ARGONNE NATIONAL LABORATORY
9700 South Cass Avenue
Argonne, Illinois 60439

THE SCIENTIFIC WORKSTATION EVALUATION PROJECT:
MULTIDISCIPLINARY EXPERIENCE WITH A SCIENTIFIC WORKSTATION

By

Richard C. Raffenetti
Goran Birgersson
Roger N. Blomquist
James M. Kennedy
Dale D. Koelling
Anthony J. Policastro

Edited by
Clifford M. Caruthers

Computing Services

December 1984

1Reactor Analysis and Safety
2Applied Physics
3Materials Science and Technology
4Environmental Research
CONTENTS

ABSTRACT ........................................ vii

Chapter page
1. INTRODUCTION .................................. 1
2. EVALUATION SUMMARY ......................... 3
3. COMPUTING, SCIENCE, AND THE WORKSTATION MODEL .......... 5
   The Workstation Model .......................... 6
   Tools ........................................... 6
   Task Workstations Versus User Workstations ....... 7
   Multiuser Workstations ........................ 8
   The Work of Scientific Computing ............... 8
4. SCIENTIFIC WORKSTATION EQUIPMENT OVERVIEW .......... 11
   Hardware Capabilities .......................... 11
   Processing Speed ............................... 11
   Memory Size .................................... 11
   File Capacity .................................. 12
   Backup Capabilities ............................ 12
   Display and Keyboard Capabilities ............... 12
   Printer Accessibility ........................... 13
   Availability of Communications Ports ............. 13
   Network Communication .......................... 13
   Operating System ............................... 14
   Software ........................................ 15
   Text Editors .................................... 15
   Programming Tools .............................. 16
   Tools for Management and Analysis of Scientific Data ... 16
   Graphics Tools .................................. 16
   Writing Tools ................................... 17
5. THE EVALUATION ITSELF .......................... 19
   Evaluation Metrics .............................. 19
   Productivity Value .............................. 19
   Costs ........................................... 20
   Evaluation Difficulties .......................... 22
   Sharing the Workstation ........................ 22
   The "Central" Location .......................... 23
   Realization of the Concept ...................... 23
   Conflicting Goals ............................... 24
### Contents

Conversion of STRAW for the Workstation ........................................ 54
  Background for Code ................................................................. 54
  Brief Description of Code ......................................................... 55
  Code Details and Conversion ....................................................... 55
The Workstation as a Production Environment .................................. 56
The Workstation as a Programming Environment ................................ 56
The Workstation as a Dedicated Resource ....................................... 57
Conclusions and Reflections .......................................................... 57

10. THE ELECTRONIC STRUCTURE OF ATOMS AND SOLIDS ........................... 59
  Characterization of Evaluator ...................................................... 59
  Preconception and Expectations .................................................. 60
    General Concept ................................................................. 60
    Unix Operating System ......................................................... 61
    Ridge 32 Hardware .............................................................. 61
  The Workstation as a Production Environment ................................ 62
    Codes Involved ........................................................................ 62
  Conversion .................................................................................. 63
  Capacity ..................................................................................... 63
  The Workstation as a Programming Environment ................................ 65
  The Workstation as a Dedicated Resource ..................................... 65
  Conclusion and Reflections .......................................................... 66

11. ENVIRONMENTAL TRANSPORT MODELING ........................................... 67
  Background .................................................................................. 67
  Objectives ................................................................................... 68
  Preconceptions and Expectations .................................................. 68
  Conversion to the Workstation ...................................................... 68
  EPRI Cooling Tower Plume and Drift Models ................................... 69
  RTM-II--Long Range Transport Model for Air Pollutants ................... 70
  UI/ANL Community Noise Prediction Model ..................................... 71
  Overall Evaluation of the Workstation ......................................... 72
  Conclusions ................................................................................. 74

12. OPERATING A WORKSTATION ......................................................... 75
  Installation .................................................................................. 75
  Education ..................................................................................... 76
  Communication ............................................................................. 76
  System Administration .................................................................... 77
  Maintenance .................................................................................. 78
  Documentation .............................................................................. 79
  Graphics ...................................................................................... 79

ACKNOWLEDGMENTS ........................................................................... 80
Appendix

A. PROGRAMMING TOOLS ........................................... 81

Program management .............................................. 81
Source code management ........................................... 81
Macro preprocessor ................................................. 82
Compiler ................................................................. 82
Static Analyzer ....................................................... 82
Subroutine Library .................................................. 82
Linker ................................................................. 83
Run-Time System ..................................................... 83
Debugging Aids ....................................................... 83

BIBLIOGRAPHY .......................................................... 85

INDEX ............................................................................ 87

LIST OF TABLES

Table page

1. Workstation Reservation Statistics ......................... 25
2. Total Elapsed Demonstration Times ......................... 28
3. Evaluation Project Dates ....................................... 33
4. The Four Largest VIM Users ................................. 49
ABSTRACT

This document is a formal report on the scientific workstation evaluation project. The objective of this project has been to evaluate a scientific workstation. When new computing systems appeared that exhibited the capabilities necessary to do scientific computing work at relatively low equipment cost, both the staff of the Argonne scientific divisions and we (the Computing Services staff) wished to know more about the equipment and the vendor claims.

Using a workstation for scientific distributed computing was a new concept that we were not entirely certain how to evaluate. We were sure, however, that the work of scientific computing was not so simple that cursory exposure to a new and unfamiliar computing resource would tell the whole story. Therefore, we proposed that we would acquire the equipment most suitable to the evaluators' needs and submit that equipment to intensive evaluation. The evaluation as well as the specification and selection of the equipment would be done by those Argonne scientists and engineers who have wondered how they might use such machines in their own work.

We invited members of the technical staff to help us define a minimum configuration and minimum workstation capabilities. After the machine was selected and delivered, we worked to integrate it into the Argonne computing environment. Following this integration, for five months of the six month evaluation period, user evaluators had an opportunity to use the workstation to do their scientific computing.

This report contains the experiences of a diverse group of evaluators. Therefore, we have organized this report into chapters. Chapters 1 and 2 introduce the reader to the project. Chapters 3 and 4 give background into our view of a workstation, and Chapters 5 and 6 provide some perspective of the evaluation. Chapters 7 through 11, each written by one of the five evaluators, describe their experiences. These last five chapters detail the more significant results of this project. Chapter 12 describes the experiences of establishing and managing the equipment. An appendix serves to provide information that is not germane to the evaluation itself, but that may be of interest to those acquiring workstation equipment. A useful reading of this report would not omit any of the perspective chapters, but you may wish to read only those evaluation chapters that relate to your own scientific or engineering computing.

The procedures through which we determined the proper equipment are detailed in R. C. Raffenetti, Scientific Workstation Comparisons Using a Benchmark, ANL/TM 415 (Argonne National Laboratory: Computing Services, April 1984).
Chapter 1

INTRODUCTION

The Argonne National Laboratory Computing Services Scientific Workstation Evaluation Project has involved (1) the acquisition of a single-user workstation in a competitive procurement, (2) the integration of that workstation into the Argonne environment, and (3) a multidisciplinary evaluation of that workstation.

Computing Services initiated this Project to gather information on the suitability of using scientific workstations at Argonne National Laboratory. The Laboratory will use this information in planning for this new computing resource. We hope that the scientific and engineering staff will be able to identify their computing requirements and experiences in relation to those of the participants in this project.

The technology of computing equipment is advancing rapidly. The appearance of large-scale integrated circuits has resulted in a desire for miniaturization of all components of standard computer systems. Companies have formed to integrate the pieces and are marketing complete computer systems, both hardware and software, which promise to bring large scale computing capabilities into the office environment.

A forerunner of the scientific workstation (which we define below) is the so-called personal desktop computer, itself a kind of workstation. The personal desktop computer is a complete computer, consisting of a processor, a mass storage device (floppy or hard disk), an input device (keyboard), an output device (display), as well as capable operating software to manage the hardware, a file system, and a run-time environment for programming and program execution. Most personal desktop computers are based on 8-bit or 16-bit microprocessor integrated circuits (chips).

The architecture of the processors used in personal desktop computers cannot meet the needs of "heavy-duty" scientific computing work. Scientific computing and mathematical modeling in particular require floating-point computer arithmetic with computer word sizes of 32 bits or greater and large logical memory space to accommodate the large programs and extensive arrays of program data. However, having mastered the design and implementation of 16-bit microprocessors, semiconductor companies are now striving to implement a generation of 32-bit microprocessors and related circuitry. Out of these efforts have appeared the complete computer systems which are capable of scientific computing.
The makers of fast, low-price minicomputers are also entering the workstation market. The primary reason for their appearance appears to be the availability of Reduced Instruction Set Computer (RISC) architecture. The main principle of RISC architecture is that a simplified instruction set is sufficient to do the work. For example, RISC machines have register operations but do not have operations that use data directly from memory. Advantages of the RISC architecture stem from the simplification of design and implementation. Rather than design a new microprocessor with a reduced instruction set, some vendors have been able to use off-the-shelf components to build up a processor on a board or a set of boards. The necessity of designing and implementing integrated circuits is thereby eliminated, and the remaining design is at the board level, where the design difficulties are better understood. The best and fastest of the existing integrated circuits can be employed. The time that elapses between the start of design and the product validation can be so short that technology advancements will not already have rendered the design goals outdated. The product is therefore more competitive than it might have been, had the design and implementation taken longer. The workstation that was selected for this evaluation project is a RISC architecture machine.

Another way to use RISC architecture to build computers cost-effectively is to use an RISC machine as the basis for emulating a machine with a more complex instruction set. In this case, the user software and the operating software rely on the complex instruction set, which in turn uses the RISC hardware. In all cases, if an RISC machine is advantageous, it is because the reduction of implementation tasks reduces costs without sacrificing the performance goals.

Although equipment prices for the capability level characteristic of a scientific workstation are now relatively high, we believe that in the future prices will drop to the point where it might be practical for each Argonne heavy-duty computer user to have his own dedicated workstation. Past history indicates that as soon as some resource which has been shared becomes economical in comparison with the manpower wasted by the sharing efforts, sharing stops. Computer terminals are a case in point of a shared resource that proliferated for personal use when prices dropped.

Note that future price reductions will not accrue only to small machines. Parallel price-performance trends should occur for minicomputers and mainframe computers. The choice of using a personal, dedicated workstation for scientific and engineering computing or of using the other available computing resources will depend on other considerations as well as on price.
Computing Services has initiated and completed a project to evaluate a scientific workstation. This report documents the experiences of the project participants, a group consisting of personnel from Computing Services and from some Argonne scientific divisions. Because the computing needs of each of the participants were diverse, there are few common conclusions.

In the remainder of this document, we discuss our observations and conclusions. A summary of these observations and conclusions appears below:

- A workstation is a dedicated computing system that supplies a fixed quantity of computing resource for its user. A workstation's interactive response and its total throughput are limited by the speeds of various components of the workstation hardware and the operating system software.

- Workstations are capable of running scientific programs. The compilers, linkers, and run-time system, as well as the other tools or utilities, are able to contend with the special needs of large programs.

- A workstation is not uniformly successful as a production computing environment because of limitations in processing resources, such as file space and processing power. While limitations resulting from a lack of optional equipment can generally be eliminated through acquisition of that equipment, limitations in processing power are more difficult to transcend.

- Workstations offer a potential for increased user productivity which has not been confirmed by this work. The period of evaluation in this project has probably been too short to reveal the benefits of using new tools. Many valuable tools were not tested.

- Evaluators agreed that the lack of a fast and versatile data communication option inhibited their work and limited the workstation's value. A good communication mechanism is essential in a workstation and in other computing resources.

- The lack of a convenient backup mechanism for files made an unpleasant chore out of a very important file maintenance task. A good file backup mechanism is necessary in a workstation to...
The Scientific Workstation Evaluation Project

encourage users to maintain security against unrecoverable data losses.

• New workstation products suffer from software immaturity, sometimes in features which are of extreme importance, such as the compiler and the debugging tools. The debugging tools are especially necessary to determine if bugs have resulted from user program errors or from system software difficulties. Execution bugs in the latter category are very disruptive.

• Certain difficulties existed because participants shared a system and did not have a dedicated system for their work. The sharing caused a number of circumstances out of the ordinary for a dedicated workstation.
Chapter 3
COMPUTING, SCIENCE, AND THE WORKSTATION MODEL

To discuss the scientific workstation model, it is useful to gain some perspective of the motivations for workstation use. For the ensuing discussion we define two extreme types of computing:

- Shared computing: This type of computing is performed on computing equipment that serves multiple users simultaneously. For example, a central, multipurpose computing installation consists of shared multipurpose computers that do the computing for an entire organization. These shared computers are centrally located and serve the diverse needs of the user community. Batch subsystems are shared and accessed through queueing of batch jobs, and interactive subsystems are shared by apportionment of small "slices" of resources to each system user.

- Dedicated computing: This model differs from shared computing in that users with dedicated computing environments do not share their computing hardware resources. Dedicated systems may share information through networking and/or other forms of computer communication, but the processing resources remain dedicated to a single user; they are not accessible to other users.

A developing concept in computing is "distributed computing". In common usage this term sometimes means only a computing system model consisting of a network of shared systems. In other cases, the term refers to a system involving only dedicated computing (not shared computing) with communication capabilities for sharing data. It is, however, unlikely that a distributed environment without shared systems would survive, mainly because many resources are not cost-effective unless shared. Typical shared resources are "array processors" and peripherals such as fast printers and high-capacity mass storage.

The main advantage of shared computing is the reduced cost per user realized by sharing expensive resources, such as the operating costs (manpower, electric power, air conditioning, etc.) of a supercomputer. The main advantage of dedicated computing is that the processing resource is always available to the one user; therefore, response time will always be the same.

The structure that represents the best elements of the shared and dedicated computing models is a networked hierarchy of distributed shared systems (not necessarily in one location) and dedicated systems. The shared resources must be easily accessible to every system user. This computing model provides both for the sharing of expensive resources and for computing resources dedicated to individual users.
THE WORKSTATION MODEL

A workstation is an element in the distributed computing service model. It is a dedicated computing system that provides familiar, responsive computing resources with which the operator/owner can do day-to-day work. For example, if the user does programming, then the workstation must supply all of the necessary tools for the programming task. The tools should include, but not be limited to, a text editor to create and modify programs, a compiler for the implementation language, a linker to build executable images, libraries of useful modules that the linker can search, various testing and debugging aids, and, most importantly, a run-time system. The run-time system should include all the elements necessary to execute images, including a run-time library and a run-time debugger. With these tools the programmer does not have to rely on external resources.

In some cases, the workstation may not be the system on which the application will ultimately run. The workstation must therefore have the tools to design applications that can be easily transported to a designated system.

The workstation must be a useful, capable system standing alone. It does not include any of the shared, special purpose resources. However, a workstation that provides easy access to other resources is more powerful because it provides that access.

The work environment of a workstation can be duplicated by shared computer systems. For example, any workstation having a Unix operating system would have a programming environment similar, if not identical, to that of some other Unix environment. Sharing makes a resource unpredictable in the sense that a given user may not be able to get a fixed share of the resources all of the time. (If a shared resource is more powerful than necessary for the user base, then users will be satisfied until the user base outgrows the resource. On a workstation, if the user needs exceed the workstation capacity, then the alternatives are to acquire the necessary resources either externally to the workstation or to buy a replacement or upgrade to the workstation.)

Tools

A tool is a piece of equipment that is designed to aid the craftsman in doing a task common to his work. In the computing industry, the phrase "software tool" denotes those programs which computing craftsmen or other professionals use in doing their computing work. Many tools used to be known as utilities; they were designed to perform the tasks common to the general user. The software tools concept is broad because it covers specialized tools as well as general

---

tools. The specific name is associated with the development of the Unix operating system. The Unix "workbenches" imply sets of tools to do certain specialized computing work (e.g., the programmer's workbench, the writer's workbench, the documenter's workbench). A workstation, being one kind of "workbench", should have the tools which are appropriate for the tasks being carried out. The tools which are appropriate for a scientific workstation are implied by the tasks of scientific computing (see below).

The Unix system is particularly useful as a workbench because it has a system by which simple tools can easily be combined to do more complex jobs. This concept is fundamental in the Unix system. The Unix interactive environment, called the "shell", has the concepts of "standard input", "standard output", and "standard error" data streams, and each process has all of these streams. The ability to redirect the data streams is an important feature which adds flexibility to the working environment. The standard input stream may be taken either from a file or from the terminal keyboard. The standard output streams may be diverted either to files or to the terminal display. Perhaps the most useful capability is that the standard input stream of one process may be taken from the standard output stream of another. This system is called a "pipe" and is the mechanism for linking tools together. The Unix multiprocessing system not only runs all of the processes communicating through pipes concurrently, it also manages the synchronization of the information moving through the pipes.

Task Workstations Versus User Workstations

The workstation concept in this project is that of a user-dedicated system. The projected cost of a workstation should be low enough relative to the cost of manpower to devote a workstation to a single user. We assume that this low cost will permit a user workstation to be idle some of the time without that lack of use becoming a matter of concern. (Note that past history confirms this assumption. Personal terminals are commonplace where once they were considered a shared resource for a group of users.)

A related concept of "task-dedicated" workstations may apply to near-term use of scientific workstations because, until their value can be substantiated, they seem relatively expensive. We use the term "task-dedicated workstation" to mean a single-user computer system that is used by several users, but not by all at the same time. This is the operational model for use of any very expensive equipment. For example, an X-ray machine operated in a hospital is a task-dedicated machine. Such equipment is usually scheduled by some sort of "appointment" mechanism. An example closer to the workstation area is a turnkey computing system for integrated circuit design. Electrical engineers

---

schedule time on the equipment to use its special software for their design work. The system is not dedicated to a single user.

Multiuser Workstations

Another concept for the use of workstation-type computing equipment is that the equipment be shared. That is, it must be capable of providing the environment and resources for multiple users at the same time. This capability must include not only sharing of the interactive resource to give adequate and fair response to all users, but also disposition of the background processing cycles. The issue here is whether all users of a shared, workstation-type computing system can be assured of getting their fair shares.

This model is useful insofar as the particular equipment can provide adequate and well-managed resources. If the equipment is not significantly overpowered, then its resource management capability is critical. The equipment used in this project is advertised as a multiuser system. Our minimal experience with its operation in multiuser mode indicates that the capability of the computing resource to provide good responses for multiple users is limited. Even if the software does the sharing job well, limitations may occur because of the hardware. For example, all file system activity, including normal I/O, paging, and swapping, may go to the same disk, creating a bottleneck which causes work to be suspended.

THE WORK OF SCIENTIFIC COMPUTING

In science, computers perform a variety of direct and indirect tasks. Argonne, for example, uses computers not only for research but also for administrative and technical purposes. The technical staff use special-purpose dedicated computers for laboratory data acquisition as well as the general-purpose computers such as the divisional VAX computers and the central IBM computers.

In this project, we found that scientific workstations were of interest to those scientists who construct mathematical models of physical processes and use them for simulation to answer real questions. That work is common because experiment simulations done with models are often more practical than physical experiments. The practicality may derive from cost as well as from safety, time, and other factors. For example, building a nuclear reactor without first doing simulations may lead to the discovery, after considerable time and expense, that the design was not the most effective choice. Mathematical modeling of physical processes is a very large segment of the computing that is done by the Argonne scientific and engineering staff. Modeling is a major activity of the Laboratory.
In considering how the new generation of workstation equipment will be used at Argonne, we concluded that the technical staff might benefit from powerful workstations. The work of a person who does modeling can include the following computing tasks:

- Model development
- Program development
- Production computing
- Data management
- Data analysis
- Document preparation

In model development, short programs are written quickly to test theory and conjecture. After satisfactory models are found, better programs are written to be more effective for "heavy-duty" simulations. These programs will also be refined as usage dictates. Production computing is a process in which the modeling programs are used extensively to create the desired simulations and to acquire new data. Acquired data must be managed and also analyzed to produce the overall results of the modeling efforts. A variety of documents are necessary to communicate the results of a scientific research or engineering development program.
Chapter 4

SCIENTIFIC WORKSTATION EQUIPMENT OVERVIEW

We have previously defined the model workstation as a dedicated system that provides familiar, responsive computing resources with which the operator/owner can do day-to-day work. A scientific workstation is a workstation that provides the tools necessary for scientists to do their work. The hardware and software must be appropriate for the scientific computing task. The scientific workstation must accommodate the specialized needs of its user.

This chapter does not address specialization further, but it does enumerate the software tools necessary for each of the technical tasks described in Chapter 3. We also discuss the hardware resource requirements for workstations in general.

HARDWARE CAPABILITIES

The subsections below discuss some of the important capabilities of workstation hardware:

Processing Speed

In a single processor system, the processing speed determines how quickly single tasks are completed. The power of the processor must be sufficient to accomplish its tasks within a reasonable time. The large range of tasks in scientific work requires an equivalent range of processing speeds. For example, programming and run-time testing require more processing power than do the setting up of jobs for execution on a remote computing system or the charting and analysis of the acquired data.

Memory Size

In scientific computing, large programs manipulate extensive data arrays. The capability to provide virtually unlimited memory capacity for programs, either with physical or with virtual memory, frees the programmer from solving problems which are not directly related to scientific goals. A virtual memory system readily provides for growth and for the occasional extraordinary need. (We point out that virtual
memory, while lifting certain size constraints placed on programs, does not ensure that computing resources are used efficiently. A phenomenon called "thrashing" occurs when a computer system uses more resources to manage a program's virtual memory than to complete its work.)

File Capacity

The need for file space (disk) grows during the lifetime of any system. For example, the space requirement for operating system software will increase because of changes and enhancements to the system. Users will develop new programs and accumulate data that they will keep online for ready access. Increased file capacity through networked file servers and through additions to the hardware will certainly be necessary.

Backup Capabilities

The workstation user must guard against system failures by routinely "backing up" the files. Backed up files are normally preserved on mountable media, but they can be preserved on separate, remote file storage systems in a network. Backing up files on a remote file system can be more convenient for the workstation operator, who then does not have to manage the physical backup media. If remote file backup is not practical, then we recommend a high-capacity, fast, mountable device such as a streaming cartridge tape. The higher the capacity, the fewer media the user will need to use and manage. As file backup becomes easier, users are more likely to backup their files.

Display and Keyboard Capabilities

Displays which permit viewing of large portions of files, both in length and width, are especially valuable. Additionally, larger screen sizes and graphics capabilities may be used by the software to provide multiple "windows" in which separate tasks may be viewed, all at the same time.

The keyboard is the primary input device through which the user communicates with the system. A "QWERTY" key arrangement is the most common, and added program function keys should be available for special applications. Other input devices associated with the keyboard and used to manipulate special applications (such as "windowing" or "menu" selection) will also increase productivity by making input faster. A touchpad, a mouse, and a light pen are examples of fast input devices. One limitation to productivity is the speed at which the user of a computer can communicate with it.
Printer Accessibility

The capability to produce printed output is invaluable to users whose work habits include the analysis of paper copy. Workstation features and tools often are designed to avoid the need for printed copy, but until users adapt their work habits, a local printer or access to one through a network is necessary. A port for connection to a local printer is thus also necessary.

Availability of Communications Ports

Computer-to-computer communication can be implemented by using asynchronous communications ports (usually RS232's). While this communication option is not necessarily desirable because of the limitations in capability, it provides flexibility in accessing systems to which network access may not be practical, at least in the near future. A communications port configured with an auto-answer modem is also valuable to permit users to access the workstation from remote locations (e.g., to inquire about the progress of some important task). Therefore, two asynchronous communications (RS232) ports (and two modems) are useful in a workstation, one to communicate out of the workstation to another computer, and one to serve as a remote terminal port. These two ports would be in addition to the port used to connect the primary display station. Networking communication options would probably alleviate the need for networked workstations to have communications ports configured for these purposes.

Network Communication

Hardware options to provide good intercomputer communication--especially network communication--are essential. The more critical aspect of communication is the array of capabilities provided by the software. We have noted in previous sections that networking can replace many devices that are often high-priced workstation options. The failure rate and costs for maintenance might be high too. Network options to replace hardware are likely to be cost effective, especially in the longer term. In fact, there is activity in the computing industry to develop the networking standards which should result in lower network hardware prices.
OPERATING SYSTEM

Computing system software is often described in terms of layers. The lowest layer is the operating system, which itself is often divided into a kernel and other parts. The other layers build up a view of the computing system for the applications to rely on.

An operating system controls the bare computing hardware and provides four major capabilities on which other software depends. These capabilities are memory management, file system management, central processor management (scheduling), and peripheral processor (input/output) management. A distributed system may also integrate network management into the operating system.

The operating system is not used directly by a typical workstation user, but it provides a foundation for the workstation environment. Our experience during this project has been that the operating system upgrades have improved performance of the acquisition benchmark by a factor greater than two. One improvement seen was decreasing the time to start a process. Presumably this improvement involved the file system, memory management, and also the processor management. Tuning of the operating system to make better use of the hardware is an area in which new workstations will exhibit substantial change and eventual maturation.

The command (or control) language is one facet of the operating system that the user sees. This control language is the user's interface with the operating system that initiates such processes as the manipulation of files and the execution of images (processes). Modern operating systems have uncomplicated control languages that free the user from the necessity of becoming expert in matters not germane to the scientific goal. However, although the average user of a computing system might get by with the simplest commands, users must often comprehend and improve the interaction of their programs with the operating system to make programs more efficient or to use more sophisticated programming techniques. Changes to the control language for these improvements make even simple control languages appear complex.

---


5 For details of the benchmarking process, see R. C. Raffenetti, Scientific Workstation Comparisons Using a Benchmark, ANL/TM 415 (Argonne National Laboratory: Computing Services, April 1984).
SOFTWARE

We discuss here the tasks of scientific computing and the workstation application software that are used for each task. There are two major categorizations of the tasks of scientific computing:

1. Programming and the tasks associated with it require a specific set of tools. Programming is a general task dependent on the scope of the result to be achieved. There are small, simple programs as well as large, complex programs to be written. Each program has its own logic. Therefore, programming tools must be able to build a very general program.

2. The rest of the tasks of scientific computing can often be addressed with existing application tools; such tasks include the management and manipulation of the results of the scientific questions which are being pursued.

Besides the task-specific tools, there are also many general tools that apply to all scientific computing tasks. It is neither practical nor necessary to enumerate all of the tools in this report, so we restrict the discussion to the major ones.

The extent to which a scientific workstation supplies modern, capable tools for both programming and for the other tasks of scientific computing is a measure of the value of the equipment package. Useful tools should be of "industrial strength" and should not fail under the weight of real scientific applications. The importance of modern tools is that they conform better to modern programming practices, such as structured programming. That is, they incorporate the benefits of advances in software technology.

Text Editors

Text editors are certainly the most used tools in any interactive computing and job submittal environment. Editors are used to create and modify input data files and program source files, and to review output files. Early text editors were line-oriented, but the newer ones use the full terminal screen as a window into the contents of a text file. The editor software changes the file to agree with the changes that take place on the terminal screen.

For example, Unix systems have a variety of editors with differing capabilities. Perhaps the most useful one is vi/ex, which has two modes of usage, vi (visual or full terminal screen) and ex (line-by-line). The original line-by-line Unix editor is ed, and Unix also has a stream editor called sed. The latter is efficient for making simple global changes.
Programming Tools

The development of good programming tools in modern systems stems from the fact that the programming tool developers are also the consumers. Those most experienced in programming should be the best qualified people to design good programming tools. The following is a list of tools for building scientific programs:

- Program management
- Source code management
- Macro preprocessor
- Compiler
- Static analyzer
- Subroutine library
- Linker
- Run-time system
- Debugging aids

This list is quite generic with respect to the programming task. The Fortran language is almost universally used for scientific and engineering programming. See Appendix A for short descriptions of each of the programming tools listed above.

Tools for Management and Analysis of Scientific Data

Tools for management of scientific data permit the user to create and manage data without writing special programs for each new application. They permit the user to display the data in different ways depending on needs. Data is analyzed by doing further computations such as some form fitting or curve fitting. A "spreadsheet" is an example of a tool for management and analysis of scientific data that is usually associated with 8-bit or 16-bit personal desktop computers.

Graphics Tools

The display of graphics is a logical extension of the analysis of data where a picture aids the interpretation of results. It is useful to have some graphics capability in a workstation used for scientific computing. The quality need not be the highest, as long as there is a way to acquire high quality graphics by sending the data to another "networked" system. A workstation ought to have graphics application
tools to make quick displays of general data, as well as libraries of high-level graphics modules to use in programs.

Writing Tools

An important product of scientific and engineering research is written technical reports and other documentation. Writing tools for the preparation of scientific reports enable the workstation user to generate near-final documents and to include information easily from the user's data management system. The capability to enter text that includes mathematical formulas is valuable and is an important criterion of a scientific writing or word processing tool. If workstation users can enter these mathematical formulas themselves, they eliminate the possibility of another typist's typographical errors.

The Unix computing system includes various text formatting and typesetting tools, such as roff, nroff, and troff, which meet many of the needs of a scientific user. A system called "TEX", which is used on bigger computing systems, is also useful for scientific writing.
Chapter 5
THE EVALUATION ITSELF

To avoid reflecting possible biases among disciplines, the scientific workstation evaluation project involved scientific staff from various divisions of the Laboratory. Six participants agreed to do their work on the workstation and to write their experiences and impressions for this report. Chapters 7 through 11 are their contributions. One out of the original group dropped out because of his program's reliance on nonstandard Fortran features which the project workstation compiler does not have, namely the data input feature called "namelist". His withdrawal from the project points up the need for computer users to conform to standards to be better able to take advantage of new technology.

EVALUATION METRICS

Establishing the value of a particular workstation requires demonstrating its service in relation to the service of other available computing resources. The questions that we wish to answer are:

1. Does a user with a dedicated workstation produce more science and/or engineering for the Laboratory?

2. From the Laboratory's point of view, are the relative costs of the scientific output higher or lower than before use of the workstation?

In this section we describe some ways to gauge value in dollars.

Productivity Value

Insofar as productivity is concerned, two situations might occur. In one case, the workstation enables new work to be done that could not otherwise be done. For example, some special software is available on the workstation that is not available on other computers. Productivity is clearly improved. In the second case, the dedicated workstation environment makes the user more effective at getting the job completed than do other resources. Here, the degree of enhanced productivity is less tangible.
All of the tasks except production computing are likely to require many man hours. The usefulness of a workstation in accomplishing those tasks depends upon the extent to which its user has become more productive and/or more efficient. We feel that these new generation workstations have tools that permit users to realize net productivity gains. Note, however, that the six month duration of this project may have been too short a time to measure gains in productivity accurately. Some researchers claim that a twelve-to-fourteen month period may be necessary for new users to realize fully the expected gains.\footnote{E. Collett, "Tackling Design Environment Issues", \textit{Digital Design} (July 1984).}

To the extent that a net productivity gain can be established, the value of the gain can be quantified. For example, if a user becomes 10\% more productive, then the workstation has permitted the organization to realize the gain at no increase in manpower equivalents. The value of the workstation to the organization is 10\% of the salary and applicable overhead of one full-time person-equivalent. Note that in this project we have not determined the net productivity change. We don't know whether it is 1\% or 100\% or whether it is a gain or loss.

We emphasize "net productivity" here because there are tasks associated with a workstation that may have no equal in a shared computing environment. Backing up the workstation file system is one example of such a task. If a workstation generates many new tasks, then the net productivity change could be a loss. The chance of a loss can be minimized by choosing the system and its components wisely.

\textbf{Costs}

A simplistic way to figure the cost of a workstation might be to average the capital cost over the productive lifetime of the equipment (say four years), along with the annual maintenance fees, which are about one-tenth the purchase price. These costs add up to 35\% of the purchase price per year for four years. Supplies may add another one or two percent to the annual costs. There are also costs associated with the time taken by the owner or operator to maintain hardware and software or to arrange for that maintenance and possibly to do necessary programming that is not directly toward the scientific or engineering goals. The magnitude of these latter costs might depend strongly on the workstation capabilities and service record. The overt costs are largely fixed; there are no additional overt costs related to the amount of computing done in the workstation. In this project the maximum allowable workstation purchase price was fixed at $50,000. Therefore, if we disregard the manpower costs mentioned above, the yearly cost to operate the workstation is roughly $17,500. (Note that if it can be shown that productivity has not decreased, then the disregarding of the manpower costs is justifiable; if not, then these costs are significant.)
Another way to determine the value of a workstation is to place a value on the computing obtainable from it that is equivalent to the cost of comparable computing delivered from other computing resources. This argument is the most difficult to substantiate because of the dichotomy between costs and charges. Charges, which are the user's cost of computing, are not necessarily related to the costs incurred in providing the computing service (see below). Moreover, different arguments may be made (1) if you add one workstation to a large organization and (2) if you consider replacing the central computing service totally with dedicated workstations. In this project each user takes a position that is more like the first situation. If the concept were to proliferate, then the second situation might be closer to fact. We analyze cost here from the point of view of adding one workstation. In this model the computing service charges remain the same because the perturbation of the large system is negligible. The computing service will still be necessary to serve everyone else, and the amount of work removed to the workstation cannot affect the charges for the service. We first describe in more detail the dichotomy of charges and the costs of providing a service.

Computing service costs and charges are not equivalent for the following reasons:

1. Two types of costs are involved in a service. These are (1) fixed costs--the capital equipment costs and other costs associated with having a service available--and (2) variable costs--the costs that vary with the volume of business, or the cost associated with running one more job. In determining the average cost of a job, both fixed equipment lease and operating costs and variable costs are included at Argonne. However, in other organizations, variable costs are normally quoted when one part of that organization sells the computing service to another; the overall organization underwrites the fixed costs.

2. Joint costs arise when two services share some of the same resources (e.g., hardware, effort). The apportionment of joint costs to the several services they make possible is completely arbitrary.

3. Some costs that are not directly related to a specific computing service may be added as overhead expenses and may be charged out to service consumers.

For example, the Argonne central IBM computing services include not only machine cycles, but also operator availability and assistance, some data management, systems programming, documentation, and consulting. These services do not come with equipment. To the extent that users provide these services for themselves, no matter what computing equipment they use, they should include a value for their effort in their cost estimates.

However, because the major task of some workstation users is production computing, the comparison of the workstation cost
effectiveness for that task with users' costs for the other available production computing resources is important to our project. The amount of production computing that users can do on a mainframe depends on the workload, the scheduling, and the availability of funds to buy better handling (higher priority). Users consider the mainframe to have unbounded capacity to execute computing jobs unless there is more workload than capacity (i.e., unless the saturation point has been reached). The workstation capacity for production is less than that of a mainframe and might place an upper bound on the computing work done unless resources external to the workstation are used.

Consider the following analysis of relative costs in the light of our observations above. Suppose a benchmark showed that a workstation could produce results at one-tenth the rate of an IBM 3033. (Note that we have chosen the IBM 3033 as an example for comparison only to make the comparison concrete. The "one-tenth" value is hypothetical.) If that workstation produced 20 hours of work in one day, then that work might be equivalent to two hours of IBM 3033 computing. The current batch charge to run a one-half megabyte program for two hours on an IBM 3033 at Argonne at an average priority multiple of 0.5 is $179. Therefore, the workstation would be producing $35,800 worth of computing in a 200-day period in our hypothetical example using a factor of ten.

This analysis is, of course, simplistic. It does not take into account the uncharged costs we have identified. A more detailed analysis would also consider other factors that relate to the computing characteristics of each individual (e.g., specific programs, type of computing).

The evaluators in this scientific workstation project have attempted to determine such issues as the number of hours of useful IBM 3033-equivalent computing that a given workstation can produce in one day. Chapters 7-12 contain their conclusions.

EVALUATION DIFFICULTIES

The following difficulties occurred just because the evaluation was operated as a project. These should not be viewed as limitations of a workstation per se.

Sharing the Workstation

Because the evaluation included several users, there had to be some kind of physical sharing of the equipment. Evaluators were not able to separate themselves from their normal duties to work solely on the workstation for a long dedicated period (e.g., a week or longer). Therefore, we started a reservation mechanism by which the evaluators could make "appointments" to use the equipment at their convenience. In fact, for a good part of the evaluation period, the workstation was not
capable of providing for the computing needs of a single user. For example, severe run-time bugs originating in the Fortran compiler stopped some work until fixes were made. However, users could do editing and compile programs to eliminate syntax errors and could also solve link-time problems. If the workstation had been dedicated to one user, then the run-time errors would have been more critical and the user would have lost a large fraction of time. Users made appointments to access the workstation during the first five months of the evaluation period. Some workstation availability was lost because users reserved time and did not use it.

The "Central" Location

Another facet of the same issue is that the workstation was not located near any of the evaluators but was more or less "centrally" located in Building 221. This location was sometimes inconvenient for a user who suddenly found some free time to use an available workstation time slot. More important than the availability, though, is the fact that the workstation environment was different from the evaluators' own offices. Therefore, the only work to be done was that already on the workstation or other work which the evaluators might have brought along. In a user's own office, all of the materials and tasks to occupy the time between workstation interactions are available. For example, if a user submits directives to rebuild a program and knows that the process will take twenty minutes, then the user will naturally have available other work with which to occupy time profitably while the process is completing. The Workstation Evaluation and Demonstration Room was an unnatural surrounding for the scientific job although it was a comfortable environment.

Realization of the Concept

A difficulty arises because we had only one workstation to evaluate. This particular workstation embodied the workstation concept in the minds of its designers. It therefore had certain characteristic features. A great number of features are important in the realization of a "best" workstation, and no two evaluators have precisely the same point of view. The features of one workstation may be strengths to one evaluator but limitations to another. Our evaluation of the concept is thus limited by the realization of that concept to which we were exposed.

If a user were selecting a workstation solely for that user's work, selection of the best workstation would be relatively simple. The user would specify the requirements of the workstation and would select the workstation that best met those requirements.

Of course the potential buyer must be aware of the importance of various workstation features and capabilities. If important
requirements are treated too lightly in the acquisition process, then
the user may not obtain the "best" workstation for that user's project.
(In some products the necessary options may be added later; in others
the necessary options might never be available.)

We believe that the greatest common limitation of the project
system was the lack of a better file transfer communication option.
Unfortunately, while the machine chosen had the best performance
capabilities, it did not have adequate communication capabilities.

Conflicting Goals

The primary goal of the participants in this study was to complete
their science and engineering projects. The evaluation of the
workstation was a secondary goal that suffered to some extent because it
was secondary. Insofar as the participants could do the evaluation in
conjunction with their projects, the evaluation posed no difficulties.

RESERVING THE SYSTEM

There are two ways to give users access to a workstation which
accommodates one user at a time. One is to allow dedicated use by each
user for longer periods of time, and the other is to allow intermittent
use by each user. The first way is closer to the workstation concept of
having the equipment dedicated to a single user. The second way makes
the workstation more like a shared resource. The evaluators were not
able to devote themselves uninterruptedly to this project for long
periods of time (such as a month or even a week); thus exclusive
long-term use of the workstation by one evaluator at a time was
impractical. The project therefore began with an "appointment"
arrangement. We provided an appointment sheet for users to set aside
preferred time slots. During the normal 40 hour week, then, only about
six hours were available to each of the six users. Some time had to be
set aside for system administration tasks. Participants could use
unreserved time on an "as-available" basis.

We provide a rough idea of the workstation use in Table 1 below.
Since the system had no capability even for the simplest accounting, the
data in Table 1 comes from the appointment sheets that the users filled
in. These appointment sheets are somewhat inaccurate, because users did
not always keep their appointments, and sometimes they actually used
free time slots that they had not reserved. However, the statistics
still indicate the approximate use.

Table 1 presents data for the three consecutive four-week periods
beginning March 5, 1984, and ending May 25, 1984. It represents the
days Monday through Friday between the hours of 8:30 a.m. to 5:30 p.m.
only, each week consisting of 45 hours, and each four week period
consisting of 180 hours. These time periods cover the twelve weeks
The Evaluation Itself

...after the integration work and before the period of dedicated usage. The data consist of the hours reserved by each user and for system management activities (mainly backup) during each period. In this table the users, which are not identified, are ordered by each user's total hours reserved (the column on the right) over the entire twelve weeks. The table shows the total hours reserved for each four-week period as well as the numbers of available hours. The unreserved hours are the differences between the available and the reserved hours.

TABLE 1

Workstation Reservation Statistics

<table>
<thead>
<tr>
<th>User</th>
<th>Mar. 5 - Mar. 30</th>
<th>Apr. 2 - Apr. 27</th>
<th>Apr. 30 - May 25</th>
<th>12-Week Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>User A</td>
<td>22</td>
<td>4</td>
<td>46</td>
<td>72</td>
</tr>
<tr>
<td>User B</td>
<td>19</td>
<td>20</td>
<td>31</td>
<td>70</td>
</tr>
<tr>
<td>User C</td>
<td>20</td>
<td>10</td>
<td>11</td>
<td>41</td>
</tr>
<tr>
<td>User D</td>
<td>19</td>
<td>8</td>
<td>2</td>
<td>29</td>
</tr>
<tr>
<td>User E</td>
<td>2</td>
<td>9</td>
<td>0</td>
<td>11</td>
</tr>
<tr>
<td>System</td>
<td>27</td>
<td>17</td>
<td>15</td>
<td>59</td>
</tr>
<tr>
<td>Reserved</td>
<td>109</td>
<td>68</td>
<td>105</td>
<td>282</td>
</tr>
<tr>
<td>Available</td>
<td>180</td>
<td>180</td>
<td>180</td>
<td>540</td>
</tr>
<tr>
<td>Unreserved</td>
<td>71</td>
<td>112</td>
<td>75</td>
<td>258</td>
</tr>
</tbody>
</table>

The data in Table 1 prompts a few observations. First, there was a large disparity in the reserved time over the group of user participants. On the average, each of the five users could have reserved about 30 to 33 hours for each four week period. The maximum reserved time was 46 hours and the minimum was zero. Second, the reservation statistics show a very inconsistent pattern. Finally, the statistics show that 48% of the available time was not reserved.

After some time had passed, we determined that users were able to attempt one dedicated week of usage. At the end of the six-month project period, sufficient time was reserved so that each of the five remaining participants could work for an entire week at a time.
Chapter 6

PERSPECTIVES

In subsequent chapters each user/evaluator will describe his experiences. To avoid undue repetition in those chapters, this chapter provides the context for the evaluation process. This evaluation scenario is quite important; readers should not pass over the information.

THE RIDGE 32 COMPUTER

The Ridge 32 is only one of several possible scientific workstations. The procedure by which we selected it was a competitive procurement procedure. The competition served to ensure that we acquired the "best" equipment. The Request for Proposals (RFP), a document to which vendors responded with bids, listed a number of mandatory and desirable capabilities. Computing Services carried out benchmarking to determine the performance. Table 2 compares the performances of the Ridge 32 with some reference machines that are not workstation class machines but that are familiar to Argonne computer users. We measured performance as the elapsed time to do the work, which consisted of running programs in two processing streams (foreground and background) and printing some listings. We used elapsed time as a criterion because in a single-user system the only useful measure of getting the job done is how long it takes until the result is available. The cost associated with a workstation is the amount of time the user may have to wait for the result. If the time is too long, then the wait may affect the user's personal productivity. The benchmark performance was rated as 80% of the acquisition criterion as set down in the RFP. The Ridge 32 achieved the highest rating because its performance of the benchmark demonstration that we constructed7 was far better than those of the other competing systems. Compared to the other offerings, the Ridge system rating on the other desirable items was neither the best nor the worst.

---

TABLE 2

Total Elapsed Demonstration Times

<table>
<thead>
<tr>
<th>Computer System</th>
<th>Elapsed Time</th>
<th>Relative Times</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ridge 32 (Aug. 83)²</td>
<td>1:09:50</td>
<td>10.4 4.3 2.7</td>
</tr>
<tr>
<td>Ridge 32 (Mar. 84)³</td>
<td>43:33</td>
<td>6.5 2.7 1.7</td>
</tr>
<tr>
<td>Ridge 32 (Jun. 84)⁴</td>
<td>32:09</td>
<td>4.8 2.0 1.2</td>
</tr>
<tr>
<td>IBM 3033 (VM/CMS)</td>
<td>6:42</td>
<td>1.0 0.41 0.26</td>
</tr>
<tr>
<td>VAX-11/780 (VMS)</td>
<td>16:26</td>
<td>2.5 1.0 0.63</td>
</tr>
<tr>
<td>VAX-11/780 (Unix)</td>
<td>26:06</td>
<td>3.9 1.6 1.0</td>
</tr>
</tbody>
</table>

¹ Hours:minutes:seconds
² Original acquisition benchmarking
³ Acceptance testing: this time is 38% less than the original time.
⁴ Following the installation of ROS 3.1: this time is 54% less than the original time.

The performance of the Ridge 32 is a great advantage of the equipment. If all other characteristics were equal, then clearly the Ridge would be the best. But, characteristics are seldom equal, and indeed the Ridge 32 has limitations, some of which were known and some of which could not have been identified prior to acquisition. It is probable, too, that the other workstations that have been conceived, designed, and built also had and still do have some of the same limitations and even some others. The point of this discussion is that once you choose a piece of equipment, for a project of the present nature or for your own purposes, you will have to endure the limitations of that equipment. Missing features and capabilities that you have taken for granted will take on new importance when you have to work around them. In this project the absence of standard features and capabilities might be especially damaging because unless readers are careful, they will associate the limitations with the workstation concept rather than with the particular workstation.

It is probable that existing workstations have and will continue to have limitations associated with their implementations. We associate the limitations with the process of maturation. In a computer system, the software and the hardware must work well together, but each of these may have been developed separately. Therefore, an iterative process of refinement takes place once a working computer system is established. It would be unreasonable to expect that the refinement has taken place in a new product. The refinement will take place and the resulting system maturity that users expect from their equipment will be achieved. It is probable, but not certain, that workstations that have been "in the field" for a longer time will be more mature. It is also virtually certain that workstations which are merely "announced" will not be
mature at all. (Note that in the jargon of the computer trade, products are often "announced" long before they are designed or built. Some announced products never get built because the "announcement" does not generate enough interest from consumers.) Maturity comes as users and customers demand and receive the features that they need. A following section traces the changes that we have seen in the Ridge 32 workstation throughout the project period.

The entire workstation package consisted of a system unit, a display, a detached keyboard, and a printer. The system unit was a large package containing the computer along with a 60 megabyte hard disk, two megabytes of memory, and an eight-inch, double-sided, double-density floppy disk drive. The floppy disks each had a capacity of about one megabyte. For communication and device attachment, there were four asynchronous terminal ports, a printer/plotter port, and a port for the display. The display was a high resolution, bit-mapped graphics display. The keyboard had a "QWERTY" layout and had cursor keys. There were also program function keys which, to the best of our knowledge, were not used by any utilities. The dot-matrix printer supplied with the system was a Genicom 3404 which is rated at 400 characters per second and can be programmed for many font sizes and paper sizes.

The Ridge System: Strengths and Limitations

The Ridge system has a variety of strengths and limitations with respect to the scientific workstation model. For any one individual's work, the Ridge itself could be more than one needs or less than one needs or both. The following sections elaborate on our observations.

Processing Speed

Apart from other considerations, the execution performance of the Ridge for single processes that do not do much paging or other input/output is very good for computing systems in the Ridge price class. For double precision linear algebra computations, the Ridge central processing unit (CPU) times are close to those of a VAX-11/780 with the Unix operating system from Berkeley or of a VAX-11/780 with VMS, the Digital Equipment Corporation (DEC) operating system. Where we cite CPU times, we point out that CPU time comparisons can be very misleading because many useful programs do not have the "memory bound" central processing behavior of the programs used to determine CPU time.

---

The Scientific Workstation Evaluation Project

comparisons. Scientific programs require large memory size, and when memory is virtual, the paging delays may be significant. Input/output is always needed and may also cause substantial delays.

Software

The Ridge 32 system includes the Unix operating and programming environment. The software tools included are both modern and capable and afford the scientific workstation user some powerful aids for programming and other scientific computing tasks. The command language and command environment (the shell) permit combining tools to do complicated tasks without writing traditional programs. There is also a large library of useful subroutines which can be called from Fortran programs. In the Ridge system, there are over 200 command-level programs and over 125 Fortran callable library programs.

Backup

In the workstation acquisition, we did not emphasize the capability to do fast, convenient backups of system or user files. This lack of emphasis was a mistake in this project; in the future it would be a mistake not to provide for backup in any significant computing resource. Hardware provides the capability, and software gives convenience. The Ridge computer system included a one megabyte floppy disk drive, which we used for backup. No special software was provided. We provided a backup service for users: the Fortran source files were backed up on a weekly basis, and modified Fortran source files were backed up on a daily basis. The procedure was formulated as a Unix shell script (a sequence of commands which may be executed from a file). Because of the slowness of the operating system in executing shell scripts, the backup process was impractical until we made the script more efficient. Even then, one weekly backup required one hour and thirteen minutes to copy 595 files to eight floppy diskettes. We believe that a more expert user of Unix could have economized the task time somewhat, but the inconvenience of handling the floppy disks would still exist.

In the case of the Ridge system, it would have been useful to have the manufacturer supply a backup procedure that would better use the hardware. It would have been still better to have obtained real backup equipment for the workstation.

In this project the system administrator performed the backup on behalf of the users, and the users were cautioned that only limited backups were being done (only Fortran source files). If a workstation were operated by a single user, then the responsibility of backing up and restoring files would be clear; that user would do the backup or arrange to have it done. The assurance that recovery is possible comes from doing regular backups. Regular backups are more likely to happen if the equipment is fast and the procedures are convenient.
Communication

A mandatory requirement of the workstation was that it provide for communication to the IBM computers at Argonne. We knew from our experience with VAX to IBM communication that file transfer is an essential capability. We knew that computing systems with the performance characteristics that we could expect to see in a scientific workstation would need very good communications.

We believe very strongly that good, error-free communication of files is essential in any significant computing resource. No workstation should be purchased without a plan for establishing good communication at the outset. If the initial communication is not networking, a plan should exist for ultimate upgrading to permit network access to other computing resources.

Two communication capabilities were established between the Ridge system and the IBM systems. One consisted of custom programs to send or receive files to and from IBM VM/CMS or Wylbur sessions that were started with a virtual terminal program. This method did not do error checking. A second method was to use Kermit, a public-domain, terminal emulation and file transfer program that was implemented on the Ridge system by the manufacturer. The Kermit program does error checking but only works with interactive systems like VM/CMS. It does not work with Wylbur, which is merely an interactive editing environment.

In either case, the terminal sessions were established over a 9600 baud shared X.25 circuit into the IBM 3705 telecommunications front end. The local connection (all in the same building) was relatively error-free, so the lack of error checking for Wylbur to Ridge file transfers was never found to have introduced unrecognizable errors. The effective rate of file transfer from Wylbur to the Ridge was less than 3000 baud, and in the other direction it was not as good. Because most of the participants do not use VM/CMS, there was not much use of the Kermit program. The Kermit error checking operation slowed the transfer rate.

The Operating Environment--Unix

Unix has become an important operating system in the 32-bit workstation market. There is a rush to bring new hardware to market, and the promise of the Unix system is that it is easily portable to new hardware. Portability was a goal of Unix design and implementation.

---

Indeed, there is a demand in the marketplace to have Unix, rather than other systems, as a operating system and environment. There is a range of opinions about Unix, from its being the best to its being the worst. The Ridge 32 in this project used ROS, the Ridge Operating System, which is an implementation of Unix System V and which has extensions from the Berkeley 4.2 version of Unix.

Process Management

The Ridge computer system is marketed as a multi-user system that can serve one to five (or so) users. Insofar as the Ridge system is used by one user, the work can be managed easily and the system performs well. The user manages his own processes; the process identification number is displayed when a background process is initiated. Managing one's own computing tasks is not particularly convenient, but it is possible. Assignment of priorities ensures that those tasks the user considers important are worked on before those which are less important. During most of this project, these features were missing and/or inoperative.

Although we never set out to study multiple users, we did note that the primary difficulty with multiple users was having to manage a large number of priority levels that exist for processes while not being able to determine what other users' processes were doing. Rules were necessary, so that one user would not dominate the resource when another was entitled to a share. Such rules are hard to enforce; therefore, it would be better if the system did the management job well.

WORKSTATION HISTORY

The Ridge 32 has changed considerably (mostly improved) since the day we first saw one. We include this section to give readers a perspective of those changes, so that they might set expectations. First, as is the case with the project workstation vendor, many vendor companies are small and have limited resources. Because of this similarity among the vendors, you will probably find that the changes made in the Ridge workstation will be similar to the changes other companies will make in their products. Second, this section provides the physical scenario for the evaluation.

We will limit our discussion to the past and will refrain from attempting to describe the future. It is our experience that when the vendor makes a planned system change, the change does not actually include all of the features that have been mentioned. This fact should not necessarily create a negative impression, because the equipment owner may often hear of unscheduled changes. The decision process that dictates which of the changes are to be made and when they are to be done is complex. A long list of special interest groups, including the current owner community, the possible customers, and the technical expertise and interests of the developers, influence the process.
We have seen both hardware and software changes during the lifetime of the project. Each major change that brought significant improvements and new features also resulted in some existing features being "broken". Newly broken features can be more damaging to a user's work than the advent of new features can be helpful. For example, we had no special hardware and software to do complete backup of the system and/or user software. It was therefore impossible to fall back to the earlier release of the operating system when major broken features prevented work from being done. This situation posed more of a problem than if the hardware were broken. Hardware can be replaced easily, but repair is the only alternative for software with no replacement option (no backup). The process of repair is inherently more problematic than straightforward replacement and can therefore lead to longer delays.

Table 3 contains the more important dates of the evaluation project. The Ridge workstation was originally scheduled for delivery in December of 1983. Because of the long continuous holiday period at the end of the year, we agreed with the company to take delivery at the start of 1984 (the machine having been shipped in 1983). Argonne received the Ridge workstation on January 5. On January 9, a representative from the California offices of the company arrived to uncrate and install the equipment. The operating system (ROS 3.0) had been loaded onto the disk, but some software was patched, and programmable read-only memory modules (PROMS) were changed. The system started without incident. At about the same time the company hired its first sales representative for a Chicago-area sales and service office. On January 16, the Ridge was available for the project participants to reserve time and start learning the system, although a good deal of the
time was being reserved for system administration and other integration activities. By February 6, most of the integration activity had been completed, and most of the time was now available for users to reserve. On March 5, the system was put through the acceptance tests, which consisted of running the original acquisition benchmark demonstration. The workstation was relocated from an ordinary office environment to the Electronic Office Workstation Evaluation and Demonstration Room (A-142) in Building 221 on April 2. On April 16, the company hired its first systems engineer for the Chicago-area office. Prior to this move, the California office had supplied the technical support. On May 30, ROS 3.1, a new version of the operating system, was installed. The periods of dedicated usage began on June 4, and the project period formally ended on July 8. Due to the delay in delivery and installation of the communication software, the Ridge workstation was also available to the participants through most of July and August. On August 27, the Materials Science and Technology Division purchased the Ridge 32 workstation and had it moved to Building 223.

WORKSTATION EVOLUTION

The Ridge computing system evolved considerably during the project. The evolution consisted mainly of the inclusion of Unix operating system features that had not been implemented in the earlier versions and of other changes that improved existing capabilities.

The acquisition benchmark demonstration is an indication of the overall performance of the system in doing a certain unit of multiprocessing. The acquisition benchmark demonstration performance (see Table 2) improved dramatically. There is a difference greater than a factor of two between the demonstration runs of August 1983 and June 1984. We also noticed changes and improvements in some of the independent system functions that affected overall performance.

It is not essential, nor was it necessarily useful, to the project to do detailed analyses of each element of the system performance. However, we did note these elements whenever they were evident. Our benchmark demonstration also produced some central processor times for each of the demonstration programs or tasks. We were therefore able to compare them between runs. By and large, the dramatic improvements in overall benchmark performance did not come from improvements in processor times. The elapsed times of tasks decreased considerably. We assume that much of the other processing associated with a task was economized, including the I/O delays, the paging delays, the swapping delays, and the task initiation and termination processing. The next section details some of the specific changes and the expected areas of improvement.

---

10 R. C. Raffenetti, Scientific Workstation Comparisons Using a Benchmark.
Software Change History

This section presents a brief history of the software difficulties and changes. Whenever a change occurred, most of the previous difficulties got fixed, but generally some other difficulties appeared. From this history, you should become aware of some of the situations that the use of immature software can cause.

January 11, 1984: Installation of the Ridge with ROS 3.0

After the initial installation of the workstation equipment, we encountered the following difficulties and software bugs:

1. Divide by zero halts programs with no indication of an error.
2. The existence of background or "child" processes prevents logouts until they complete.
3. The vendor-supplied method for communication to the IBM machines is not operational.
4. Compiler messages cannot be redirected to a file instead of to the display.
5. Interactive response with multiple users is very bad.
6. Out-of-range constants cause the Fortran compiler to hang.

These difficulties did not prevent users from trying to convert their programs, but they did make progress very slow. Bug determination was difficult.

February 29, 1984: Patches and Prerelease Software Installed

The manager of Ridge software development came to Argonne to fix some of the difficulties and to get the communication to the IBM systems going. The command `nohup` was added, which enabled users to start background processes that could be left running after logoff. The following difficulties appeared after the changes:

1. Modules were missing from the Fortran library for the exponential function (i.e., `exp(x)`).
2. The capability to dial-in from a remote terminal now fails.
March 7, 1984: Fixes Installed

A system engineer from the Boston office of the company installed some "fixes" while at Argonne in conjunction with a talk by the company president. Difficulties following the changes included:

1. The compiler now goes into a loop when a value in a parameter statement is out-of-range.

2. We are still unable to set the priority of background processes.

Interactive foreground work was slowed considerably by the processing of background work.

May 30, 1984: Installation of ROS 3.1

In addition to our receiving the new software, the company reformatted the disk for efficiency. The nice command was added, which made it possible to set the priority of a background process. The following new difficulties appeared:

1. Arithmetic traps, including underflows and overflows, are now being handled differently (and incorrectly).

2. The linker cannot find some Fortran-callable routines.

3. Our custom software for file transfer between Wylbur and the Ridge is not working.

4. The linker is causing separate data areas to be superimposed when an option needed for big programs is used.

5. File I/O operations are not correct when rewind is done.

Subsequent patches and changes fixed these last difficulties. The new version of the operating system came at a very bad time for the project. While it brought some new and important features from which the project would benefit, the loss of features that had formerly worked caused disruption. The timing was bad because the project was just entering the periods of dedicated usage.

Hardware Changes

The Ridge hardware performed well during the project. The company quickly attended to the three failures that did occur:

1. The first printer was replaced by the company because of poor printing. The replacement printer failed and was exchanged for yet another unit.
2. A Ridge system engineer noticed that occasionally the hard disk would stop. The inconsistency prevented us from finding out what caused the behavior and how long it might have been happening. The disk was replaced by the company, and the file system was copied from the old disk to the new one.

3. On another occasion the system failed to boot. This was the only failure that caused the project to lose effort or time. The company replaced one of the system boards in less than a day.
Chapter 7

SAS4A, A PROGRAM FOR REACTOR ACCIDENT ANALYSIS

Goran Birgersson
Reactor Analysis and Safety Division

CHARACTERIZATION OF EVALUATOR

My work includes supervision of SAS4A, a liquid-metal fast breeder reactor accident analysis program. The code is large (approximately 100,000 lines), has a significant number of subroutines (approximately 300), and uses typically two megabytes of memory on the IBM 3033, but larger cases can be run. Production runs take from fifteen minutes up to more than one hour of CPU time on the IBM 3033. SAS4A solves a set of partial hyperbolic differential equations which describe the reactor fuel behavior during a hypothetical reactor accident. The modeling of fuel behavior is a complex task, and suggested code changes need to be extensively tested before being incorporated into the SAS4A production version. The code advances the solution over as many as 1000 "main time steps", each of which may be subdivided into hundreds of "elementary" time steps. Traditionally, the solution has been exhibited with main "edits" (i.e. formatted displays of system state parameters) at selected time steps and detailed edits depending on the feature studied. This program produces large amounts of output for a single run (49,000 to 99,000 lines). The fact that only a few time-dependent quantities have been plotted is a definite shortcoming.

Codes of this kind can produce a virtually unlimited amount of information. The problem is how to handle it.

I spend about 60% of my time debugging the code and maintaining the source file and the corresponding load module library. Other members of the Reactor Analysis and Safety Division do the development work. My remaining time is evenly divided between programming and production computing tasks.

Debugging on a mainframe like the IBM 3033 involves bugs that fall into either "simple" or "hard" cases. Examples of simple cases are "division by zero" and "square root of a negative number". These get

---

flagged by the computer and cause a trace-back to be produced. Together with the information given by the compiler, these tracebacks usually make it easy to find the offending quantity. Sometimes these error messages will indicate an input error or a simple oversight in the programming, but often they are the consequences of a "hard bug". In the latter case, the code has derailed, but only slowly, because of something usually far removed from the first indication of an error. My method for dealing with these bugs is a painstaking one--to follow the computation in reverse (that is, to find out the numbers that made up the "offender", then to pick out a "suspect" among these, then to find the numbers it was made from, and so on). The process usually ends inside one week, but by then it has produced six feet of printed output and taken me on a grand tour throughout the entire code.

All my runs have been done on the IBM 3033 with Wylbur to edit the source and data files.

PRECONCEPTIONS AND EXPECTATIONS

The first goal was to establish whether SAS4A could run on a Scientific workstation. The second goal was to establish whether the scientific workstation would be an adequate tool for work on the SAS4A code. Could the debugging runs, which normally are short (only two to three time steps) but produce lots of output, be run on the workstation, and would it be possible to use the workstation as a computer dedicated to run SAS4A?

In double precision, the Ridge 32 uses eleven bits for the exponent and 53 bits for the mantissa (the 53rd bit is always one and is not retained). In a test program, I found that the Ridge loses about one decimal digit, in comparison with the IBM 3033. The test problem involves inversion of an ill-conditioned six-by-six matrix for which the exact result is known. This behavior under these conditions suggested a mechanism for testing the robustness of the algorithms employed in SAS4A, simply to monitor where and if the solutions to the same problem run on the two computers diverge.

As a side benefit, I expected to discover and be able to remove any bugs in the code that had survived only because the IBM environment had rendered them harmless. For example, I consider it good practice always to initialize variables to zero, even if the loader will do that for you. This practice would increase portability of the code and detect local system dependencies. Some compilers have better diagnostics than others. The IBM compilers do not advise against the use of undefined variables, for instance.

I expected that the evaluation would prove that the scientific workstation could be an efficient workhorse in my office and could also improve the quality of the SAS4A code.
THE WORKSTATION AS A PRODUCTION ENVIRONMENT

Conversion of SAS4A to the Ridge

The first task facing me was the transfer of the code from the IBM system to the Ridge. We transferred the code in chunks of 15,000 to 20,000 lines of coding. This procedure worked well, and we had an estimated transfer rate of 2000 baud, although the line had 9600 baud capacity. The transfer of source and input files took a few hours. The next step was to compile the source, which took between three and four hours. The SAS4A routines use a large number of COMMON variables, and a special compiler option was needed. We made a few corrections to the code at this point. We included "OPEN-statements" for all files in the MAIN routine, and we converted "REAL FUNCTION *8 FK (T)" to "DOUBLE PRECISION FUNCTION FK (T)."

The next step was to create an executable load-module. Because of the large number of subroutines, the normal "load" command failed and indicated that most of the routines were absent. In the load command, I used "*.o" to indicate "all which end in .o". The result was that an internal buffer overflowed and probably destroyed vital information. I circumvented this result by forming 40 "partial" load modules of eight routines each and loading these forty to form one executable load module. The whole loading process took twenty-five minutes. The execution took ninety seconds to start up but failed almost immediately.

The error indication was of a most disturbing and strange nature. After printing more than 2000 lines of floating point output, the I/O-package suddenly was unable to print floating-point values using D- or E-conversion. Only after a very long search that almost exhausted all computer time available to me did I discover that the Fortran compiler or possibly the loader had given the wrong address to the block of constants and internal variables generated by the compiler in one specific routine. As soon as this routine was entered, the I/O-package went sour. In all fairness, I must mention that the Ridge documentation warned against compiler optimization and I had used it! My remedy was to place all local variables in a newly created "named common". After this change, SAS4A ran on the workstation.

Capacity

I ran our largest standard test problem, the first part of which takes thirty-six minutes of CPU time on the IBM 3033. I did not want to print the complete output on the line printer, so I routed the output to the screen and saved only the most revealing data on disk, a few numbers per time step. The execution of this test took between 19 and 19 1/2 hours. Here, a factor of 32 exists between elapsed time on the Ridge and CPU time on the IBM 3033. This test, which uses 2800K bytes on the IBM, caused SAS4A to do a lot of paging. It has been argued that displaying the results on the screen slowed down the execution as well. I did not have an opportunity to verify this assertion.
Communications

I have already indicated that the effective transfer rate from the central IBM computers to the workstation was on the order of 2000 baud. The transfer rate of files from the Ridge to Wylbur on the IBM computers was a lot worse, and during the transfer, my Wylbur account was unavailable. That is, I could not start a Ridge-to-IBM transfer, walk away, and resume normal Wylbur activity on the IBM. Hence, I could only transfer data at night or when I had no need to access the IBM system through Wylbur. The Ridge system as it was set up was a complete computing system. A powerful communications capability is needed if the Ridge is to be made an integral part of a computing environment.

THE WORKSTATION AS A PROGRAMMING ENVIRONMENT

The idea of using a workstation for program development is a very appealing one. An almost instantaneous turnaround time would allow me to concentrate on one task at a time and finish it before attacking the next one. In a computing environment with slow turnaround, it is necessary to keep several projects going at the same time. Invariably, one will lag. Given permanent access to a workstation, I probably would do all my development work, short of SAS4A, on it. Debugging tools are of great value and my favorite is Watfiv (which was not available on the Ridge 32). Unfortunately, Watfiv cannot execute SAS4A. The workstation debugging tools are inadequate for the difficulties I encountered.

THE WORKSTATION AS A DEDICATED RESOURCE

The workstation has the potential to serve as a dedicated resource for short SAS4A runs and could be used to track down the "hard" bugs I referred to earlier, but first some modifications to the system must be made. First, a more powerful printer is necessary. Second, the turnaround time must decrease for a jobstream that consists of inserting changes in a few subroutines, compiling these subroutines, and creating and running a new executable module. In my attempts to make SAS4A run, this sequence took close to thirty minutes. Improvement in the loader might shorten this time.

A scheduling system which could place jobs with low priority in limbo and allow jobs with higher priority to start and run until completion, after which the previous job would start up again would be desirable. In this way, long-running jobs could use all the "spare" CPU cycles in between tasks. (This feature appeared with ROS 3.1.)
CONCLUSIONS AND REFLECTIONS

My conclusions are as follows:

• The SAS4A program can run on a Ridge scientific workstation.

• Because of immature software and an unfortunate combination of problems that plagued the conversion of SAS4A, I could not form a definite opinion on the use of the workstation as a vehicle for debugging. I tend to regard the workstation as being too slow (the turnaround takes too long) for this type of task because the IBM 3033 seems to deliver the answers just as quickly.

• The workstation might be useful as an inexpensive number cruncher, if an adequate printer and a working priority scheme were installed.

• I had no chance at all to judge the robustness of the algorithms of SAS4A because only one real run was made, and too little information was saved.

• In the light of hindsight, I can maintain that SAS4A was in excellent condition, and that the conversion to the Ridge should have caused no problems at all.

• My final conclusion is that the Ridge 32 scientific workstation as I tried it is not adequate for a code that requires such intensive use of so many computing resources as SAS4A does.
Chapter 8

VIM, A MONTE CARLO REACTOR NEUTRONICS PROGRAM

Roger N. Blomquist
Applied Physics

BACKGROUND

My work on Argonne's IBM computers consists primarily of development and maintenance of VIM, a large general purpose neutron or photon transport Monte Carlo code, and the development of relevant new computational methods. Most of the requisite tasks are source code editing, debugging, and testing using problems which require several minutes of CPU time. In addition, I improve, test, and maintain the large (nine megabytes) nuclear reaction database which VIM uses. The source code for VIM and its auxiliary programs, including Applied Physics Division utilities for dynamic memory allocation, consists of 27,000 lines of Fortran coding, including 170 subroutines, and requires one megabyte of storage. The compiled subroutines and functions occupy 500 kilobytes and the load modules need a total of one megabyte. VIM typically accesses eight to eleven datasets, one of which can be very large (tens of megabytes). Four are accessed every minute or so during a VIM run. The auxiliary codes can access over 100 datasets in one calculation, although by careful coding one can reduce this number to about twenty open at any single moment. Although originally written only for Argonne's IBM computers, VIM was modified for export to CDC computers and to other IBM installations several years ago, although not in Fortran 77. It was already running at several other laboratories. Before and during the initial phase of this project, I modified VIM according to the Fortran 77 standard.

In addition to programming and methods work, I also run VIM to perform benchmark and design reactor physics and shielding problems. Each of these calculations typically requires several hours of CPU time and 1000 to 3000 kilobytes of memory on the IBM 3033, as well as substantial input preparation.

---

OBJECTIVES

Overall Goal

The overall goal was to demonstrate whether or not a scientific workstation is sufficiently powerful and convenient. Is it an attractive alternative to Wylbur and the batch computing environment on the IBM 3033 computers for someone who develops and uses a large production Monte Carlo radiation transport code?

Subgoals

Is it possible to convert VIM and its auxiliaries for application on the Ridge? The ease of conversion depends primarily on the degree of Fortran standardization in VIM and in the workstation compiler, and the convenience of system utilities and special functions used for dynamic memory allocation and timing information.

Is it possible and convenient to perform production Monte Carlo calculations on the Ridge with VIM? Is the Ridge fast enough to complete overnight (in 16 to 24 hours) the amount of simulation I can expect to perform overnight (say in three hours of CPU time) on the IBM 3033? Is the mass storage sufficient for the data used and generated during typically complex calculations? Is data transfer to and from the IBM computers feasible?

Does the Ridge provide software resources sufficient for further development and maintenance of VIM? Are the compilers and the linkage editor fast enough, are the diagnostics and execution error messages informative, and are the debugging capabilities useful?

Can I perform major Monte Carlo calculations while programming, testing, and debugging new methods or capabilities of VIM on the Ridge? Can the Ridge support two versions of VIM? Is there excessive foreground/background interference?

PRECONCEPTIONS AND EXPECTATIONS

From information promulgated informally, I expected that the Ridge might not be fast enough for production computing. I was also concerned that mass storage would be insufficient. From my experience on a microcomputer, I felt that the editors and other such software would be adequate. I knew nothing about Unix at the beginning of the project.
CONVERSION OF VIM FOR THE WORKSTATION

The first part of the conversion process involved merely compiling each Fortran subroutine, which required roughly a week of modification and compilation. Since the Fortran 77 version of VIM had been compiled only on the IBM 3033, there were some residual Fortran errors. Roughly a hundred other Fortran errors of several types were due to our use of IBM's non-standard Fortran 77 extensions in the code (e.g., the z-format for hexadecimal output representation). A bug in the compiler/linkage editor that precluded routing Fortran diagnostics to either the printer or to a file for later viewing complicated the process, until the vendor resolved the problem a week later. The Fortran syntax diagnostics were informative and conveniently referred me to the Fortran source line (card-image) number consistent with the editor's line numbering.

The second part of the conversion process consisted of substituting functions on the Ridge for IBM system function macros and for special Fortran functions used in the Applied Physics Division's dynamic memory allocation package (BPOINTER), in a general purpose timing routine (TIMER), and in bit manipulation. The Ridge 32, as initially delivered, included functions callable from only one or two of three languages (Fortran, PASCAL, and C). The more sophisticated random number generator, the more versatile timing functions, and the memory allocation functions were not callable directly from Fortran. One could, however, call C-callable functions from an intermediate function written in C and named so as to be recognizable to a Fortran program. Coding these transition functions becomes more complicated when sequence rules are called that differ between Fortran and C. For example, when Fortran calls a function, it passes the addresses of the arguments rather than the values. In contrast, C sometimes receives values rather than addresses. To overcome the lack of Fortran-callable functions, one must have access to a C language expert or learn at least some of the language. Unfortunately, it is not immediately clear to the novice exactly how much or what parts of C he must learn. I did not complete this coding except in the few instances where it was absolutely necessary.

The third part of conversion involved testing the converted codes. Although I easily performed a very simple one-energy group, infinite medium calculation that bypassed much of the coding, VIM could not complete the third stage of processing (the statistical analysis and the final edits). I also was able to complete a one-group reactor subassembly calculation, but a more typical complicated calculation with cross section data failed. In the project time allotted, I was unable to determine if these difficulties resulted from input errors, code bugs, or system problems. More details appear in a later section. I spent about two months in the debugging stage.

Ridge provided extensive documentation that was generally useful. Initially, however, there was no cross-reference index, a severe problem for the novice. The man command lists pages of the manual for the user, if the user knows the name of the Unix command or other function in which he is interested. A good cross-reference index would be an enormous enhancement to both the interactive and bound manuals.
A Ridge representative introduced the project participants to the workstation at a training and practice session. He explained how to use the floppy disk drive, how a user can unintentionally delete or clobber his files, and what to do when the system becomes "hung" or a process is in an infinite loop. He also reviewed for us a few of the more useful Unix commands. Additionally, the project coordinator conducted several training sessions on very powerful Unix capabilities--pattern matching, using the editors, and controlling the compilation and linking of large programs after source code modification. The sharing among the participants of information, techniques, and applications in the Unix environment was very useful, as it is on any computer system.

THE WORKSTATION AS A PRODUCTION ENVIRONMENT

The Ridge completed a simulation of a one-group reactor subassembly test problem in 120 seconds with no use of compiler optimization. The identical calculation on the IBM 3033 consumed eleven CPU seconds. I was not able to perform similar comparisons with a more realistic, multi-energy, multi-region problem. The Ridge might compare better with calculations that would be more processor-intensive. Any speedup in the Ridge's I/O would also probably improve its comparative performance. In any case, it seems that one could perform about the same amount of production computing (say two to three hours worth of CPU processing) in a 24-hour day that one can complete overnight on the IBM computers. Currently the IBM batch systems are unsaturated and provide quick turnaround for short jobs. For example, we are able to test an input deck in five minutes without asking for high priority, high cost handling. Therefore, there is no incentive to run on the Ridge.

Along with computing speed, the prospective workstation buyer in the Applied Physics Division must be concerned about data transfer. The division develops, maintains, and heavily uses several large codes and code systems that are shared by a dozen or so users and several code developers. The code systems include large data files of up to eight megabytes and a number of large load modules to which each user must have ready access. If data transfer is slow, users will be strongly motivated to keep personal copies of data files on each workstation; the obvious impact on mass storage requirements would be unfortunate. Since these code systems files are modified and updated frequently to improve capabilities and eliminate residual coding errors, users with personal copies on each workstation would need to modify, recompile, and relink periodically. They would have to be aware of how current their code versions were. This proliferation of source and data files would also cause unseen mass storage costs.

In the case of VIM, the library of cross section files (about nine megabytes) would require about five days of transfer time from the Wylbur system at the current rate of about three 80-character records per second. These files would, of course, be stored on floppy disks at the workstation once they were imported. The volume of printed output produced by VIM and the other Applied Physics codes can be so enormous
that some users regularly route it to microfiche. It is difficult to imagine using VIM without an output device substantially faster than the current workstation printer, unless the workstation could send output to a shared print device quickly.

TABLE 4
The Four Largest VIM Users

<table>
<thead>
<tr>
<th>User</th>
<th>Monthly CPU Hours</th>
<th>Annual Charges</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>9</td>
<td>$10344</td>
</tr>
<tr>
<td>B</td>
<td>6</td>
<td>$4584</td>
</tr>
<tr>
<td>C</td>
<td>5</td>
<td>$3530</td>
</tr>
<tr>
<td>D</td>
<td>4</td>
<td>$5664</td>
</tr>
<tr>
<td>Totals</td>
<td>24</td>
<td>$24132</td>
</tr>
</tbody>
</table>

In addition to the feasibility questions for a production workstation discussed above, the comparative costs must be assessed. A look at some of the VIM usage data will help us to determine whether or not the Ridge might save Applied Physics Division enough computing funds to justify its purchase. Table 4 displays some data for the biggest users of VIM in the Applied Physics Division. We show the average monthly CPU hours determined from nine months of usage data. The annual charges include both CPU and CPU-memory charges for this work, which was done at Argonne (these figures do not include VIM use at Argonne West or VIM use charged to other divisions). For example, suppose we bought a Ridge 32 for $60,000, including five years worth of maintenance contracts, and that this price represented the total cost of the system (which, of course, it would not). From the annual charges in Table 4, user A would require six years of exclusive use of the Ridge to break even. Since user A would require an average of only three hours per day of CPU time on the Ridge, he could share it as a production facility with other VIM users B, C, and D. Such shared use would reduce the payback period to less than 2.5 years while consuming eight hours per day of CPU time. The breakeven point would be, at worst, marginal. Of course this example does not take into account other costs that we would have to consider for a totally fair comparison. These costs would include not only the expense of maintaining multiple code versions, communications systems, and other peripherals but also the effort costs for operations, systems programming, user education, etc.
The programming work I performed on the workstation was limited to production code conversion. I wrote several small programs to test Fortran functions that were different from their counterparts in the IBM function libraries, namely bit arithmetic, LOCF (which returns the address of a variable), timing functions, and a random number generator. Most of the programming work, however, was debugging the version of VIM brought to the Ridge.

A great deal of the debugging process consisted of making corrections or inserting output statements in several subroutines for one run. A difficulty with adding output statements to a program for debugging was the inability to reblock the output to get the last printed lines. The make utility was exceedingly useful, since it totally automated what on IBM systems is the linkage editor input for a modified code. The vi, ex, and ed editing combination was superior to Wylbur in flexibility and power. A persistent aggravation for one who has long used Wylbur, however, was the case sensitivity of the vi and Unix commands. The turnaround for compiling and linking was marginal in that these operations were not fast enough to provide a big advantage over the IBM computer service. The Ridge took 1.5 minutes to sort through the source and object files to determine what recompilations and relinkings were required, about one minute for each subroutine recompilation, and four minutes to relink VIM. This turnaround is comparable to the turnaround available from the batch IBM systems.

The big shortcoming of the Ridge when compared to Argonne's IBM systems is its inadequate debugging tools. In the IBM environment, we obtain a great deal of information from execution error messages, tracebacks, and from core dumps, especially with the VS Fortran compiler. From the tracebacks, we usually can find the range of Fortran statements in which the error manifested itself, and we then can look up the value of any suspect variables in the dump. This procedure can be somewhat cumbersome, but it is very powerful. When interrupts occurred on the Ridge, we did not always receive notice of what kind of error occurred, and tracebacks were not provided. When errors invoked Ridge's "DEBUG" system, the interrupt address was readily available and could be compared to a linkage editor cross reference map to find the subroutine where the error condition occurred and the address within that subroutine. That information converted to a Fortran line number only through comparison with the assembler listing of the subroutine. Assembler listings are usually long, and such comparisons sometimes could not identify the exact code location unambiguously. Furthermore, the subroutine assembler memory cross reference map was difficult to decipher. Tracebacks provide the programmer explicit information as to which call to the interrupted subroutine is in force (i.e., the name and line number of the calling subprogram), and in the Ridge that feature was not easily available. Debugging a large code such as VIM on the Ridge without good debugging tools was very difficult, even with an intimate knowledge of the code.
THE WORKSTATION AS A DEDICATED RESOURCE

The most important aspect, after data transfer, that can limit the usefulness of the workstation is turnaround time. In the batch computing environment, I am normally working on several projects, one of which may be production computing. At the same time that one debugging or production job is in the batch system, I ordinarily work on other programming projects, usually involving editing or compilation and then short execution. Although I did not perform any long-running production calculations on the Ridge, I did edit files and carry out file management tasks while a compilation/linkage editing job was running in background. There seemed to be enough interference to slow both tasks substantially. The software available for checking process status was cumbersome and verbose. Interference also occurs in the IBM computers but only when dataset access is blocked by another task or when the system is saturated.

CONCLUSIONS

The scientific workstation is in principle an attractive computing alternative to the batch and Wylbur systems. It may be fast enough to save a good deal of money for scientists who do lots of computing.

Because of poor debugging capabilities, I was not able to demonstrate the usefulness of the system as either a production or a programming environment. I could recommend such an approach to computing only after (1) installation of an easy-to-use symbolic debugger or some equally useful debugging capability and (2) full accommodation of interfacing system utilities (in this case, C-functions) with Fortran. Furthermore, data transfer must be substantially faster for a workstation to make use of centrally maintained databases. I believe that the disk storage capacity of the particular configuration would have been, at best, marginal for my purposes. Both a larger disk and a tape drive would be necessary for effective use of the system. The Unix operating system and editors are relatively easy to use, but users need better documentation. In this project, the system administration was centralized; a workstation may not have such a centralized system. One should consider this desirable feature as another overhead to pay. Slowdowns that occurred because the computer was processing multiple tasks at the same time might be mitigated through the addition of a priority system.

Another function for a workstation might be not as a scientific workstation for one or even a few scientists but as a scientific task workstation. In this mode, the several users would run only one code, such as VIM, on the workstation. This restriction to one code might increase the machine load factor substantially, generating more substantial savings. It might also reduce the impact of a data transfer bottleneck.
Chapter 9

STRAW, A FLUID-STRUCTURAL AND THERMAL-STRESS FINITE ELEMENT PROGRAM

James M. Kennedy
Reactor Analysis and Safety Division

OBJECTIVES

Overall Goal

My overall goal was to find out whether or not a scientific workstation could be used to provide interactive and batch computing resources for my research needs in the development and project work associated with the fluid-structural and thermal-stress finite element code STRAW.13

Of particular interest to me was how difficult it might be to convert one version of STRAW to operate on a workstation. The answer to this question would depend on (1) how well the code was written and (2) what special difficulties would arise because of special functions and auxiliary devices.

Also of interest was the capacity of a workstation for production computing. Would the workstation produce results fast enough for the project engineer to meet the predetermined milestones? Would there be good communication with other computers or devices necessary for part of the production work? Could the workstation handle large mass storage?

Of some interest was the capability of a workstation as a programming environment. Would the software resources be similar enough to those of the IBM facility that development and maintenance work could be converted to a workstation in an orderly manner? Would the

---

workstation software give good feedback to the user during the programming task (debugging, printed output, etc.)?

Finally, I was interested in whether or not a workstation could manage a multi-task environment well. Would there be limitations or advantages associated with doing development work while there were long-term jobs running in the background?

PRECONCEPTIONS AND EXPECTATIONS

I did not expect the job to be extremely difficult because I had recently transported the STRAW code for use on a Cray and on a CDC 7600 at Los Alamos National Laboratory. With the help of an expert on those systems, the job took one day. I was more interested in getting the code operational in a new workstation environment than in investigating the development tool capability. In interactive computing, my primary experience is with the Wylbur editor, and I had no knowledge of Unix.

CONVERSION OF STRAW FOR THE WORKSTATION

Background for Code

Over the past years, considerable progress has occurred in the development of technology and validation of methodology for the safety of fast breeder reactors. This development addresses readily identifiable safety problems. It has involved (1) identifying potential safety problems emanating from phenomenology considerations, (2) identifying the response of materials, components, and systems to phenomena through development of calculation techniques and experiments, and (3) alleviating serious consequences pinpointed by the methodology and experiments through alternate design concepts.

The Engineering Mechanics Program in the Reactor Analysis and Safety Division has been active in developing extensive methodology to gauge the consequences of a hypothetical core disruptive accident. For large hypothetical energy releases in the core, the response of all important primary system components was evaluated with the development of fluid-structure interaction codes. Such a code is the two-dimensional STRAW program, which would use dynamic loads to provide large plastic deformations in subassemblies. Thermal field and stress evaluation is also a primary concern.
Brief Description of Code

Originally developed for subassembly fluid-structural response to local pressurization and/or thermal loading accidents, STRAW has evolved into a program with a wide range of capabilities. STRAW is now a two-dimensional nonlinear finite element program that has a weakly coupled thermo-mechanical formulation in addition to the structural and fluid-structural interaction capability. One may use STRAW to perform static (steady-state) and transient analysis. The finite element method employed is valid for large-displacement and rotations plus material nonlinearities. This method employs a convected coordinate technique (each element being associated with a coordinate system that rotates with that element) and a direct-nodal force computational scheme for the internal nodal forces. The governing equations may be solved by means of an implicit or an explicit temporal integration scheme. The implicit scheme allows the calculation of long-duration transients, static analysis and coupled or non-coupled thermal-stress analysis, since a strict stability limitation is not imposed on the time-step size, as is required for the explicit case. In the explicit case, lumped masses are used to represent the inertial properties of the structure, continuum and fluid, while the implicit scheme uses a consistent mass mode of calculation. Explicit-explicit partitioning can be used when strong differences in multi-material stiffnesses exist. Flexural, continuum and hydrodynamic element solutions can be achieved in plane and axisymmetric configurations. The hydrodynamic element is either Lagrangian or quasi-Eulerian. Flexural, continuum, and Lagrangian hydrodynamic elements are velocity-strain formulated. Prescribed temperature, heat flux and film-coefficient are the thermal boundary conditions available for the thermal field solutions.

Code Details and Conversion

I have been the primary developer of the STRAW program with assistance from several other individuals over the past several years. I am the individual responsible for the ongoing development and maintenance.

The explicit versions of the STRAW code have approximately 45 subroutines with 5,000 lines of source code, while the implicit have approximately 70 subroutines with 9,000 lines of source. Load modules for the explicit and implicit versions base program need roughly 200 and 400 kilobytes storage respectively. The code has its own dynamic allocation facility that allows for no limitations on problem size. To make production runs, the program may expand to 600 kilobytes. For convenience in obtaining quick batch turnaround, test runs can often be accommodated in a 250 kilobyte program region. Most functions are contained within the code.

As stated above, an explicit version of the code has recently been transported to Cray and CDC 7600 computers at Los Alamos National Laboratory. To transport this code, I first converted the code at
Argonne with VS Fortran on the IBM 3033. With some assistance from a person familiar with the Los Alamos system and some minor modifications, I was then able to make the code operational on the Los Alamos computers.

Converting the selected explicit version for use on the Ridge 32 was easy, with only a few necessary modifications involving Fortran 77 code acceptable to the IBM 3033. After a period of machine learning, the only changes that I made were to resolve external references and the numerical magnitude of a single controlling constant associated with graphical output and to add two source statements necessary for reading input data. This information is of special interest because it offered further evidence that the code had been written in standard form and is extremely portable. I never tested the graphics aspect because it was unavailable during my study.

The documentation provided by Ridge was adequate for the work I undertook. A fair amount of understanding of the Unix system was necessary for quick and easy use of the manuals.

The initial background training was adequate and sufficiently informative to enable one to start performing useful work. The periodic informal sessions were of limited value to me because most of the discussions were not germane to my work.

THE WORKSTATION AS A PRODUCTION ENVIRONMENT

My evaluation of the workstation as a production facility involves a set of sample problems covering many of the options available in this explicit version of the STRAW code. The IBM 3033 performs this task easily in approximately 295 seconds, while the Ridge took approximately 40 minutes. This time ratio was about what I expected and considered reasonable for the type of operations involved in the code computational effort. In light of this computational efficiency, the Ridge appears to be a practical tool for performing analysis with the STRAW code. However, after completion of the calculations, the volume and speed available for retrieval of information were somewhat less than I would like.

THE WORKSTATION AS A PROGRAMMING ENVIRONMENT

The minimal programming that I did on the Ridge 32 involved debugging the version of STRAW transported to the Ridge. This debugging went smoothly. For other programming tasks, I did not find the Ridge efficient in producing information. It was relatively slow in accomplishing the tasks I customarily do much more quickly on the IBM 3033 with Wylbur. When I made a mistake, it was difficult to break and control the processing. Many times I had to restart an entire process. Errors in the code were also difficult to fix because the error messages were inadequate.
THE WORKSTATION AS A DEDICATED RESOURCE

The degree to which using a workstation would be efficient would depend upon the extent to which one used it and the speed and flexibility with which the workstation could accomplish the desired multiple tasks. The Ridge appears to be very capable in handling the multiple type of tasks desired by the engineer/scientist. However, multiple tasks occurring simultaneously on the workstation seems to cut severely into turnaround time (efficient user service).

CONCLUSIONS AND REFLECTIONS

The concept of a scientific workstation as a reasonable tool for a time-sharing and batch computing facility appears to be attractive. The scientific workstation (Ridge 32) appears to be a good and reasonably efficient tool for computing. The Ridge handles many types of tasks in individual form capably but becomes unduly slow with multitasking. Debugging capabilities and communications need improvement. At this time my personal preference is to continue my development and project work with Wylbur and the IBM 3033 until further advancements in scientific workstations make them more desirable.
Chapter 10

THE ELECTRONIC STRUCTURE OF ATOMS AND SOLIDS

Dale D. Koelling
Materials Science and Technology Division

CHARACTERIZATION OF EVALUATOR

My research deals with the electronic structure of atoms and solids. In pursuing these studies, I perform calculations with a wide variety of computer programs, all of my own construction. The programs typically range from 500 to 5,000 lines in size and are written in Fortran. They do intensive, double precision number crunching. Most datasets involved are small (about 50 kilobytes), but some are medium sized (about one megabyte). This dataset requirement may expand as we begin to deal with more complex systems. The majority of the computer cycles used are appropriate to a non-interactive environment. The computer must grind on for some time to produce a few numbers, which then are collected together with the results of more runs before a final analysis is performed. The codes performing these calculations are evolving, yet moderately stable, production codes. The effort that consumes the majority of my time with the computer is the process of evolving those codes and the process of analyzing and utilizing the resultant data. Data preparation is a relatively minor operation. Overseeing the long runs is, unfortunately, a time-consuming task, especially if the operating system does not provide good control language tools. Analysis often requires the production of special purpose codes. I have at least one program in a writing and debugging phase at all times.

Limited access to computing cycles (primarily for budgetary reasons) restricts the scope of my efforts. In addition to easy access to computing cycles, I need efficient program production in the course of my research. Most of my approximately 20 years experience has been with IBM equipment, primarily in a batch mode. Over the last five years, I have been using more interactive facilities--Wylbur, VM/CMS, DEC VAX/VMS, CTSS, and Unix. I have also been effectively augmenting my efforts with personal microcomputers, with one at home and another at my office.

PRECONCEPTION AND EXPECTATIONS

General Concept

If our physics is to advance, rather than to be merely a more thorough examination of similar problems, we must have access to the more powerful machines. There is a certain danger that access to capacity through workstations will confuse the perception of our need for increased capability through Class VI and Class VII machines. The workstation merely offers increased capacity to do the smaller jobs. It does not offer any new capability to take on the bigger jobs that are the future of our research. The workstation must be a tool to access the greater capability of the supercomputers, or we face stagnation in our research.

My primary goal for a workstation is that it serve as an effective and tailorble front end to large-scale computing. It should provide all the tools to do most of my small-scale computing. Large-scale number crunching, mass storage, and miscellaneous other tasks in which no advantage accrues from specialization (most printing and plotting devices fall into this category) belong in a central computing facility. I envision the primary use of a workstation to be for building and maintaining programs. Then, when resources are needed on a large-scale number crunching system, an up-to-date copy of the program would be transferred to it for use. This scheme of source code maintenance achieves a coherence in calculations that might be run on several different hosts by avoiding the many difficulties associated with maintaining separate copies of the same programs. Clearly, the workstation must have good communication. I would prefer to do interactive computing on the workstation, but I recognize that some of my calculations require larger computational power that must come from a more powerful computer.

There are a number of features required by this model for the workstation. I list them here with comments.

• Speed: Interactive computing must be responsive if it is to be advantageous. The faster the system, the bigger the calculation that can be handled before having to switch to the mainframe. In a program debug and validation mode, the processing power determines how real the test cases are that can be employed: it often requires considerable computation to establish the environment where a bug will manifest itself. Further, just the time to do a test compile for program sizes typical of my applications becomes debilitatingly long unless one operates at speeds comparable to the speed of a DEC VAX-11/780.

• Communications: A front end system must communicate with the back end by definition. One feature that must be present is a terminal emulator with file transfer. This mode is very flexible—a must when we communicate with the IBM systems, remote Cray computers, and other minicomputers (of our collaborators). We will always need to communicate with that new or old system somewhere.
However, that mode is limited by modem speeds and can be very slow! Maintaining the source of a 5,000 line code and transferring it each time it is needed is not feasible at terminal emulation speeds. Having a better file transfer mechanism also is mandatory if one wants to use the central facility for most printing and backup services. I firmly advocate that money is better spent on improved communications than on local printers or backup devices.

- **Tools:** One needs a good compiler (Fortran in my case), a symbolic debugger, a good editor, file management, and assorted other tools. I do simple printer plotting, and hence I foresee graphics as a very desirable tool. Emphasis should be on flexibility and tailorability, more than on ease of learning, because it is intended that the workstation be a long-term operation. Because of the "back-end" concept, I prefer that the compiler and other tools help me to keep source code portable by adhering strictly to standards and by having features to identify extensions to standards when they are used.

**Unix Operating System**

Having had some exposure to Unix on the Mathematics and Computer Science Division VAX, I was aware of both its flexibility and its terseness. Because I could foresee that I would have to work with the system, I was eager to gain experience. What I had to learn was that there is a style appropriate to using Unix that differs from the style common to most batch usage. The essence of the style appropriate to using Unix is that one works with small units strung together to form the final solution, rather than working with one very large package with many options. My casual experience with Prime and Apollo systems and my extensive experience on the VAX would imply that this Unix usage style is common in using minicomputer systems; the Unix usage style is just more extreme. Finally, Unix is not Fortran-oriented. A symbolic debugger or even a first quality (i.e., bug free with good optimization) Fortran compiler is not included with all Unix systems. These are needed for my model of a workstation. Thus, while Unix is a very usable operating system, it must be augmented by these tools to fulfill my requirements.

**Ridge 32 Hardware**

My expectation was that the Ridge 32 could easily be used for production computing, and this expectation has proven correct. Significantly, the Ridge 32 functioned as expected with virtually no hardware failures. I was concerned that software limitations would confine the system to only a production role and would not permit a workstation role. That is, I had great reservations about the lack of debugging tools and of a better communication option, both of which would limit my use of the Ridge as a workstation. My fears were borne
out. The Ridge, however, was the only system that offered the speed required at the time of the acquisition.

**THE WORKSTATION AS A PRODUCTION ENVIRONMENT**

*Codes Involved*

I acquired initial experience on the Ridge 32 with an atomic structure code. This code performs a usable unit of calculation in roughly a minute and uses very small datasets. The majority of its computation is in performing one-dimensional integrals and solving two coupled first order differential equations. The final production computing for a recent manuscript was done on this system.\(^\text{15}\) This manuscript discussed the application of optimized effective potential techniques to density functional theory.

The second package of codes converted was a series of electronic band structure analysis codes. This package does Fourier analysis of the band structure to provide a fast, convenient representation. From this representation, the density of states is calculated to determine a Fermi energy. This energy calculation is then used to determine the Fermi surface. This package was my primary example of interactive computing. It is important to determine extremal cross section areas of these topologically complex surfaces for comparison with experimental de Haas van Alphen data. Determination of these cross sections is done manually by inputting planes to be explored and looking for extrema. For a complex Fermi surface, this procedure requires an interactive search.

Finally, I brought up and ran a version of my major production codes. These are self-consistent, linear-augmented plane wave method electronic band structure calculations. In the first of these codes, large generalized eigenvalue problems are set up and solved. In the second, the solutions are reanalyzed to obtain a charge density. That density is then turned into a potential via an integral transform. These codes require long precision, run for a long time, and involve a large (approximately one megabyte) wavefunction dataset. They would form the bulk of my background computing queue.

CONVERSION

The major conversion efforts were in finding and circumventing compiler bugs. Most of my codes were already in Fortran 77. For those that were not, only the minor efforts of converting alphabets to character type and introducing OPEN statements were adequate to get started. However, the Fortran compiler delivered initially was totally unacceptable in that it had a number of bugs and very poor performance. This compiler was replaced by an improved version with better performance. The major bug I found in that compiler was its mistreatment of parameter statements associated with type real data. Type integer parameters worked fine. Most of the compiler bugs were associated with the Fortran 77 features. Debugging tools were clumsy at best, but this deficiency of the system was known in advance. One interesting feature did appear which seems to be a characteristic of the Ridge floating point format. There is a very large dynamic range in double precision (11 bits in the exponent). One of our algorithms was unstable in the regime of very small numbers. This same program caused no difficulties on the DEC VAX and the IBM systems, we think because the very small numbers were set to zero. We easily reworked the algorithm for stability.

Capacity

The Ridge ran the atomic codes roughly one-third faster (in elapsed time) than the VAX-11/780 with the VMS operating system. The solid-state augmented plane wave codes also ran faster, but the comparison was not well controlled. One cycle completed in about five hours. Simple materials could be converged in a couple of days, while the more complex ones would take months (I have done both on a VAX). For this evaluation, I converged a fairly simple material (ScN) because of the one week limitation. With some planning and a better use of shell scripts, it would not be difficult to keep a background job running at all times. (I would have done so earlier in the project were it not that, until release 3.1 of the Ridge operating system was installed, a background job seriously impeded the foreground.)

The primary value of a workstation is in doing interactive computing. It is the man-machine interface for one's computational efforts. A major component of this interface is the programming environment. The other component is the interactive solution of problems where it is inefficient to try to program the computer to do all the analysis. My example was the Fermi surface problem. In doing such computing, it is important that the response of the computer be reasonable, and the response of the Ridge for that problem was quite acceptable. A factor of two slower could have been tolerated, but a factor of four could not. When the response gets too long, one tries to do other things while waiting, and concentration on the details is lost. A second application that I had wanted to implement involved a sophisticated non-linear least squares fitting that could greatly profit from interactive graphics. That effort was blocked by a difficulty with
the communications software prior to and during my testing week (clearly demonstrating the importance of communication).

There are really only two possible advantages of the workstation as a background production machine:

1. As a dedicated machine, it can be tailored either by the addition of hardware (such as array processors) or special software configurations.

2. It is reliably available, regardless of short-term budgetary limitations.

On the other hand, it is roughly an order of magnitude slower than the Computing Services IBM 3033 machines, which are at least an order of magnitude slower than class VI machines (for my work). So, as production systems, workstations do not offer any new capability and, in fact, provide less. The workstation does offer increased capacity, however, with the very convenient feature that it is in small increments of cost. This capability is in contrast to adding a new mainframe, which is a quantum jump. The critical factor now becomes the question of cost effectiveness. Assuming that the workstation delivers one-tenth the productive capacity of one of Argonne's IBM computers (an underestimate in my case), the system provides about three percent of the total IBM computing capacity—which is greater than my current usage of that facility. The cost factor is a more difficult item. Using the $50,000 declared price, the capital cost for a four year period is $12,500. Maintenance (at one percent per month) should run about $6000. Operating costs are inestimable, but can be managed to be quite small, if necessary, at some loss in effectiveness. The most significant part of these costs will be communications in my model. The remaining cost is the researchers' time to maintain the system. This I am quite willing to overlook, primarily because of a comparable effort to acquire time on other machines, participate in users groups, etc.

Note that a fully accurate cost analysis would include additional "hidden" costs for services commonly provided with mainframe computers, such as systems programming, operations management, data backup, consulting, and communications with the vendor. For the workstation user who must provide these services himself, these costs should represent at least the value of the research time lost. Still, the workstation appears to be a cost effective way to do the background computing, if it can be managed so that it is fully utilized. I (and most of the other Materials Science and Technology Division theorists) can do that with very little effort. However, the availability of cost effective workstations does not lessen our need for the greater capabilities of the central computing complex. The central complex includes the IBM 3033 mainframe computers for more powerful, faster computing and operates a number of devices to which we need access—printers, tape drives, mass storage, communications. These needs would continue even if the workstation were faster than the central computers (and it is much slower). Some computations will run too long on the workstation to be timely. In fact, some run long enough that one begins to approach the mean time to failure.
THE WORKSTATION AS A PROGRAMMING ENVIRONMENT

My model for a scientific workstation requires that it be a strong tool for the task of program development. As anticipated, the Ridge system did not perform this task satisfactorily. Standard Unix does not come with a symbolic debugger, and I believe this to be a critically necessary enhancement of the system. The debugging tools on the Ridge consisted of a low-level debugger that was effective but clumsy. It was necessary to work with an assembly language listing and at the same time not to lose the program conditions. This necessity, coupled with the bugs encountered in the compiler, made the Ridge weak for program development. The software is just not in place for Fortran yet. An exception was the use of the Unix tool make coupled with subdirectories. One can easily envision that capability to be the basis of an improved project management scheme. The brevity of this exercise precludes any realistic evaluation of all the tools, but the Monitor technology being used in the Mathematics and Computer Science Division16 to program the Denelcor HEP has impressed me with the desirability of the macro capability of the Unix tool m4 for the production and maintenance of source code.

The Ridge deficiencies in this area are no doubt the growing pains of a new system and will be corrected. The deficiency should not cloud our evaluation of a workstation environment. I have used the Apollo, Prime, and DEC VAX VMS debugging facilities and find them a great enhancement of my productivity. It is unfortunate that to acquire the speed we had to give up this very important feature. It is appropriate that the Ridge system be revaluated in this area when the company introduces their symbolic debugger and new compiler.

THE WORKSTATION AS A DEDICATED RESOURCE

Having a powerful computer dedicated to an individual or a very small number of individuals provides good service for their needs. The yearly charge for a workstation like the Ridge is about one-fifth that of a professional staff person. Although it is not super-expensive, there is no advantage to the workstation being idle. It can be kept busy by maintaining a background queue at all times. This different aspect of the workstation environment is that someone is now directly responsible for always keeping a task in the machine.

One additional candidate to increase the utilization is word processing. I currently do word processing on a personal computer. That application should move to the workstation. Vi is a good full-screen editor, and I did use the Ridge for manuscript preparation.

One further simplification is that the editor used for manuscript preparation is the same as the one used for computation. I personally am very comfortable using a workstation as a word processor.

Finally, I should observe that a dedicated workstation can be tailored to its user. For example, if the editor of choice is the Wylbur editor, it is possible to acquire a Wylbur editor for the environment. If one can justify commands in Sanskrit and the funding is available, so be it. The workstation accommodates the human; the supercomputer requires accommodation from the human. By inserting the workstation between the human and the supercomputer, one should be able to accommodate both. The Ridge was shared and was communicating only with the Argonne's central IBM systems, so I did not experiment much with this kind of accommodation. However, my large-scale computing also extends to Cray computers, and I can see that I will be doing more with the Denelcor HEP computer, too. To be effective in switching between these, I will need all the help I can get. As a trivial example, because I am currently working with five separate editors, I cannot be very sophisticated with most of them. I also make mistakes when I give EDT commands to Wylbur, Wylbur commands to TEDI, etc. I really want to use one editor to maintain a common master source code file and use a macro facility to insert machine-dependent pieces into source codes that are to be exported. With such an organization, debugging is done locally only once, rather than at each installation. The current limitation on this scheme is in the communication. The Ridge did not prove a good test for this model, as its communication capabilities were weak at best. However, my experience with other systems indicates that good communication capabilities are an achievable goal.

CONCLUSION AND REFLECTIONS

The concept of a workstation environment is a valid one, but this study lacked several features to demonstrate it. The workstation has the capability to run effective background production. The project demonstrated some of the useful tools of Unix but not a good debugging environment. The compiler has some bugs, and the primary debugging tools are not yet in place. Communications are also weak and caused considerable disruption during my dedicated testing period by not working at all. Thus, I conclude that this test did not prove the concept, but did show the promise. (We felt the promise adequate—and will be continuing the test—within Materials Science and Technology Division, now that we are acquiring the Ridge equipment.) We expect that both deficiencies will be eliminated as the Ridge system improves and the Apollo systems compete with it in speed (we have tested the Apollo systems speed with some demonstration codes). I am convinced that the capability to fulfill my model for a scientific workstation has arrived in the marketplace, although it is still in shakedown. Combining my experience with the Ridge with that on other systems, I am also convinced that the workstation is a tool that will dramatically affect how well we cope with the computational problems facing us in the next few years.
Chapter 11

ENVIRONMENTAL TRANSPORT MODELING

Anthony J. Policastro
Environmental Research Division

This chapter summarizes the observations and findings of a group of users whose specialization is the mathematical modeling of environmental transport physics. It includes the results of work done with three separate programs and by three individuals, Richard A. Carhart, Larry R. Coke, and Michael J. Wastag. We consider the models tested on the Ridge 32 typical of the larger codes developed for simulations of environmental impact. We discuss our experiences and provide some subjective commentary for review by similar user groups who may be considering the Ridge 32 as a principal computing resource.

BACKGROUND

Our research in the area of environmental transport involves development, modification and operation of computer codes which vary from about a quarter of a megabyte to two megabytes in size. The typical code size is under half a megabyte. The codes that we work with are written exclusively in Fortran IV or Fortran 77. Some of the codes require array storage up to one megabyte, and generally the execution is both CPU and I/O intensive.

Because our projects have a limited computing budget, we must continually plan against cost overruns. While the costs of interactive VM/CMS computing have decreased, they have decreased at a slower rate than the rate at which batch computing costs have decreased. We have therefore chosen to work more in the Wylbur and OS batch environment. The main difficulty with VM/CMS is with its relatively high charges, especially during the "prime" hours (Monday through Friday, 7:00 a.m. to 7:00 p.m.). While batch is suitable for production runs, it is not the best environment for code development and debugging. The relatively high cost of using VM/CMS has led us to use the batch system almost exclusively. With the batch system, however, we sacrifice human efficiency for the sake of reduced total costs (labor plus computer time). In fact some of us work around these limitations by using VM/CMS during non-prime hours to take advantage of both the interactive computing and the lower charges. The Ridge provides us the potential of increased worker productivity through the use of an interactive system.
which permits intensive usage for sustained periods of time with a fixed cost known in advance.

OBJECTIVES

Our primary goal was to evaluate the Ridge 32 computer to determine its potential as a medium for the development and execution of computer codes for environmental modeling. We also wanted to compare the "number crunching" power of the Ridge against the IBM 3033 and to evaluate the compatible software utilities. To these ends, we planned to perform test runs of existing programs to determine the extent to which the Ridge 32 could be used in our work.

PRECONCEPTIONS AND EXPECTATIONS

Based on our experience with other microcomputer and minicomputers, we had both high hopes and some reservations about the Ridge 32. Our experience has been that microcomputers are well suited for word processing and the rapid development of small utility programs (e.g., using the Basic language). On the other hand, we had found serious difficulties and limitations because of a poorly written compiler (IBM Fortran 1.0 for the IBM Personal Computer). We were somewhat skeptical that the Ridge Fortran 77 compiler would be of sufficiently high quality to handle large codes. We had no previous experience with Unix, but we felt that a new system was worth learning to have increased productivity and reduced costs. As a group we have considerable experience using both Wylbur and VM/CMS; yet we recognized the potential productivity and economic advantages of a successful Ridge 32. Our microcomputer experience made us aware of the benefits of controlling our own computer. Our Ridge evaluation plan was to carry out two types of work assignment on the Ridge 32: (a) to run existing programs, and (b) to develop new programs in an interactive environment.

CONVERSION TO THE WORKSTATION

We selected a number of large models that were already operational on the IBM mainframe for our comparison study. Three individuals performed this task in parallel, working on the codes with which they were intimately familiar. The results are summarized below for each of three environmental models.
EPRI COOLING TOWER PLUME AND DRIFT MODELS

The EPRI model consists of four codes designed to be run in sequence. These codes had been written in Fortran IV with careful attention to transportability. As a result, we expected that implementation on the Ridge would be easy. We had to do our file transfers by modem. We encountered difficulties with the Kermit file transfer system and finally abandoned the transfer attempts through that utility in favor of an intermediate microcomputer transfer. This method worked successfully but required more time than the direct connection. When some bugs in the Kermit utility were fixed, we transferred some necessary data files from Wylbur. Unfortunately, we found the Ridge Kermit utility unfriendly because it does not permit transmitting the "break" signal needed to interrupt interactive tasks in the IBM sessions. We were able to transfer our data in spite of this difficulty. One serious difficulty remained: the EPRI codes accept data from meteorological tapes, and some of these tapes are not in edit format compatible with Wylbur. We could either reblock the tape or else find another method for data transfer. We believe that it is dangerous to depend upon an editor to transfer general EBCDIC files. Furthermore, we believe that reblocking is only a last resort. Therefore, for this test case, our solution for the difficulty was simply to avoid it by selecting a different test tape. This is not a prerogative which we would have in our production work.

Once the data and codes were transferred, we compiled the codes. Compilation revealed several minor syntax errors that had resulted from language differences between Fortran IV and Fortran 77. We used the line-oriented Unix text editors to correct these and attempted to execute the first of the four codes. The program stopped with no diagnostic other than the word "abort", and that was terribly disappointing. Being familiar with the code, we made some good guesses and traced the problem to omission of the necessary statement to open a file.

We repeated this type of debugging for the other three codes and eventually obtained successful results. Almost certainly, a user unfamiliar with the codes would not have found these bugs easily. It was evident that the debug support for the Fortran 77 on the Ridge during the project term was terribly inadequate. On the plus side, the language implementation appears to be very good; transportability was excellent for both Fortran IV and Fortran 77 codes.

We made the actual runs with no difficulty whatsoever. The Ridge I/O redirection and default assignment features allowed programs written in the older version of Fortran to run with minimal changes. Execution times were measured for each code and compared to the known elapsed time on VM/CMS. The multiple tower plume and drift code (the third code of the four code sequence) was run on VM/CMS in 22 minutes elapsed wall clock time. The corresponding time on the Ridge was 65 minutes. The seasonal/annual preprocessor code (the first code) had similar times of five and fifteen minutes. In each case all program output was written to disk. Both programs perform extensive computation and I/O. The
results were better than we expected; times were uniformly a factor of	hree longer than the VM/CMS elapsed time. We find this fact especially
impressive because we obtained the VM/CMS times during hours when we
normally experience the best CMS response. We have observed VM/CMS
response to degrade up to a factor of ten longer during prime hours when
usage is heaviest. Now that the codes have been set up, the Ridge is a
perfectly suitable production environment except for the file transfer
problems.

RTM-II--LONG RANGE TRANSPORT MODEL FOR AIR POLLUTANTS

This work was started during the one-week dedicated test period
with the user having no previous Ridge or Unix experience. The user did
have very extensive experience using the IBM mainframe and Fortran IV,
as well as a large amount of microcomputer experience. The goal was to
implement the RTM-II code on the Ridge and to rerun one of the
long-range transport prediction cases (simulation of a two-day period)
that were produced on the IBM system last summer. This model uses a
three-layer two-dimensional grid within which it solves the transport
equations for pollutants as a function of time (with time steps of five
minutes). Thus, a total of 576 time steps had to be run for the two-day
environmental transport simulation. The simulation uses six input
files, two of which are short and have records limited to 80 bytes, and
four of which are large, unformatted, and total 5.8 megabytes. The
simulation produces output files requiring an additional megabyte of
disk space.

For RTM-II, the IBM resource requirements were the following:

a) 512 kilobytes to compile; 512 kilobytes to execute

b) 8.7 CPU seconds to compile (with the IBM Fortran G compiler)

c) 37.0 CPU seconds to execute

During the week, the user was able to learn enough about Unix to
manipulate, edit, and print files and to compile and run Fortran 77
programs. Since the method of file transfer excluded our binary input
files, these files had to be formatted and then transferred. The
formatting of the data caused a loss in precision that disappeared when
the binary files were later produced internally with full precision on
the Ridge. The code itself was converted with limited changes needed to
compile with Fortran 77. However, during attempts to run the model we
encountered run-time errors. The Ridge did not produce feedback that we
could use to determine why it had ended abnormally, and we even found it
difficult to discover where the program had stopped. The absence of
good debugging tools was a nearly insurmountable barrier. The
subroutine where the error occurred was easy to find because the
subroutine was small. But the source of the difficulty was a segment of
code in a large subroutine. By the fifth test day, the code was
operating, and we were able to gather the following comparison
statistics on the test run:
a) Compile time was about five minutes.

b) Execution time was six minutes with binary input files (with formatted files the time was twelve minutes).

For our needs in running RTM-II, these compilation and execution times are adequate. We found later that there were small numerical differences in results when the binary files were established directly on the Ridge with the preprocessor programs. We determined that the sources of the differences were the different numerical precisions of the two systems. (The Ridge single precision numbers have more bits for the mantissa.)

UI/ANL COMMUNITY NOISE PREDICTION MODEL

The UI/ANL noise model predicts noise levels at any number of receptors from any number of sources. The model was designed to be as machine-independent as possible to minimize modifications to the code by future outside users. The noise prediction model consists of two Fortran IV codes (a preprocessor and a noise propagation code) along with a noise source directory. The three files, which total approximately 5000 records, were transferred from the IBM mainframe to the Ridge 32 with no difficulty.

There were very few compilation errors with the Ridge Fortran 77 compiler. In fact, all the compiler errors in both the preprocessor and the noise propagation codes were associated with data initialization and character type declaration statements. The errors were corrected with minimal modifications, although it did take a while to understand and decipher the meaning of the errors indicated by the Fortran 77 compiler. Some of the compiler diagnostic messages were difficult to comprehend at first reading. This difficulty was also due to our limited knowledge of standard Fortran 77. We made a few guesses at correcting the errors before the diagnostic messages vanished or changed to warnings.

The noise preprocessor and prediction codes did end abnormally while executing for the first time with the same I/O error. The error diagnostic displayed on the terminal screen because of the abnormal termination cited a part of a Fortran format statement associated with a write. Apparently, there was some formatting error. We discovered that a syntax error existed in the format statement that was accepted by the compiler as proper syntax, but upon execution, the system had difficulty in interpreting the format specifications. We located the same syntax in other format statements throughout both codes and modified it appropriately.

We compared the printed output for a successful noise model test case executed on the Ridge with the same printed output from the IBM batch system. The test cases used exactly the same input data. The two printer files were identical character for character. We would have been disappointed and puzzled if there had been any discrepancies, since the noise model does simple and straightforward arithmetic computations.
The noise preprocessor code ran slower on the Ridge by a factor of eleven, while the noise prediction code was slower by a factor of twenty. We compared the Ridge execution times with identical runs on the IBM mainframe batch system, where the H extended Fortran compiler with default options was employed.

OVERALL EVALUATION OF THE WORKSTATION

For users who need to run large "number-crunching" computer codes, it is important to have adequate computing power. Our experience on the Ridge has been very favorable; in fact, our results uniformly show better performance than previously quoted benchmarks, provided VM/CMS is compared to the Ridge. The comparison of batch with the Ridge is less favorable but still satisfactory. A good rule of thumb for Fortran CPU-intensive codes is that they require a factor of three longer in elapsed time than the optimum performance of VM/CMS (typical of 2:00 a.m. on a weekend). Allowing for the heavy load during a typical weekday, the elapsed time execution compares very closely, assuming that the Ridge 32 is lightly used. Note, however, that the compile time for Fortran 77 is a factor of 10 longer than the optimal VM/CMS elapsed time. In any case, it is fair to note that people work in elapsed time, so the waiting time for batch queues and the true waiting time on the mainframe must be included in comparisons with the Ridge. We find the Ridge performance in compilation and execution to be acceptable in practice.

It is also significant that the Ridge virtual memory system permits execution of codes at least as large as the largest code permissible under the present VM/CMS limit at ANL. If this result holds up to the full virtual memory capacity, the code size will not be an issue.

Like the vast majority of the scientific and engineering computer models, our codes are written exclusively in the Fortran (or Fortran 77) language. It is therefore essential to have an efficient compiler and good diagnostics during compile and execution stages. The Ridge compiler does appear to generate efficient code. The problem with the current ROS 3.1 version is its lack of good diagnostics and the absence of a symbolic source-level debugger. Even the simple cross-reference tables found on other compilers are unavailable on the Ridge. The absence of a symbolic debugger almost negates the benefits of having an interactive work environment. This situation is expected to change when the vendor delivers the next upgraded version of the operating system (ROS 3.2) in October 1984.

A very important feature of the Ridge 32 is its use of the Unix operating system. ROS incorporates features of Unix Version V as well as BSD 4.1. Members of our evaluation group had not used Unix previous to using the Ridge, so it was necessary to learn Unix. Our previous experience with operating systems was primarily with Wylbur, TSO, and VM/CMS at ANL and with CDC elsewhere. Therefore, we may offer some encouragement to those who may be faced with the same situation. At
least some members of our evaluation group were thoroughly familiar with the CMS system as general users. We found that Unix was easy to learn. Although some of its commands appear cryptic, we feel that the operating system is generally user-friendly and provides a wealth of utilities, such as a spelling checker, a choice among five text editors, mathematical typesetting, general text formatting utilities, niceties such as a calendar, a reminder service, mail, graphics capabilities, file scanning and comparison utilities, communication utilities to remote systems, data security by encryption, file access control, and the Unix filenaming path structure.

Our experience generally supports the view that Unix is truly a human-engineered operating system designed to assist the user in getting his work done. Other operating systems, like VM/CMS are currently more limited (how does one construct a tree or pipeline in VM/CMS?). Operating systems should provide the means to carry out a project from top to bottom; they should be capable of importing data and communicating with other systems, communicating among users, developing and running programs, and preparing and formatting the final report document. Unix is the only environment we have used that can readily do all of these things.

One additional benefit of Unix is its wide availability. At least one member of our group has worked with Unix on another machine since learning it on the Ridge. The knowledge of Unix was immediately transferable. The same cannot be said of those systems running only on proprietary vendor hardware. For the general user community, user transportability is a great benefit that is at odds with the interests of hardware vendors. The growing popularity of Unix may force certain vendors to relegate their proprietary interests to the broader interests of their customers and to develop better and more transportable software.

The major problems now facing our working group center around I/O. The lack of a fast printer in the project Ridge system and the slow transfer rate to the mainframe affect the output adversely. The Ridge will generate all of the disk output previously handled by the mainframe. In this respect, the IBM system at ANL is very powerful and allows the transfer of large files throughout its network. It seems desirable to acquire a networking capability so that the mainframe I/O hardware is readily accessible. In particular, the absence of tape I/O on the Ridge was a problem for us.
CONCLUSIONS

We discovered two major reasons why the Ridge 32 can serve the needs of our modeling group better than the IBM 3033 mainframe:

1. The Unix operating system offers an environment superior to IBM VM/CMS.

2. Use of the Ridge 32 will reduce our computing charges.

We temper our observations on the software deficiencies unique to the Ridge by noting that the vendor will have an upgraded version within the next several months that will address these deficiencies. We note two deficiencies in the project workstation configuration:

1. The limited I/O facilities

2. The lack of high speed and binary file transfer between the Ridge 32 and the IBM mainframe

Overall, we have determined that the Ridge 32 can serve as our primary computing environment in a distributed network encompassing the IBM mainframe, the Ridge 32 minicomputer, and various microcomputers (for offsite users).
Chapter 12

OPERATING A WORKSTATION

The scientific workstation equipment was available to user participants without their having to be involved in either the operational or the management aspects. However, the workstation had to be operated and managed either by the user or by another party. Much of the management activity involved planning for maintenance, both of the hardware and of the software. In this chapter we describe our experiences so as to provide perspective on the additional efforts required of the manager or owner of a scientific workstation like the Ridge 32. Some of the tasks are non-technical and do not have to be handled by a scientist or engineer. Some of the tasks must be done by the workstation user.

Our first activity before releasing the system for heavy user activity was to integrate it into the Argonne environment in such a way that the newly created computing environment would include the Ridge 32 as an efficient part. Conceptually, the Ridge workstation should be the center of its user's computing universe. Other resources should be available easily as needed from the workstation. From the workstation point-of-view, integration involves establishing communication to and from the already well-integrated computing environment at Argonne, where the IBM computers form the hub of a file transfer network. Also, we had to acquire a base of experience.

INSTALLATION

Ridge personnel installed the Ridge computing system as specified by the acquisition (lease) contract. It was delivered with the software installed and essentially had only to be plugged in and turned on. The installer checked the system over before turning it on; this precaution may have served to avoid some unknown difficulty. The installer changed some microcode and patched some software at the same time.

The installer helped to set up a modem for dial-up access to the Ridge. He also attempted to get communication software working that would provide file transfer between the Ridge and the IBM systems. (It was the lack of working communication software that delayed acceptance of the Ridge until March 5 after the company installed and demonstrated the Kermit software on February 29. The Ridge plan to use Unix utilities to do the required communication failed because the utilities would not adapt for communication to the IBM environment.)
EDUCATION

As a part of the equipment contract the vendor was to supply one day of training. In fact, the company provided about a day and a half. We involved about ten persons in the "hands on" educational experience, including the project participants and some other interested people. Four display terminals were attached to the Ridge in addition to the graphics display. (This experience was to be our first taste of the poor handling by the Ridge of multiple users and multiple processes. Indeed, when the Ridge was first delivered, the multiple process handling was extremely slow. There was no opportunity following the installation of ROS 3.1 to test the system in the same configuration to see if the behavior were still bad.)

For the purpose of acquainting users with Unix, which was new to all of them, the sessions were adequate. A better structured presentation could have covered more ground and have been better oriented to the technical audience. Prior to the sessions, commercial documentation of Unix was available to the user participants. For the most part they had not seen a Unix documentation set.

During the project term, we held regular meetings to bring the evaluator participants together to share experiences. This activity was particularly valuable to most of those who were involved in the project. These meetings were also an opportunity for the presentation of short tutorials to introduce Unix features and programming tools to the participants. The project manager, Richard Raffenetti of Computing Services, prepared and presented the tutorials.

COMMUNICATION

No company-supplied networking option or equipment options existed (namely a synchronous port and driver software) to implement a custom networking option. The company intended to meet the acquisition requirement of communication with the IBM machines by using Unix virtual terminal emulator utilities plus custom software. The initial attempts to establish communication failed because of special communication protocols of the IBM systems. The virtual terminal emulator functions primarily for Unix to Unix communication.

However, before the company finally provided a communication solution (about two months after delivery of the equipment), we had analyzed the communication capabilities and had written custom software to do the job of file transfer to and from the Ridge and either IBM CMS or Wylbur. The Unix utility cu was used to establish virtual terminal emulation to connect to either VM/CMS or Wylbur. After setting up the file transfer, we discontinued the session and ran a custom program on the Ridge to cause transfer to the remote editor or capture of text received from the remote system. For example, in a Wylbur session the custom program caused Wylbur to list the active file and at the same time to capture that listing and divert it to a file. This mode of file
transfer does not provide for error detection, but for the relatively
close connection distances involved, errors were not frequent enough
ever to pose a problem. We established effective file transfer between
Wylbur, VM/CMS, and the Ridge by the end of January.

While the method was effective, it was not fast. The virtual
terminal connection to the IBM systems was through a shared, X.25
multiplexer/concentrator where the line speed to the IBM 3705
telecommunications front end was 9600 baud. The most commonly used
transfer was from Wylbur to the Ridge. The transfer rate for user data
in that case was about 3000 baud (375 characters per second) at best.
Transfer from the Ridge to Wylbur was slower.

SYSTEM ADMINISTRATION

Some of the tasks of system administration for the workstation
existed because of its use as a shared project resource. Other tasks
will exist for any workstation, whether it is shared or not.

We established and administered accounts so that all users would
have access to the workstation. Account administration consists of
placing entries into a file in the system manager's directory. The line
has an encrypted password field that the system sets through user
commands. There is also a user identification number and a group
identification number. Another file describes the user groups. We used
the group feature where several persons might work together in the
evaluation. That feature permits the assignment of different file
protection attributes for groups and for other user categories. File
protection was not a great concern of the evaluator participants, and,
in any case, there was never a difficulty with file security.

One system administration responsibility was to maintain sufficient
supplies. For the Ridge workstation, the supplies consisted of floppy
disks, printer ribbons, and paper for the printer. Enough floppy disks
were ordered and were on hand for each user to get started. Eventually,
Central Stores stocked the specific diskette used in the Ridge for easy
and quick availability. The current price of a single diskette from
stores is $3.86. The printer arrived with only one ribbon, but we
obtained a supply of six to provide for replacement. During the eight
months while the Ridge was operating in building 221, the ribbon was
replaced five times. The cost of the ribbons for the Genicom 3404 dot
matrix printer was $45 for a box of two. We noted that each ribbon
lasted approximately as long as one box of paper (about a month). We
used the same paper that the IBM 3800 printer uses. The cost of a box
of this paper from Stores is $16.64. The printer was set to print in a
font size so that the standard 132 character lines would fit on the
11-inch-wide paper, and there could be 68 lines per 8-1/2 inch page
(eight lines per inch). Once we had set the printer up in this way, we
stored its configuration internally. Additional adjustments were not
necessary.
Custom commands consisting of versions of the Unix pr (print) command were placed in the standard library so that users could easily print numbered or unnumbered listings of source code. The printer was not spooled in this workstation, so multiple users could each send information to it at the same time. Output produced that way was generally confused and of no value. Even as a single-user resource, it was possible to create the same situation. In general, only that user working at the machine did printing, thereby avoiding the lack of print spooling or other synchronization.

System administration required some communication with the vendor. The system administrator was a funnel for information flowing to the vendor and was generally the vendor contact for reverse information. When difficulties occurred, they were communicated to the vendor for resolution. This sort of exchange of information occurred frequently because of the immaturity of the system and was exacerbated by the installation of revised but not thoroughly tested software. The vendor response to problems that prevented work from being done was very good, but the time spent in tracking problems was significant. All the major problems were communicated to the vendor by telephone. The vendor also used a manual problem tracking system. Our experience was that the elapsed time for problem response with the manual system was about one month.

MAINTENANCE

In general, maintenance of workstation computing equipment hardware may be done under contract by either the vendor or by an independent maintenance organization. Maintenance of the software may be handled similarly or it might be done in part by the workstation operator or owner.

Fixing hardware problems requires special training and an inventory of spare parts. It would be unusual for scientific and engineering staff to acquire the expertise to do their own hardware maintenance. In this project we have found the maintenance response and the expertise of the Ridge personnel to be quite adequate to handle problems. Factors such as cost, size, and numbers will influence how maintenance of workstations will be done in the future.

In this project, the company handled maintenance of the software. With telephone assistance from the company's California office, the Ridge system manager installed the new software. Early on in the project, before the company had hired and trained a systems engineer for the local office, the company supplied some software on floppy diskettes to fix software problems. Indeed, the availability of easily distributed media such as floppy diskettes and the low system cost may result in the owner doing much of the software maintenance, just as personal desktop computer owners now do much of their own maintenance.
DOCUMENTATION

To aid the user participants in learning the Ridge working environment, we prepared documentation to supplement that supplied by the company. The vendor documentation consisted of the Unix documentation set, the Fortran language standard document, a book on Pascal,\textsuperscript{17} and on-line help with the Unix \texttt{man} command.

We prepared nearly forty pages of draft documentation with the objective of describing the site-dependent considerations. The first part described the simplest commands necessary to compile, link, and execute a Fortran program. It also described who to call and what to do to resolve problems. The second part described communications procedures. There were step-by-step descriptions of how to operate our custom file-transfer programs. We also described the procedure to transfer files to the workstation via floppy disks written on a PDP-11 and also the methods to transfer files to the PDP-11. The third part described elements of the hardware, such as the front panel switches and the floppy disk drive, and how to use them. We intended a fourth part to describe the graphics library, but the vendor was unable to deliver such software during the project term. A number of appendices in the document described recommended reading, the PDP-11/70 in Computing Services, guidelines for workstation scheduling, locally written commands, the library of local subprograms, and the specifications for the timing routines.

GRAPHICS

As we stated above, the vendor was unable to deliver a user-level graphics library which would enable the participants to add graphics to their programs.

The Ridge system did have some graphics tools which would be useful for analysis tasks. These tools could accept data files, even directly from a running program, and produce certain limited kinds of plots on the high-resolution display. Although the system printer (Genicom 3404) is capable of doing graphics, there was no software to drive it. We verified that graphics could be displayed on Tektronics 4014 terminals; these same graphics should be printable on certain Versatec printers.

ACKNOWLEDGMENTS

The project to evaluate scientific workstations began under the leadership and guidance of Dave Snider, Manager of Systems Programming prior to his untimely death. Dave felt strongly that the Laboratory should lead the way in evaluating this new kind of equipment. We have carried out the project essentially as he planned it.

For this document, Goran Birgersson (Reactor Analysis and Safety) wrote Chapter 7: "SAS4A, a Program for Reactor Accident Analysis". Roger Blomquist (Applied Physics) wrote Chapter 8: "VIM, a Monte Carlo Reactor Neutronics Program". James Kennedy (Reactor Analysis and Safety) wrote Chapter 9: "STRAW, a Fluid-Structural and Thermal-Stress Finite Element Program". Dale Koelling (Materials Science and Technology) wrote Chapter 10: "The Electronic Structure of Atoms and Solids". Anthony Policastro (Environmental Research) wrote Chapter 11: Environmental Transport Modeling". Richard Raffenetti (Computing Services) wrote the remaining chapters.

We wish to thank Eugene Rackow (Computing Services), who helped to keep the workstation communication system functioning and who also wrote the custom software that made communication possible. He provided valuable assistance to vendor personnel in matters relating to our IBM computing environment. We wish to thank Debbie Coultis, Doug Engert, Fred Moszur, and John Schofield (all from Computing Services) for reviewing drafts of this document.

We have formatted this document with the University of Waterloo Script (Version 83.1) and Syspub text formatters.
Appendix A

PROGRAMMING TOOLS

The following sections describe tools that are useful for building scientific programs. They also provide examples of each tool.

PROGRAM MANAGEMENT

A program management tool serves to maintain an entire program. Scientific programs are large: they consist of many modules (subroutines and functions), and each module in turn may consist of many lines of source code. As changes are made to individual modules, certain parts of the program system must be brought up-to-date; that is, changed modules have to be recompiled, images must be relinked, and modules in libraries must be replaced. A program management tool enables the updates to be done without doing wholesale recreation of all secondary items (object modules, libraries, etc.) The Unix tool make is an example of an effective program management utility used by participants in the workstation project.

SOURCE CODE MANAGEMENT

Source code management tools are library utilities to manage text, but they may also have knowledge of the source code language. The better source code management systems permit the tracking of source code changes and retrieval of previous versions. The change history is kept to describe the versions, and the source code is often maintained as an original, plus the change modules that create the versions from the original. The Ridge Unix system has the SCCS (Source Code Control System), which consists of several programs. This system was not used in the workstation project.
MACRO PREPROCESSOR

A macro preprocessor is a program that accepts source code as input and does string replacements based either on fixed or on user-supplied rules. This tool is useful to maintain transportable source code that must be adjusted to the vagaries of each target computer's characteristics (e.g. word size) and language extensions (e.g. library functions). The Ridge Unix system has the preprocessors m4, a general macro processor, and ratfor, a special processor that has its own input language but that produces Fortran source code as output. Another well-known macro preprocessor is mortran.

COMPILER

The most important language in scientific and engineering programming is Fortran. The current Fortran language standard\(^\text{16}\) is also known informally as Fortran 77. No scientific workstation can be without a full-featured Fortran compiler. The Ridge system includes a Fortran compiler available through the command f77.

STATIC ANALYZER

Traditionally, programs are tested by compiling them and by running them. Compilers test syntax and program logic completeness and execution tests for logic errors that can be found with the test data. Static analysis tools scan source code to look for logic errors that run-time testing may not encounter. A Unix tool for static analysis is lint, which is for C language programs and not for Fortran. Static analysis tools for Fortran are available in the software market.

SUBROUTINE LIBRARY

A library of useful subprograms is essential for scientific programming. A library facility is necessary so that the modules can easily be linked to the program to form the image. There should be several kinds of subprograms that can be maintained in libraries as they would be in a shared computing system. Normally a mathematical library is present that contains the mathematical functions supplied by the Fortran language. Some libraries are available from vendors to do graphics, linear algebra, and other numerical procedures that are commonly used in scientific computing. LINPACK and EISPACK are examples of subprogram groups that are best maintained in libraries.

LINKER

The linker, which is sometimes called the loader, is used to connect all of the modules of a program. It builds the executable image, which then can be executed. A linker should have optional features to increase its flexibility. The ability to link a subset of a large program's modules is useful because it enables a user to select modules provisionally from libraries. The Ridge Unix system linker is invoked with the command `ld`.

RUN-TIME SYSTEM

A run-time system is necessary to execute program images. Any workstation system without a run-time system is not useful for scientific programming. Systems which do not have a run-time system are called "cross-development systems", and their purpose is programming for operating system development, not scientific programming. A cross-development system might be useful to develop scientific programs for a server system (for example, an array processor computer networked to the workstation). However, a scientist should be able to develop and test programs on the workstation (including executing them) as well as to have the cross system to develop programs to be run remotely.

DEBUGGING AIDS

A wide variety of debugging aids exist that we forget about when we have them. When they are missing, we realize how important even the simplest of them are. Dumps, tracebacks, the compiler, a static analyzer, the linker, and the run-time system are all used to locate errors in programs. A new tool for debugging is the symbolic debugger, which permits programmers to interact with their programs while they are executing. They may examine the contents of variables, change the values, single-step the program, etc. Additionally, there are programs available that read program dumps to provide information that may otherwise be difficult for a novice to extract. Indeed, much of dump reading is just an exercise in data conversion. A traceback is an example of an interpreted dump that yields a fixed output format.
BIBLIOGRAPHY


INDEX

Acquisition benchmark ... 14, 27, 34
Administration of a scientific workstation ... 77
Backup procedures ... 3, 12, 33, 60
Communication ... 3, 31, 42, 53, 57, 60-61, 66, 73-76, 79, (see also "Network communication")
Communications ports ... 13
Computing technology ... 1
Costs of computing ... 20-22, 49, 64, 67, 77
Data analysis ... 9, 16
Data management ... 9, 16
Debugging ... 4, 39, 42, 50-51, 54, 56-57, 59-60, 63, 66, 69-72, 83
Dedicated computing ... 5-6, 42, 51, 57, 64-66
Distributed computing ... 5-6
Document preparation ... 9, 17, 65, 73
File capacity of a scientific workstation ... 12
Fortran ... 16, 19, 23, 29-30, 35-36, 41, 45, 47, 51, 56, 60-61, 63, 67, 69-71, 79, 82
Graphics ... 16, 60, 63, 73, 79, 82
Kermit ... 31, 69, 75
Maintenance of a scientific workstation ... 78
Mathematical modeling ... 8, 39, 45, 53-55, 59, 62, 67, 74
Memory of a scientific workstation ... 11
Multiuser workstations ... 8
Network communication ... 5, 12-13
Operating system ... 14, 29-31, 33-34, 61, 63, 72
Pascal ... 79
Performance ... 28, 41, 43, 48, 56, 60, 62-63, 65, 69-70, 72, (see also "Processor speed of a scientific workstation")
Personal desktop computers ... 1, 59
Physical environment of workstation ... 23
Price of a scientific workstation ... 2, 7, 20-22
Printer accessibility ... 13, 77
Processor speed of a scientific workstation ... 11, 27, 29
Production computing ... 3, 9, 41, 48, 53, 56, 62
Productivity of users ... 3, 19, 27, 67
Program development ... 16, 42, 50, 56, 59-60, 65, 67, 81
Reduced Instruction Set Computer architecture ... (see "RISC architecture")
Reserving the workstation ... 22, 24-25
Ridge Operating System (ROS) ... 35-36, 42, 63, 72, 76
RISC architecture ... 2
Run-time system ... 6, 83
Shared computing ... 5
Software tools ... 6, 11, 15, 60, 65, 68, 73, 81
Text editors ... 15, 50, 65-66, 73
Unix ... 6-7, 15, 17, 29-31, 34, 54, 56, 60-61, 65, 68, 70, 72, 73-76, 78, 81-83
Distribution for ANL-84-100

Internal:

C. H. Adams  T. H. Dunning  R. A. Noland
L. W. Amiot  D. E. Engert  S. E. Ott
R. A. Bair  D. O. Hale  A. J. Policastro
P. J. Bertoncini  R. L. Herriford  R. C. Raffaletti
G. Birgersson  P. E. Hess  G. S. Roediger
R. N. Blomquist  J. M. Kennedy  G. S. Ross
R. G. Bucher  D. D. Koelling  F. O. Salter
M. K. Butler  B. J. Koprowski  C. G. Schlesselman
L. W. Carlson  R. F. Kulak  R. C. Schmitt
C. M. Caruthers (45)  S. C. Long (7)  J. E. Schofield
R. K. Clark  P. C. Messina  D. Y. Smith
W. J. Cody  R. J. McMahon  J. P. Unik
J. J. Dongarra  J. L. Midlock  M. Wastag
F. E. Dunn  F. M. Moszur

ANL Patent Department
ANL Contract File
ANL Libraries
TIS Files (6)

External:

DOE-TIC, for distribution per UC-32 (170)
Manager, Chicago Operations Office, DOE