. DSM-11 will run on any of the PDP-11/34, 11/45, 11/60, or 11/70 processors and allow up to 63 simultaneous users. Ownership of terminals and other peripheral I/O devices is established by the MUMPS OPEN command. Devices are associated with MUMPS channel numbers: the console terminal is always device #1, the line printer is always device #3, etc. Spooling to LPT's is available as well. Networking is allowed among as many as five systems running DSM-11, thus facilitating global file access.

A good feature of DSM-11 is support for journaling of data entries. If desired, an additional copy of data modified on disk is written to another device. In case of system failure, the journalled entries can be used to restore disk files to their most current state. As a general backup system, the Standard MUMPS Backup and Utility System (SMBU) is a part of DSM-11. SMBU is a

bootable, stand-alone system that labels disks and tapes, formats disks, and provides complete backup and restore functions for MUMPS as well as non-MUMPS disk files. Presumably, to run SMBU, one has to shut DSM-11 down. This seems very inconvenient (DSM Technical Summary).

III. Hitachi Micro MUMPS

From the Systems Development Lab of the Hitachi Corporation comes Micro MUMPS, a "practical subset of Standard MUMPS," implemented in PL/M on an Intel 8085 with 63 KB of main memory and 20 MB of disk (Watanabe 1981). It is a single-user MUMPS machine on a micro targeted for such environments as small business, planning sections, doctors' offices, labs, and so on. Due to the single-user nature of the system, the developers decided to remove certain Standard MUMPS commands, functions, and system variables. These include LOCK, OPEN, CLOSE, \$JOB, indirection (@, XECUTE, \$TEXT), \$RANDOM, and SELECT. Some of these choices are sensible; others are questionable. Why leave out \$RANDOM and indirection?

New language features include

- * ZCALL call subroutines written in other languages.
- * ZP output catalog of MUMPS routines.
- * ZG output catalog of global variables.

* \$ZC, \$ZN parameter values obtained by ZCALL.

* ZGO resume execution from a keyboard interrupt.

(Watanabe 1981)

IV. Comparisons

Three of the implementations are full MUMPS systems, having all three characteristics thereof: DSM-11, RT-11 (multi-user versions), and Micro. F77 MUMPS retains an internal B-tree handler for global variable handling (and is thus two-thirds of a MUMPS system), while DG MUMPS is only an interpreter, depending on the operating system and a data base utility for its other abilities. Details of the global file implementation methods are:

System	Method
DG	external INFOS II
DSM-11	internal multiway B-tree
F77	internal B-tree
Micro	internal modified prefix B-tree
RT-11	internal B-tree

The code sizes of the interpreters are:

System	Code size
DG	64 KB + 12 KB overlays (AOS version)
DSM-11	40-64 KB **
F77	64 KB + 168 KB overlays *
Micro	13 K "steps"
RT-11	46 KB resident + disk-based overlays *

* Includes B-tree handler.

** Does not include buffers and user space.

Three of the interpreters allow for networking of some sort: DG, DSM-11, and F77. Implementation languages are, where known, high-level languages. In fact, the system utilities for DSM-11 are written in MUMPS. Except for Micro MUMPS, all versions are written for minicomputers. F77 MUMPS can be run on the micro-ECLIPSE-based Desktop Generation 10-SP under AOS, though.

CHAPTER 5

DATA BASE CONSIDERATIONS

One of the main uses of the original MUMPS machine was that of a data base manager. The operating systems and systems utilities of that time were not sophisticated enough to provide good data handling capability, so an integral data base management system was a necessary and welcome part of MUMPS. The actual internal representation of the data bases and symbol tables did not matter at that time, since the MUMPS programmer and user were concerned only with the global variables available to them through the MUMPS machine.

As time passed, operating systems and utilities became smarter and more capable. File and data formats became more standardized due to the need for information exchange between system utilities and user programs. Current emphasis is on integrated software using common data formats provided by codified utilities throughout a product line, such as AOS/ECLIPSE or across product lines, such as DIF (Data Interchange Format) used in certain microcomputer products: VisiCalc, VisiPlot, the Personal Filing System, etc.. There is now a greater need to decentralize the MUMPS machine, as has been done with F77 MUMPS by removal of multiple user timeslicing capabilities. FORTRAN 77 and the AOS CLI provide line

editing functions for commands and program entry within F77 MUMPS, thus bringing about another standard.

Remaining in F77 MUMPS are the interpreter itself and the data base facility, which uses a B-tree data structure to represent the sparse, hierarchical global variable system. Other implementations of MUMPS use other methods. DG MUMPS uses the DGC INFOS II data base utility, onto which it offloads the burden of maintaining globals. To the MUMPS programmer and user, the actual method of implementing globals doesn't matter: it's invisible. However, using a separate DBMS facility for F77 MUMPS would have several effects:

1. It would reduce the code size substantially. The benefits are that fewer modules are necessary, and less effort is needed for maintenance and enhancement. There would, of course, be the initial overhead of recoding and testing.
2. It would enable other utilities to access MUMPS files in a standard manner.
3. Portability of the product would be reduced.
4. With the advent of data dictionaries, it would allow cross-language availability of MUMPS data structures.
5. It offloads the burden of data management onto another process, which under AOS is

asynchronous, thus maximizing use of the machine.

6. It allows networking under AOS and DGC's XODIAC.
7. It expedites backup and recovery of data base files. The MUMPS machine has no innate provisions for this; users have to write their own routines for the purpose, which is at present a redundant task.

A candidate for DBMS to be used with F77 MUMPS is the Data General/Data Base Management System (DG/DBMS). This is a CODASYL-type data base manager that supports network data base structures with Data Description and Data Manipulation Languages. DG/DBMS is accessible by applications programs written in COBOL, PL/I, FORTRAN 5, and FORTRAN 77 or through the PRESENT utility, which has been described earlier. DG/DBMS offers the advantages given in the list above (DG/DBMS Reference Manual).

CHAPTER 6

METHODOLOGY AND PROBLEMS WITH F77 MUMPS

The following is a brief account of the development of this thesis project and some of the reasons behind the methodology as well. The main goals have been stated in Chapter 1; they are enlarged upon here.

The interpreter source files were received on an RX02 (8-inch) format diskette along with a hardcopy listing of the files. The listing was a valuable aid for consistency checking against these files. These files were transferred to an ANSI standard format magnetic tape, which the DGC AOS system utility LOAD was able to read. Once filed on disk, they were the same as any other text files except for the DEC standard use of a carriage return/new line pair as terminators. AOS uses a single new line.

The first step was to select which version of DGC FORTRAN compiler to use. The choices were FORTRAN IV, MP/FORTRAN IV, FORTRAN 5, or FORTRAN 77. Both of the versions of FORTRAN IV were rejected out of hand for several reasons:

1. Neither will run on the new DGC 32-bit MV series machines. This is the architecture toward which DGC is migrating and is very important now; ignoring these machines would be a mistake.

2. Several important language features possessed by RT-11 FORTRAN, such as easy string and byte manipulation, were absent from these versions. Byte manipulation is necessary for symbol table and global variable storage.
3. Both versions of FORTRAN IV are bound quite closely to the RDOS and AOS operating systems in a way similar to that in which the RT-11 FORTRAN is bound to RT-11. This degree of operating system dependency was to be avoided in the final product, primarily for reasons of portability.

DGC's FORTRAN 5 is very close to the ANSI 1966 FORTRAN standard, so it is quite close to RT-11 FORTRAN, which is based on that standard with extensions. It might have been a good choice except, as with the FORTRAN IV's, it lacks adequate byte and string manipulation facilities. In addition, FORTRAN 5 seems to be a transition product between FORTRAN IV and the eventual choice, FORTRAN 77. Following are the reasons for selection of FORTRAN 77:

1. It has good character and byte manipulation facilities that closely parallel those of RT-11 FORTRAN not only in function but in form as well. This means less time spent in conversion and less likelihood of error.

2. It is supported fully on the 16-bit ECLIPSE and the 32-bit MV-series architectures.
3. It will run under DGC's high-level language debugger, SWAT.
4. It is essentially operating-system-independent, meaning that it can be relatively easily transported to other machines.
5. It is a fairly young dialect of a very old language, which means it will be around for some time and supported on a number of different systems.

The next step was to try to compile the unmodified source code to see specific areas of trouble. This was the first step in a series of incremental development steps. The project was not completely visualized at its beginning; a learning process about RT-11 FORTRAN, FORTRAN 77, and the workings and requirements of the MUMPS interpreter took place during the project. During this time extensive new documentation was added to the source files as an aid during the project as well as an aid to further development. Obvious syntactic differences between RT-11 FORTRAN and FORTRAN 77 were identified and changed. In this class are such items as built-in system functions that differ between the two FORTRANs, or direct translation of RT-11 LOGICAL*1 character storage to CHARACTER storage in FORTRAN 77.

Other differences, such as the ways terminal I/O is handled, required more thought and broader changes.

Some of the initial changes turned out not to work and had to be changed again. An example of this is the implementation of the MUMPS global variable system, which is described in Chapter 2. In the original RT-11 version, the system was implemented as disk files consisting of LOGICAL*1 (byte) arrays containing both numeric and character data, pointers, keys, etc. As the discovery of how to convert the internal representation of REAL and INTEGER type variables to CHARACTER type variables and vice versa had not yet been made, INTEGER variables were used in the global file arrays instead of CHARACTER. This worked well enough, but the files were quite large (more than twice the size of the original RT-11 files) and access was very slow. After the innovation of conversion between numeric and character types, the global file handling was reconverted to resemble the original RT-11 methods much more closely. Both disk file size and access time were greatly reduced by this, too. In the global variable handling routines conversion from character to numeric types is done only when numeric manipulation is necessary, thus reducing the extra overhead required in the FORTRAN 77 version.

Another non-straightforward decision during this time was about the single/multiuser question. The RT-11

interpreter was set up to be either for one user or up to four users, depending on certain parameters set up at compile time. For this, the interpreter had code to handle time slicing, individual terminal I/O, swapping of user space and information, etc. This method was based on the way RT-11 handles device allocation and I/O. The MUMPS interpreter (or any program running on RT-11) can open devices and perform I/O directly; therefore, the interpreter could gain exclusive control of up to four terminals at once and set up its own scheduling, etc. Under AOS a resident process called EXEC owns all the terminals; other processes cannot get exclusive control of them. The only way other processes could get control would be for the terminals to be disabled; however, EXEC would then be unaware of them, and they couldn't be used to log onto AOS. They would be strictly MUMPS terminals, and other terminals couldn't be used by MUMPS unless they were disabled as well. This seemed to be an unnecessary restriction, so the decision was made that F77 MUMPS be single-user only. Since AOS is a multiprogramming operating system, multiple users of AOS can run MUMPS at the same time. An area where trouble can still occur is in access to global variables by more than one MUMPS user. This problem is discussed in Chapter 7.

Compilation, correction, and reformulation continued until error-free object modules (in the AOS

form filename.OB) were produced. At this point the modules were linked together using the same overlay layout as in the RT-11 version. AOS overlays are similar in structure to RT-11 overlays: they are segment-oriented. This didn't work because the resident code (.MAIN), an overlay area, system routines, etc. wouldn't fit into memory; the FORTRAN 77 interpreter was larger than the RT-11 interpreter. One reason for this is the need for room in memory for FORTRAN 77 runtime routines which are linked in from libraries. RT-11 FORTRAN doesn't need these routines to be resident since it can utilize RT-11 system routines directly through special system calls.

Taking out the multiuser support code (as it wasn't being used) from the main routine didn't work: .MAIN was still too large. Another alternative, which proved successful, was to write a dummy .MAIN program and transform the previous main MUMPS routine into a subroutine which would reside in an overlay area. The only job of the dummy .MAIN would be to call the MUMPS subroutine and end the program when the subroutine returned. This scheme worked well enough to allow debugging the interpreter to its current state. After extensive analysis a chart of which routines in the interpreter called other routines was made. This allowed the determination of an overlay map different from the original RT-11 map, and the new map worked very well.

Another experiment was to remove some less-essential code from the main MUMPS module and to overlay it. As a result, the MUMPS module fit in as .MAIN again. However, due to problems with the interface with the SWAT debugger, the resultant program could not be debugged. The current version of the interpreter has the MUMPS module overlaid as described in the previous paragraph. This is debuggable, and it is best left that way for future development.

The last step in the development process was methodical testing and debugging. Individual language features were tested according to charts and tests given in the MDC MUMPS Primer. Benchmark tests given in O'Kane (1980) were the final step. Results of these tests are given in Appendix D.

There were several particular problems with porting the MUMPS interpreter from both the RT-11 operating system and RT-11 FORTRAN. They are given here.

I. Translation of LOGICAL*1 to CHARACTER

Data Types

RT-11 FORTRAN has no concept of the character data type. As a substitute, it uses elements of LOGICAL*1 arrays as characters, since these elements can take on values of 0 - 255: the full 8-bit ASCII character set. An interface to the operating system allows "character"

I/O between these LOGICAL arrays and devices. FORTRAN 77 supports the CHARACTER data type and strictly disallows values for LOGICAL variables other than .TRUE. and .FALSE. . Therefore, a translation of LOGICAL*1 to CHARACTER data types was necessary. This entailed not only the simple swap of data typing statements, it meant changing from handling to the handling of strings. In several cases this resulted in savings in source code, since in the RT-11 scheme multiple assignment statements (often within a loop) are necessary for "string" assignments; whereas in FORTRAN 77 one assignment statement would often do.

Since FORTRAN 77 strictly disallows the EQUIVALENCing of type CHARACTER with any other data type, many problems arose, since in the RT-11 version, LOGICAL*1 was freely EQUIVALENCed with numeric data types. The main reason this was done was to provide symbol table and global file storage of type REAL variables in their internal four or eight byte representations. "Conversion" to and from type LOGICAL*1 was automatically done by the EQUIVALENCE. The scheme used in F77 MUMPS to circumvent this is given in a later section.

II. Conversion of Numeric Internal Representation to Character

The original MUMPS interpreter entered symbols into the symbol table either by putting the LOGICAL*1 bytes into the table for character data or by putting the internal representation of a REAL*4 data item (4 bytes) into the table for numeric data. This last was done by the following method:

```
REAL*4    REALVAR
LOGICAL*1 CHVAR(4)

EQUIVALENCE (REALVAR, CHVAR)
```

Type CHARACTER variables are used in place of type LOGICAL variables in the current interpreter version. FORTRAN 77 disallows EQUIVALENCING character and noncharacter variables, so another method had to be used to insert four bytes of REAL*4 variable into the symbol table. As the symbol table variable itself was type CHARACTER, obviously a type CHARACTER variable had to be used. A loophole exists. FORTRAN 77 performs strict data type checking within program modules, but it allows (or perhaps cannot enforce) type "conversion" through named COMMON blocks that may contain variables of one data type in one instance and variables of another data type in another. These variables share memory, though; and the correct orientation of variables can cause "conversion" between REAL*4 and CHARACTER*4, for example.

A simplified version of the conversion process from REAL*4 to CHARACTER*4 is as follows: in the first program is a common block PRSERC and a call to a conversion routine, CONNC (CONvert Numeric to Character):

```

      REAL*4    R4CTMP    !FOR CONVERSION R*4  <-> CHAR
      CHARACTER CR4TMP*4  !FOR CONVERSION R*4  <-> CHAR

      COMMON   /PRSERC/ R4CTMP, CR4TMP

      R4CTMP = 3.14159
C
C      PERFORM CONVERSION
      CALL CONNC
C
C      AT THIS POINT, CR4TMP CONTAINS THE INTERNAL
C      REPRESENTATION (FOUR BYTES) OF R4CTMP
      END

      SUBROUTINE CONNC
C-----
C  PURPOSE: TO CONVERT A NUMBER TO A CHAR STRING
C  EQUIVALENT.
C-----
      CHARACTER*4 S1, S2 !FOR CONVERSION R*4 -> CHAR
      COMMON   /PRSERC/ S1, S2
C
C      COPY
      S2 = S1

      RETURN
      END

```

The COMMON block has the same name, but different contents, in the main routine and in the subroutine. The assignment statement in the subroutine is necessary as a simple transfer of like data types. On return from the subroutine, the type CHARACTER variable "contains" the equivalent of the type REAL*4 variable. Conversion from character back to real is accomplished by another subroutine and another named COMMON block containing two

REAL*4 variables. Other routines to convert between INTEGER*2 and CHARACTER*2, and INTEGER*4 and CHARACTER*4 variables were also written.

III. Transportation of Sources From RT-11 to AOS

As Appendix B shows, the source system (a PDP 11/03) has only RX02 diskettes as removable media for file transportation. Dr. O'Kane sent the RT-11 FORTRAN sources on such an 8-inch diskette. Data General equipment available cannot read RX02-format diskettes. As a solution, an intermediate DEC machine having both an RX02 diskette drive and a magnetic tape drive was used. The DEC utility PIP (Peripheral Interchange Program) was used to move files from the diskette onto the magnetic tape in standard RT-11 format, which is a subset of the ANSI standard. The resulting tape was readable by the DGC AOS utility LOAD.

The only further problem, and a minor one, was that RT-11 uses a carriage return/new line pair as text line terminators, while AOS recognizes both characters as terminators. The result was that all source lines were double-spaced. With a text editor it was a small task to remove the extra characters. A corresponding problem would occur in trying to move sources back to an RT-11 system from an AOS system, as AOS uses only new line in text strings as a terminator.

IV. Removal of RT-11-specific System Interfaces

The RT-11 operating system offers FORTRAN language programmers system call capabilities through the system subroutine library which is stored on the system volume (RT-11 Programmer's Reference Manual). Services available through this library include character string manipulation, two-word integer (INTEGER*4) support, low-level device handling, and interrupt handling. The RT-11 version of the MUMPS interpreter utilized these services, meaning that it was in some essential ways system dependent: non-portable. Equivalent routines are available for FORTRAN 77 from AOS through FORTRAN 77's libraries of functions, or through writing routines in FORTRAN that accomplish the same things as the system routines under RT-11.

Difficulties arose in porting from RT-11 to AOS for several reasons.

The system calls in RT-11 are more closely bound with the system than similar calls under AOS. An example of this is the Job Status Word under RT-11. Bits in this word are set and reset to allow or disallow character by character input from a TTY keyboard. There is no equivalent concept under AOS; instead, FORTRAN 77 handles lower-level I/O.

In some instances FORTRAN 77 routines did not do the same things as their RT-11 counterparts. For

example, the LEN function returns in RT-11 FORTRAN the actual non-null element count in a LOGICAL*1 array; in FORTRAN 77 the function returns the total size of a CHARACTER variable, regardless of the number of characters in the variable. If a CHARACTER*5 variable STRVAR contained "HI", LEN(STRVAR) would return 5, not 2. This is a singularly useless piece of information. To get the same functionality in FORTRAN 77 as in RT-11, a routine to scan a string for the first null character and report the string's length based on that information had to be written.

V. Creation of FORTRAN Routines to Emulate RT-11 System Builtins

Following is a list and brief description of the routines which had to be written in order to emulate RT-11 system routines. A future enhancement would be for someone to try to remove the primary references as much as possible to cut down on space and system calls used.

The following were necessary due to FORTRAN 77's proscription of EQUIVALENCEing type CHARACTER with any other data type. Hence, the internal representations of integer and real numbers had to be "converted" into type character variables to be stored in the MUMPS symbol space.

- a. INTEGER*2 FUNCTION CCI2(STRING1) converts the 2-byte STRING1 to I*2.

- b. INTEGER*4 FUNCTION CCI4(STRING1) converts the 4-byte STRING1 to I*4.
- c. REAL*4 FUNCTION CCR4(STRING1) converts the 4-byte STRING1 to R*4.
- d. CHARACTER*2 FUNCTION CI2C(INT1) converts the I*2 number to a CHAR*2 string.
- e. CHARACTER*4 FUNCTION CI4C(INT1) converts the I*4 number to a CHAR*4 string.
- f. CHARACTER*4 FUNCTION CR4C(REAL1) converts the R*4 number to a CHAR*4 string.
- g. SUBROUTINE CONCN converts a character string to a number equivalent. This routine does a wholesale conversion of character strings to REAL*4, INTEGER*4, and INTEGER*2 types even if only one of these conversions is desired. Otherwise, three subroutines would be necessary.
- h. SUBROUTINE CONNC converts a number to a character string equivalent. See notes about CONCN above.

The following routines emulate RT-11 system calls.

- a. SUBROUTINE CONCAT(STRING1, STRING2, STRING3). This concatenates STRING1 and STRING2 and places the result in STRING3.

- b. INTEGER FUNCTION ISCOMP(STRING1, STRING2).
This compares STRING1 and STRING2 and returns an integer value i which is < 0 if STRING1 < STRING2, = 0 if STRING1 = STRING2, and > 0 if STRING1 > STRING2.
- c. SUBROUTINE PUTSTR(LUN, IN, CHARAC, ERROR).
This outputs the string in IN on device LUN. CHARAC may contain a FORTRAN carriage control character, and ERROR is the system error on return.
- d. INTEGER FUNCTION RTLEN(INSTR). This returns the number of characters in INSTR up to but not including a terminating nul. If there is no nul in the string, the entire string length is returned.
- e. SUBROUTINE SCOMP(STRING1, STRING2, RESULT).
See ISCOMP above.
- f. SUBROUTINE SCOPY(STRING1, STRING2). This copies STRING1 to STRING2, appending a nul at the end of the result string.
- g. INTEGER*4 FUNCTION SECNDS(OTIME). Returns the current system time in seconds past midnight minus OTIME. The result is an INTEGER*4 value, not a REAL*4 value as in the RT-11 function.

- h. SUBROUTINE STRPAD(INSTR, LENG, ERROR). This pads INSTR with rightmost blanks until it is LENG long. ERROR is set to .TRUE. if the result string is greater than the original length of INSTR.
- i. SUBROUTINE SUBSTR(INSTR, OUTSTR, BEGIN, LENG). This copies a string from INSTR to OUTSTR beginning at the BEGINth position in INSTR and going for LENG characters.
- j. SUBROUTINE TRIM(INSTR). This shortens INSTR by removing all trailing blanks. It appends a nul afterwards.

This section details ways in which the current FORTRAN-based MUMPS interpreter differs from the ANS MUMPS Standard. Information is from O'Kane (1980).

- 1. The LOCK command is not recognized. Timeouts are not permitted in the OPEN command, but are permitted in the READ command.
- 2. E-type (scientific) notation is not recognized.
- 3. The ASCII characters NUL (0) and those whose numbers are greater than 128 are not permitted. NUL is not permitted because it ends all strings.
- 4. The maximum number of significant digits recognized in numbers either for input or for output is 6.

5. The \$DATA function returns only 0 or 1 for local variables.
6. Argument indirection in the USE command is not permitted.

CHAPTER 7

SUGGESTED IMPROVEMENTS FOR FORTRAN MUMPS

This chapter is an annotated list of suggested future enhancements to the FORTRAN implementations of MUMPS. Some of these enhancements are specific to the DGC Advanced Operating System (AOS); others apply to this FORTRAN implementation of MUMPS.

I. AOS-specific Enhancements

- a. The addition of a CLI "push" facility would be welcome. DGC's AOS is a multiprocessing operating system which allows user processes as well as system processes to produce child processes. While in MUMPS, then, it would be useful and desirable for the user to be able to suspend the MUMPS program process, create a new "child" process, and transfer to that process running the CLI. The user could then perform any normal CLI function, such as examining directory contents, listing files to the terminal or to the printer, entering new directory structures, sending messages to other processes, etc. Several DGC products have this CLI push facility, and these have been very useful.

- b. Currently, the internal edit calls (ZV, Z=, ZL, etc.) do not conform to any DGC editor commands either in name or in function. For conformity, ease of use, and genuine usefulness, the internal MUMPS editor should be changed to conform to the DGC SED editor. There are several reasons for this.
1. By default, terminal input in DGC's FORTRAN 77 uses SED commands and protocols insofar as the effects on the line edit of various control characters are concerned. Anyone familiar with SED will be able to use FORTRAN terminal input without trouble, and this should extend to the internal editor as well. As added reinforcement, the SED line edit conventions are the same as for the CLI.
 2. SED is a solid, well-documented, well-defined editor. It is not extremely complex, so it should be easy to emulate within MUMPS.
 3. SED is the DGC editor of choice when preparing MUMPS source programs outside of the MUMPS interpreter. One of the main reasons for this is that SED produces true TAB characters instead of groups of

spaces. MUMPS relies on an initial TAB character to distinguish between an internally-stored MUMPS program line and an immediate mode command.

- c. Currently, F77 MUMPS is compiled, linked, and run under DGC's Advanced Operating System for the ECLIPSE processor. As FORTRAN 77 is also available for DGC's Advanced Operating System/Virtual Storage for the new series of MV machines, the MUMPS interpreter could very easily be ported to this new environment. This is not done currently because AOS FORTRAN 77 and AOS/VS FORTRAN 77 work in slightly but significantly different ways. The main difference lies in the strictness of data typing. For example, the compiler will raise errors if an assignment is done between an INTEGER*2 variable and an INTEGER*4 variable. As the AOS and RT-11 compilers were not as strict as this, many such assignments are performed and accepted in the current MUMPS interpreter code. These would have to be redone for transportation to AOS/VS FORTRAN 77.

One simplification to the MUMPS interpreter that would come from its being compiled under AOS/VS FORTRAN 77 in its 32-bit

code generation mode is that the need for overlays is removed. Indeed, AOS/VS prohibits overlays in its 32-bit mode of operation. Under AOS, overlays must be used due to the fact that the MUMPS interpreter is too large for the code space of 32,768 16-bit words. It is not too large for 32-bit execution under AOS/VS.

- d. To use F77 MUMPS, one invokes it with the AOS CLI EXECUTE command, then loads and runs MUMPS programs interactively under the interpreter, providing input through the keyboard and receiving output on the CRT screen. Therefore, the use of the interpreter requires an operator's presence, whether the application is a really interactive one or not. AOS, as well as being an operating system that provides a high degree of interactivity, also provides the capability to submit jobs to a batch queue using the CLI command "QBATCH." It would be useful to be able to batch a MUMPS interpreter so it could run detached from any terminal or human interaction; however, at present, there is no method for passing information to F77 MUMPS. The interpreter would need to know such things as what MUMPS program to load and run,

and on what disk files or devices to get input and give output. AOS can pass parameters given in the original CLI command to the program being EXECUTEd or QBATCHed. A suggestion for F77 MUMPS parameters which is close to AOS standards is:

```
EXECUTE MUMPS mumps_program_name outfile/L infile/I
```

where mumps_program_name is the MUMPS program to be loaded and executed, outfile is the file to which all output would be directed, and infile is a file from which all input would be taken.

II. General Enhancements

- a. Improving the error messages and adding an "on-line" help facility are good additions. The existing error messages could in several cases be made more specific. An example is the undifferentiating "FILE ERROR" message that is given for any file error. The interpreter could be made smarter so that it would give a "FILE DOES NOT EXIST", "FILE IS NOT A MUMPS SOURCE FILE", "FILE IS IN USE", or "FILE ALREADY EXISTS" error message in appropriate circumstances. Some error messages could be

made interactive, so that in addition to reporting that a file already exists, the interpreter could ask if that file should be deleted or left alone.

An on-line help facility, perhaps invoked by a "ZHELP" command, would present an overview of how to run this particular implementation of MUMPS. It would list and explain the "Z" commands, how to load, execute, and save programs, how to run the internal editor, how to change the MUMPS initialization program, how this installation works, etc. This would not be an exhaustive help facility, but it would serve to get someone started immediately without their having to seek out a manual. The help facility could be put at small cost into another overlay under AOS.

- b. Although not strictly an enhancement to the MUMPS interpreter per se, a useful addition to the MUMPS product would be an interactive program to allow the user to specify
 - 1. global file names and sizes,
 - 2. a MUMPS startup program name and the contents of that program,
 - 3. a different prompt character, or
 - 4. device characteristics for the MUMPS I/O devices.

This MUMPSGEN program would modify or create a characteristics file that would be read by the MUMPS interpreter each time it was invoked, so that these features could be changed dynamically rather than by recompilation. A default characteristics file would be supplied with the product, so the user wouldn't be required to run MUMPSGEN unless the user wanted to change information.

- c. The ability to issue CLI commands directly within the MUMPS interpreter would be handy, especially in an environment such as AOS in which there are many powerful and useful CLI commands. This ability could be added to the interpreter, but new CLI command facilities would have to be written for each new CLI environment under each new operating system every time the interpreter was ported. The alternative would be to leave out this facility when porting to a new operating system. The facility could be modularized in an overlay so that minimum effort would be required to remove it.

DG MUMPS makes several AOS(/VS) CLI and INFOS commands directly available to the MUMPS user within the interpreter either in program or immediate mode. These commands are prefixed

with the letter "Z" according to the Standard MUMPS rules for language extensions. They are:

- * ZCREATE pathname creates AOS files.
- * ZDELETE pathname deletes AOS files.
- * ZGLOBAL index file changes INFOSglobal files.
- * ZLOCK global locks an INFOS file.
- * ZRENAME filename1:filename2
- * ZWORKDIR pathname changes the current directory.
- * ZEXECUTE pathname executes any AOS(/VS) program file (.PR extension) including CLI.PR.

(DG MUMPS User's Manual).

- d. The current MUMPS interpreter does not meet ANSI standards fully. Upgrading to complete ANSI standards would be a significant enhancement. See Chapter 6 for a list of areas where the ANSI standards are not met.
- e. Documentation is an important part of every software product. A more comprehensive user's guide to this MUMPS interpreter is needed. The guide might consist of two sections: one describing the basic MUMPS interpreter, and one (which would be installation-specific) describing the mechanics of installation and

maintenance of the interpreter on a specific machine and operating system. There would be several different versions of the second section, but only one of the first.

- f. Further internal documentation is needed.
- g. This could be made into a true multiuser MUMPS machine by implementing the multiuser features in the original interpreter. This would require some adjustments to AOS, however, since terminals are not available under AOS in the same way they are available under RT-11. These would be dedicated MUMPS terminals, unable to log on to the machine except through MUMPS, unable to use the AOS CLI and utilities, etc.
- h. Implementing the LOCK command for global variables is necessary if multiple users are to share a global file pair (string file plus index file). Otherwise, synchronization would be impossible to guarantee. There are several ways to implement the LOCK command:
 - 1. OPEN exclusive and CLOSE each global file for every global variable request. Have the interpreter wait and try again if the file is already open. This is simple but very inefficient.

2. Put a "current owner" field into a global file header. This field will be read and updated if possible on every global variable request. This method narrows the synchronization window, but does not close it.
3. Create a file server program to handle all global file requests. It will be this program's job to keep track of owners of global variables. This solution closes the synchronization window, but is the most complicated of the three to implement.

The DGC product ICOBOL runs under AOS and has an ISAM file system capability. Since multiple users could want access to the same record in an ISAM file simultaneously, there is a lock server program called MLS that handles file access requests. All actual I/O is performed by system call routines within ICOBOL; the lock server synchronizes these requests among users. A scheme such as this for F77 MUMPS could handle global variable requests from different users.

4. Use a DBMS with a built-in lock mechanism

instead of the B-tree handler currently in the FORTRAN MUMPS. See the discussion on global variables for further information.

CHAPTER 8

CONCLUSIONS

This thesis project has been an enlightening exercise in exploring two different dialects of the "same" language, i.e., FORTRAN, in comparing two different operating systems, and in investigating the MUMPS language from its origin to its many current implementations on different machine architectures and under different operating systems. The RT-11 interpreter has been successfully transported to DGC's FORTRAN 77 and AOS, thus realizing one of the aims of the original FORTRAN-based MUMPS project: to produce a portable, high level language-based MUMPS interpreter. The RT-11 interpreter is widely and easily portable to computer systems supporting other FORTRAN 66-based dialects of FORTRAN. Similarly, the F77 interpreter is easily portable to computer systems supporting other FORTRAN 77-based dialects, including those running on UNIX and UNIX variants.

The current trend toward writing in high level languages for a high degree of software portability and maintainability, and the existence of versatile, interactivity-oriented operating systems as well as powerful data base facilities are giving impetus to nontraditional MUMPS interpreters: utilities as opposed

to MUMPS systems. Today's user community has heightened expectations of operating systems and data base utilities than had users at the time of the first MUMPS systems. The MUMPS interpreters reflect this, allowing users the full range of system utilities and facilities, as opposed to restricting the users to MUMPS-only facilities. The fact that the F77 interpreter does not include an operating system, but only an interpreter and data base manager shows that it follows this trend.

The RT-11 interpreter, as realized in the F77 interpreter, has been improved in that it will now run under a wider variety of operating systems. However, in comparison with other dialects of MUMPS, the F77 interpreter is far from complete. Both FORTRAN MUMPS's--RT-11 and F77--fall short of the ANS 1977 Standard in ways listed in Chapter 6. There is now a proposed 1983 ANS MUMPS Standard, which contains new features and enhancements. The FORTRAN MUMPS's should be brought at least to the 1977 Standard, and preferably to the 1983 Standard as well. In Chapter 7 are listed innovations for F77 MUMPS in specific and for either of the FORTRAN MUMPS's in general. These innovations would go far toward enhancing the capabilities of these interpreters.

LIST OF REFERENCES

LIST OF REFERENCES

- Barnett, G. Octo, Souder, Daniel, Beaman, Peter, and Hupp, Jon, "MUMPS - An Evolutionary Commentary." Computers and Biomedical Research, 14, pp. 112-118, 1981.
- Bowie, Jack, and Barnett, G. Octo, "MUMPS - an Economical and Efficient Time-sharing System for Information Management." Computer Programs and Biomedicine, 6, pp. 11-22, 1976.
- Data General/Database Management System (DG/DBMS) Reference Manual. Westboro, Massachusetts: Data General Corporation.
- Digital Standard MUMPS DSM-11 Technical Summary. Maynard, Massachusetts: Digital Equipment Corporation.
- FORTRAN 77 Environment Manual (AOS/VS). Westboro, Massachusetts: Data General Corporation.
- FORTRAN 77 Reference Manual. Westboro, Massachusetts: Data General Corporation.
- Greenes, Robert A., "MUMPS: Second Decade Design Considerations." Computers and Biomedical Research, 14, pp. 104-111, 1981.
- INFOS II System (AOS) User's Manual. Westboro, Massachusetts: Data General Corporation.
- MUMPS Primer, by David A. Bridger, Chairman. Washington, D.C.: MUMPS Development Committee Subcommittee #1, 1975.
- MUMPS Users' Group, 4321 Hartwick Road, Suite 308, College Park, Maryland, 20740.
- MUMPS User's Manual. Systems Division, Data General Corporation.
- O'Kane, Kevin C., "UT MUMPS User's Guide." 1980. (Computer printout).
- O'Kane, Kevin C., "A Portable Hybrid MUMPS Development System Host." IEEE Proceedings of Computer Software and Applications Conference 1983.

RT-11 Programmer's Reference Manual. Maynard,
Massachusetts: Digital Equipment Corporation.

Walters, Richard F., "Is MUMPS the Cure?"
Kilobaud Microcomputing, February, 1982, pp.
54-61.

Watanabe, Tan, Ohsawa, Tsunehara, and Kuma, Hisao, "Micro
MUMPS: An Interactive Database Language for Micro-
computers." Software - Practice and Experience, 11,
pp. 1307-1314, 1981.

APPENDICES

APPENDIX A

Software Tools

In this appendix is a list of the software tools used on AOS to convert the RT-11 FORTRAN sources into the finished MUMPS interpreter. Where available, appropriate documentation for the tools is listed.

AOS

AOS is described in chapter 3. DGC's Advanced Operating System Version 5.02 was used. Documentation includes:

- * Command Line Interpreter User's Manual: # 093-000122-05.
- * Advanced Operating System Programmer's Manual: # 093-000120-04
- * Managing AOS: # 093-000217-04

FORTRAN 77

DGC's FORTRAN 77 is described in chapters 3 and 6. DGC's AOS FORTRAN 77 Version 2.20 was used. Documentation includes the DGC FORTRAN 77 reference manual: # 093-000162-01

LINK

LINK is a relocatable linking loader used to combine the resultant .OB files from FORTRAN 77 compiles

into program (.PR), overlay (.OL), and symbol table (.ST) files. DGC's AOS LINK Version 5.00 was used. Documentation includes the DGC Link and Library File Editor User's Manual: # 093-000254-01

SWAT

SWAT is a high-level interactive symbolic debugging system. SWAT was used extensively and, for the most part, enjoyably to debug the MUMPS interpreter. The primary advantage of SWAT as a development tool is its ability to work with high-level language programs at source language level: the programmer can list the current debugging environment, set breakpoints, list variables, set new values for variables, etc., all in the program's original high-level language. SWAT is recommended highly to all FORTRAN 77 developers on Data General systems. DGC's AOS SWAT Version 2.11 was used. Documentation includes the DGC SWAT manual: # 093-000258-01

SED

SED is a screen-oriented text editor most suitable for editing MUMPS source programs, due primarily to the fact that SED produces actual TAB characters--needed by MUMPS for differentiation of program lines--rather than strings of spaces. DGC's AOS SED Version 5.00 was used.

Documentation includes the DGC SED Text Editor User's
Manual: # 093-000249-00

APPENDIX B

Development Environments

Below are described the development environments in which the original MUMPS interpreter was developed and on which the thesis project was done.

Original environment

Hardware

- * PDP-11/03 with
 1. LSI-11/2 processor
 2. KEV-11 floating point option
- * 60KB memory extended for Q-bus
- * 2 RX02-style diskette drives

Software

RT-11 SJ (or FB/XM for multiterminal support) VB3 operating system

Current development environment

Hardware

- * Commercial/350 (C/350) computer system with
 1. ECLIPSE processor
 2. floating point processor
 3. character instruction processor
- * 1 meg main memory
- * 196-MB ZEBRA disk
- * 800 BPI tape drive

Software

Advanced Operating System version 5.02

APPENDIX C

Building MUMPS

Here are given compiler and linker command lines with other appropriate CLI macros and files used in building MUMPS. Explanations are given in curly brackets under the lines in the macros.

Compile step. This macro was used to compile each MUMPS module. It is called DF77D.CLI.

[FORT_77_ENV]

{"Pushes" the current environment and establishes a "search list": a term in AOS meaning a list of directories to be searched for files.}

DEL/V/2=IGNORE %1%.LS %1%.ER

{Deletes old files. Do not issue error messages if they do not exist.}

F77/DEBUG/STRINGS=ANSI/L=%1%.LS/E=%1%.ER/O=%1%.OB %1%

{Invokes the FORTRAN 77 compiler through its own macro, F77.CLI, which is supplied by DGC with its FORTRAN 77 product. DEBUG includes extra information in the .OB (object) file output. STRINGS=ANSI disallows angle brackets for specification of strings in ASCII numeric form: i.e., <040> instead of the space character. The other switches specify listing, error, and object file names, respectively.}

POP

{"Pops" the environment to what it was before invoking this macro.}

Link step. This macro was used to link all the MUMPS modules into MUMPS_SYS. It is called LINK_MUMPS.CLI. Note that this version of the linking macro has the main MUMPS module in an overlay.

[FORT_77_ENV]

DELETE/V/2=IGNORE MUMPS_MP.LS MUMPS_SYS.**

```
F77LINK/DEBUG/CIS/MODMAP/E==MUMPS_SYS.ER      &
/L==MUMPS_MP.LS/O=MUMPS_SYS      &
MAIN      SUIO      RT11_SCALL      &
!* MUMPS      ! &
PARSE      ! &
DELETE      ! &
FCN      PATRN      PATRN1      MATCH      SVAR      ! &
GBLREF GLOBAL FIND KEYCOM FETSTR INSERT INSSTR ! &
INPUT      SQUASH      READJ      *! &
!* QSCAN      SCN1      ! &
GBLOPN      BINIT      SINIT      DELDAT      BLKADD      ! &
PUTNOD      SKCOMP      ! &
ERRMOD      ! &
GETNOD      ! &
MLIB      ! &
SYM      ! &
KILL1      INOUT      LIBX      *! &
```

POP

{F77LINK is the link macro supplied by DGC, and it invokes LINK, an AOS system program. The DEBUG switch directs LINK to produce special symbol files for the SWAT debugger. The CIS switch directs LINK to include character routines which can utilize the ECLIPSE Character Instruction Set, thus making the resultant programs faster in character manipulation. The remaining

switches give names to output files. The module names following are divided by exclamation points and asterisks into resident code and two overlay areas. This division is different from the RT-11 version of the interpreter.}

APPENDIX D

MUMPS Benchmarks

In this appendix are benchmark test results of several MUMPS interpreters and one BASIC interpreter running commonly-accepted test routines. The interpreters are F77, RT-11, UCD V2.20X, DSM-11 DEC MUMPS, and DGC's Business BASIC Rev. 3.00. Test environments are, for RT-11 a PDP 11/03 with two RX02 disk drives running RT-11, for F77 and BBASIC an ECLIPSE C/150 with a 50-MB Zebra disk running AOS 5.00, and for DSM-11 a PDP 11/34. Timings are for each of the command sequences to be executed 1000 times, and all times are in seconds. N/A indicates Not Available. Where possible, all tests were translated to the BASIC language for Business BASIC tests. Some of the MUMPS functions (i.e., \$NEXT, \$PIECE, etc.) are not available even in analogue form in BASIC.

Test	RT-11	UCD V2.20X	F77	DSM-11	BBASIC
1	6	9	9	1.46	1
2	27	28	76	1.73	2.79
3	46	34	81	1.44	1.13
4	79	41	113	2.52	1.46
5	29	N/A	109	N/A	1.41
6	49	151	104	1.46	1.09
7	63	33	145	2.00	1.42
8	65	156	189	3.44	N/A
9	31	N/A	113	N/A	2.74
10	130	60	270	3.09	N/A
11	197	69	344	3.34	N/A
12	63	40	192	2.74	N/A
13	23	32	82	2.01	3.34
14	322	1120	2490	28.54	N/A

The tests themselves follow. Given is the MUMPS command line. Initialization, where performed, is done only once; the FOR commands precede the actual features tested.

1. FOR I=1:1:1000 ; Simple loop
2. FOR I=1:1:1000 SET C="AAAAAAAAA"
3. FOR I=1:1:1000 SET C=10000
4. FOR I=1:1:1000 SET C=10*10
5. SET I=10,J=10 ;INITIALIZATION
FOR X=1:1:1000 SET C=I*J
6. SET C=10 ;INITIALIZATION
FOR I=1:1:1000 IF C=10
7. SET C="ABCD" ;INITIALIZATION
FOR I=1:1:1000 IF C?"ABC"
8. SET A=10,B=13 ;INITIALIZATION
FOR I=1:1:1000 SET C=A_B
9. SET A="ABC",B="DEF" ;INITIALIZATION
FOR I=1:1:1000 SET C=A_B
10. SET D="AAA/AAA,AAA,AAA" ;INITIALIZATION
FOR I=1:1:1000 SET C=\$P(D,"",2)
11. SET D="AAA/AAA,AAA,AAA" ;INITIALIZATION
FOR I=1:1:1000 SET C=\$E(D,12,15)
12. SET D="AAA/AAA,AAA,AAA" ;INITIALIZATION
FOR I=1:1:1000 SET C=\$F(D,"/")
13. SET D="AAA/AAA,AAA,AAA" ;INITIALIZATION
FOR I=1:1:1000 SET C=D
14. FOR I=1:1:100 SET D(I)=1 ;INITIALIZATION
FOR I=1:1:1000 SET D=\$N(D(100))

VITA

John B. Hampton III was born in Nashville, Tennessee on January 29, 1949. He attended elementary schools in that area and was graduated from Battle Ground Academy in May 1967. He attended the University of the South as a freshman, then transferred to Southwestern at Memphis, where in June 1971 he received a Bachelor of Arts degree in German. He studied linguistics at Philipps-Universitaet in Marburg, W. Germany and, in September 1972, began study toward a Master's degree in German at Tulane University. This program was not completed.

In January 1974 he was employed by Data Communications Corporation in Memphis as a computer programmer. In September 1976 he joined the Center for Business and Economics Research of the University of Tennessee, Knoxville as a programmer. In September 1977 he accepted an assistantship at the University of Tennessee and began study toward a Master's degree. He received the Master of Science degree with a major in Computer Science in August 1984.

The author is a member of the Association for Computing Machinery and is currently employed at SCI Systems in the Research Triangle Park, North Carolina. He is married to Susanna Bachmann of Princeton, New Jersey.