γ

WSRC-TR-91-420

NRTSC
NUCLEAR REACTOR TECHNOLOGY
AND SCIENTIFIC COMPUTATIONS

KEYWORDS: SCIENTIFIC COMPUTING
DATABASE MANAGEMENT
GRAPHICS TOOLS
JOSHUA, J70, J80, J90
ASCENT
UNIX

RETENTION: PERMANENT

Received by OSTI
MAR 1 6 1992

PROGRESS REPORT
ADVANCED SCIENTIFIC COMPUTING ENVIRONMENT TEAM
NEW SCIENTIFIC DATABASE MANAGEMENT TASK (U)

Project Manager: J. P. Church

Work Done By: J. C. Roberts
R. N. Sims
A. O. Smetana
B. W. Westmoreland

ISSUED: JUNE, 1991

M. R. Buckner — 8-22-91
Derivative Classifier — Date

SRL SAVANNAH RIVER LABORATORY, AIKEN, SC 29808
WESTINGHOUSE SAVANNAH RIVER COMPANY
Prepared for the U. S. Department of Energy under
Contract DE-AC09-88SR18035

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED

WSRC-TR--91-420

DE92 009653

DOCUMENT: WSRC-TR-91-420

TITLE: PROGRESS REPORT
ADVANCED SCIENTIFIC COMPUTING ENVIRONMENT TEAM

TASK: NEW SCIENTIFIC DATABASE MANAGEMENT

APPROVALS

M. R. Buckner 8-22-91

M. R. BUCKNER, MANAGER DATE:
SCIENTIFIC COMPUTATIONS SECTION

[illegible] 08/12/91

R. R. BECKMEYER, MANAGER DATE:
COMPUTING TECHNOLOGY GROUP

MASTER

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED

Westinghouse Savannah River Company
Savannah River Laboratory

WSRC-TR-91-420

PROGRESS REPORT
ADVANCED SCIENTIFIC COMPUTING ENVIRONMENT TEAM NEW SCIENTIFIC DATABASE MANAGEMENT TASK

EXECUTIVE SUMMARY

OBJECTIVE

The mission of the ASCENT (Advanced Scientific Computing EnvironmenT) Team is to continually keep pace with, evaluate, and select emerging computing technologies to define and implement prototypic scientific computing environments that maximize the ability of scientists and engineers to manage scientific data. These environments are to be implemented in a manner consistent with the site computing architecture and standards and NRTSC/SCS strategic plans for scientific computing.

The major trends in computing hardware and software technology clearly indicate that the future "computer" will be a network environment that comprises supercomputers, graphics boxes, mainframes, clusters, workstations, terminals, and microcomputers (i.e., a full complement of clients and servers). This "network computer" will have an architecturally transparent operating system allowing the applications code to run on any box(es) supplying the required computing resources (e.g., cycles, storage). The environment will include a distributed database and database managing system(s) that permits use of relational, hierarchical, object oriented, GIS, et al, databases.

To reach this goal requires a stepwise progression from the present assemblage of monolithic applications codes running on disparate hardware platforms and operating systems. The first steps include: (1) converting from the existing JOSHUA (J70) system to a new J80 system that complies with modern language standards, (2) development of a new J90 prototype to provide JOSHUA capabilities on Unix platforms, (3) development of portable graphics tools to greatly facilitate preparation of input and interpretation of output; and (4) extension of 'Jvv' concepts and capabilities to distributed and/or parallel computing environments.

Step (1) is nearing completion. The new J80 system has been implemented on the VAXcluster and the unclassified and classified IBM mainframes. Users of the J70 system are being ported to J80 concurrent with satisfactory conversion of the application codes. Step (2), the prototype of the J90 system, is complete. Execution of that system has been successfully ported to Sun SPARCstations, IBM RS-6000 workstations, and the Cray XMP-EA running UNICOS®. The distributed database and computing features of J90 will be added to the J80 system when conversion from J70 is complete. Step (3) is well underway with major new graphics capabilities already provided for reactor facemap applications. Work on Step (4) has just recently begun. The prototype J90 is being extended to provide for network (multiple CPU) distributed and parallel execution of modules.

1.0 INTRODUCTION

The mission of the ASCENT (Advanced Scientific Computing EnvironmenT) Team is to continually keep pace with, evaluate, and select emerging computing technologies to define and implement prototypic scientific computing environments that maximize the ability of scientists and engineers to manage scientific data.

Two tenets guide the efforts of the ASCENT Team: (1) the purpose of computing is insight, not numbers[1], and (2) the purpose of accumulating knowledge is to use that knowledge to think[2]. That is, we seek to facilitate understanding. To this end the broad, long term, goal of the ASCENT Team is to provide a computing environment that will let the scientist/engineer function at the higher level of abstraction that is his actual area of technical expertise. The scientist/engineer should be able to solve problems by interacting with conceptual representations drawn directly from the scientific and engineering domains. In this environment the scientist/engineer (i.e., the "problem solver") builds the problem model with reusable virtual objects having associated attributes and behaviors, including any real or artificial constraints. The problem solver could then test the model by perturbing it interactively and observing quantitative (archived experimental measurements; simulated or computed data) and/or qualitative (trends, approximations) responses. Such an environment would greatly facilitate the solution and understanding of scientific and engineering problems.

To provide this environment the Computing Technology Group of Scientific Computations Section has assembled a dedicated ASCENT Team whose focus is to incorporate into the SRS computing environment the appropriate developments in computational hardware and software that trail only slightly the leading edge of technology and to do so in a manner consistent with the site computing architecture and standards and NRTSC/SCS strategic plans for scientific computing.

2.0 DISCUSSION

2.1 Overview

2.1.1 *ASCENT Locus*

Basically, the ASCENT Team tries to recognize the trends of computing hardware and software technology, understand the current state of SRS scientific computing, be aware of the constraints imposed by emerging site computing architecture standards, map out a flexible plan to move towards the desired computing environment, and implement that plan.

The approach of the ASCENT Team to this task is to define and continually update a five-year strategy that leads always in the direction of the goal. The Program Plan that

---

[1] R. W. Hamming. "Numerical Methods for Scientists and Engineers", Chapter N+1, 2nd ed., McGraw-Hill, Inc., New York, 1973.

[2] This tenet, too, is not original with our team. Unfortunately, the specific reference is lost.

defines the first steps of this strategy has been recently published.[3] The ASCENT Team has very limited resources and therefore must be very selective in choosing its tasks. Specifically, each direction explored must be both a step towards the five-year strategy, and also provide an immediate application to serve as a test bed for the concept being pursued.

One view of where the scope of data management and the ASCENT team fits into the scientific computing scheme at SRS might be represented by Figure 1. In this view the center circle represents the hardware, network, and operating system, while the outer circle bounds the environment of the end-user. The user needs to communicate with stored data directly as well as through the use of applications codes. The area allocated to data management, bounded by the heavy lines, represents the responsibility of ASCENT. The interaction between the database and users and applications is via some kind of Applications Programmer Interface (API).

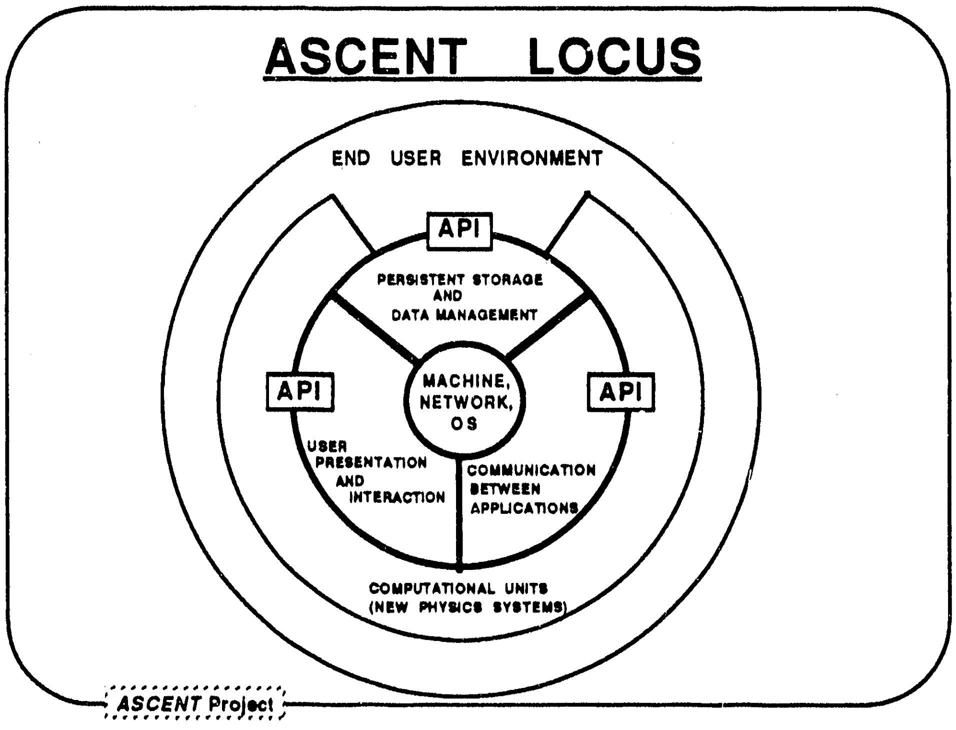

ASCENT LOCUS
END USER ENVIRONMENT
API
PERSISTENT STORAGE AND DATA MANAGEMENT
API
MACHINE, NETWORK, OS
API
USER PRESENTATION AND INTERACTION
COMMUNICATION BETWEEN APPLICATIONS
COMPUTATIONAL UNITS (NEW PHYSICS SYSTEMS)
ASCENT Project

Figure 1. Interface of ASCENT Team with Scientific Computing at SRS

2.1.2 *Trends of Computing Hardware and Software Technology*

The major trend in computing hardware is vastly more cycles, speed, memory, storage, etc. for much less money. Individual desktop workstations today have much more power than mainframe computers of only a few years ago. Network technology (data transfer rates, network system management, etc.) has advanced in parallel so that the

[3] J. P. Church. Advanced Scientific Computing Environment Group, New Scientific Database Management Task Program Plan. WSRC-TR-91-70 (February, 1991).

future "computer" will be a network environment that comprises supercomputers, graphics boxes, mainframes, clusters, workstations, terminals, and microcomputers (i.e., a full complement of clients and servers).

To go along with these hardware developments the computing software technology will provide an operating system that is architecturally transparent, thereby permitting the applications code to run on any box that supplies the required computing power (memory, cycles, whatever). The environment will include a distributed database, with perhaps local caching, with a (perhaps multiple) database managing system that permits use and manipulation of relational, hierarchical, object oriented, GIS, et al, databases. Other software developments will include well specified guidelines for modular software development that facilitate assembling complex problem models from simpler components and software reuse.

Actually, some of these features are not merely trends towards the future, but instead are already available in prototype, or even production (albeit early versions), form.

2.1.3 *Current State of SRS Scientific Computing*

The scientific computing environment at SRS is truly a 'mixed bag'. Most of the reactor physics codes, and some of the thermalhydraulics codes, used to design charges for SRS reactors are written to operate under the JOSHUA system.[4] The JOSHUA system is a shared database system with input and output via named data records and was originally developed at SRS in the 1970s for scientific code development. The JOSHUA system consists of both an operating system for data management, job execution, and terminal facilities, and a system of applications codes developed to use these facilities in solving problems at SRS. For purposes of discussion, this system will be referred to as J70.

Initially, the J70 computing environment was state-of-the-art, and presaged much of the software technology subsequently developed years later by commercial vendors. To obtain satisfactory performance many special assembly language programs were developed for data management and FORTRAN input/output routines were extensively modified. Although the applications codes still are critical for analysis of heavy water reactors, the system is not well suited to a modern computing environment. The codes are not portable because they do not comply with modern language constraints, and the operating system is dependent on a specific mainframe architecture and compiler.

Several years ago it was recognized that this situation could not continue indefinitely, and a program was started to rewrite the system to comply with standard FORTRAN 77 language specifications. That compliance would provide portability across architectures and provide a base system that would be fertile ground for prototyping and/or incorporating modern technologies. Convergence to this newer system, called J80, is underway and will be discussed below.

Many other codes, particularly those obtained from offsite or developed recently, are not part of the JOSHUA system. These include compute intensive thermalhydraulics

[4] H. C. Honeck. The JOSHUA System (U). DP-1380 (April, 1975).

codes, molecular modeling codes, seismic analysis codes, etc. Some, but not all, comply with FORTRAN 77 standards, some run on high-performance workstations, some run on mainframes under non-JOSHUA operating systems, etc. These codes, and codes now being planned, must be integrated into the advanced computing environment along with the J80 codes discussed above, once the J70-to-J80 conversion is complete.

2.1.4 *Site Computing Architecture Standards*

Site computing architecture guidelines have recently been published[5] and migration plans to move towards the proposed architecture are being drafted. The major features of the proposed architecture are: standards-based computing, data driven systems, workstation orientation, automated applications development, relational database technology, client/server technologies, emphasis on buy vs build for applications, user-based data reporting tools, modularity and object oriented programming, and site-wide coordinating organizations.

2.1.5 *Flexible Plan*

To reach our long term goal will require progressing from the present assemblage of monolithic applications codes running on disparate hardware platforms and operating systems. The initial steps of this progression must also acknowledge the progress to date in developing the 'new' J80 system to replace the J70 (JOSHUA) system.

As discussed above, the initial ASCENT Program Plan has been issued.[6] Initially, there were four main parts defined for the project: (1) completing conversion from J70 to J80, and (2) development of a J90 prototype that operates on Unix platforms , (3) development of portable graphics tools, and (4) extension of 'Jvv' concepts and capabilities to include distributed and/or parallel computing and object oriented environments. Part (1) of the project is nearing completion, Part (2) was completed, Part (3) is well underway with significant new capabilities already provided to the applications community, and Part (4) is in the initial stages of investigation.

A fifth part of the plan was to initiate training to implement and disseminate the above tools and methods. That training has begun on a limited basis for J80 and graphics applications. The more formal training originally planned will have to wait until the trainees are less consumed by reactor startup activities.

2.1.6 *Implementation*

Implementing the proposed environment at SRS requires a critical mass of qualified technical people who can devote the majority of their time to implementing (as opposed to developing) the technology. (Implementation includes QA testing of prototypic and production environments.) That organization does not yet exist as a separate entity at SRS, and the ASCENT Team is trying to provide that function temporarily.

---

5 Savannah River Site Computing Architecture (U). WSRC-IM-91-18-1 (April 15, 1991).

6 J. P. Church. Advanced Scientific Computing Environment Group, New Scientific Database Management Task Program Plan. WSRC-TR-91-70 (February, 1991).

The discussion below will provide further details on the progress to date on implementing the Program Plan.

2.2 JOSHUA Development

2.2.1 *J70 Concepts, Development, and Status*

The JOSHUA system is called 'J70' in the ASCENT reports because this new name helps clarify discussions of the migration from the original system to the newer versions. The environment of the J70 system is shown in Figure 2. The main point is that users, via terminals, and applications codes interact with (i.e., read from, and write to) and have a single view of the database.

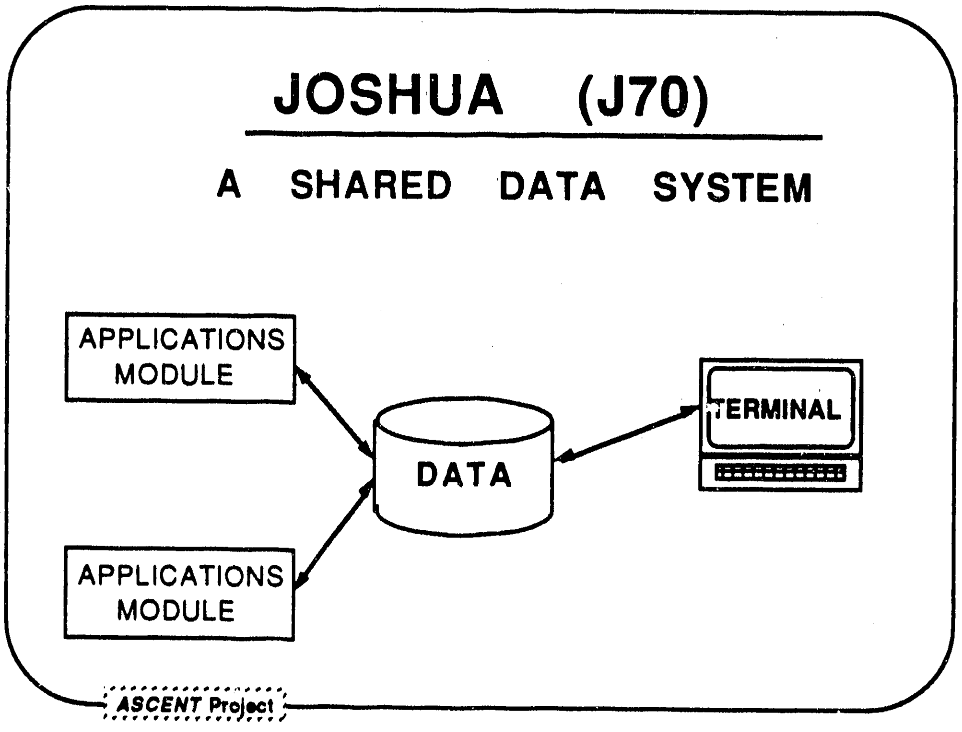


Figure 2. Simplified View of J70 Computing System

The computing system is tightly coupled: a single mainframe computer with users, terminals, and applications codes all accessing a common database. The database consists of named data records existing in either User, Standard (STD), or Job data sets. At execution the applications codes can select data from any of these data sets by following a read hierarchy established by the user at the time the job is submitted. The user can create, modify, or read these records via corresponding templates. If we superimpose the J70 environment on the model of a computational system shown in Figure 1 we get something like that shown in Figure 3. The data management for J70

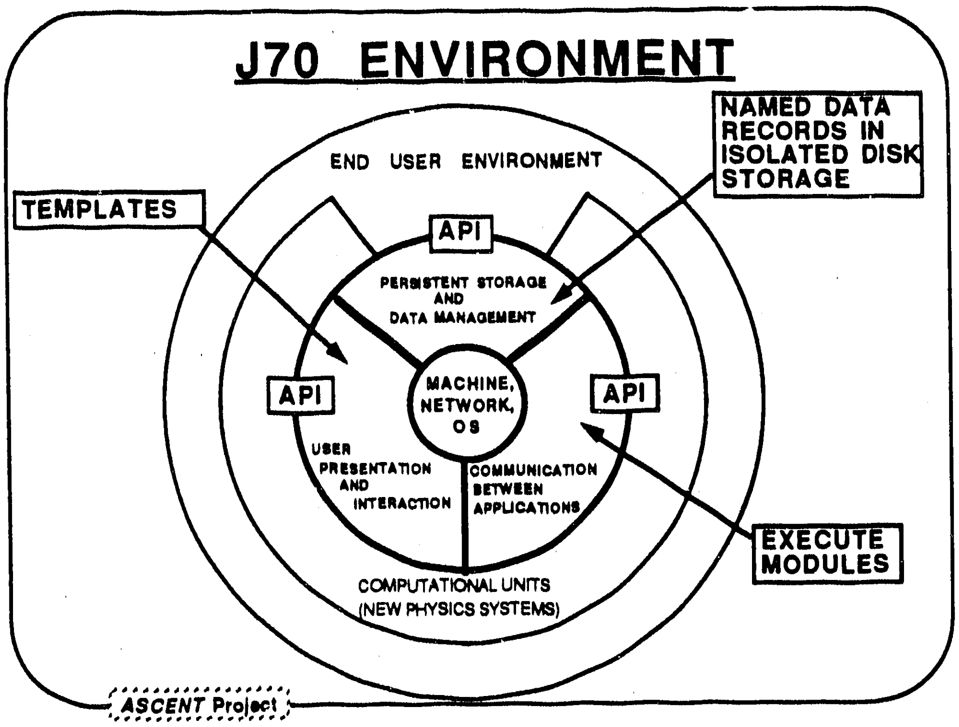

J70 ENVIRONMENT
NAMED DATA RECORDS IN ISOLATED DISK STORAGE
END USER ENVIRONMENT
TEMPLATES
API
PERSISTENT STORAGE AND DATA MANAGEMENT
API
MACHINE, NETWORK, OS
API
USER PRESENTATION AND INTERACTION
COMMUNICATION BETWEEN APPLICATIONS
EXECUTE MODULES
COMPUTATIONAL UNITS (NEW PHYSICS SYSTEMS)
ASCENT Project

Figure 3. Superposition of J70 Computing Environment on ASCENT Model

has the named data records in isolated disk storage. Following the terminology of the above model, an example of interapplication communication is the ability in J70 of one applications code module to execute another module. And, as indicated above, in J70 the templates serve as a limited API (perhaps more properly called a UI; i.e., User Interface) for user presentation of, and interaction with, the database.

J70 was state-of-the-art when it was developed. It fulfilled it's scope for the 1970's; and what it did, it did very well. But, in fact, it no longer does some of the things it used to do. As the operating system and compilers changed through the years, some of the capabilities of J70 were not maintained. The development of J70 software stopped years ago. A result is that features such as interactive graphics, which was promised for 1975, were never developed. Finally, J70 is tied to an outdated compiler which the planned replacement mainframe will not support.

A project was initiated several years ago to convert the J70 system to one that complies with modern language standards (FORTRAN 77) and, hence, is portable to other platforms. Part (1) of the ASCENT Team program is to complete the task of converting the applications codes and existing database from J70 to the new J80 system. This task is an integral part of the ongoing SRS effort to document the verification and validation of, and to subsequently certify, all existing critical reactor physics software.

2.2.2 *J70=>J80 Conversion*

2.2.2.1 <u>J80 Systems Development</u>

2.2.2.1.1 *Background*

To reiterate some points made earlier, J80 offers the same basic functionality as J70. J70 was based on FORTRAN IV, an IBM specific version of the FORTRAN 66 language, while J80 is compliant with FORTRAN 77 language standards. J70 runs only on the IBM, while J80 runs on both the IBM mainframes and the VAX clusters. J80 uses the operating system to do many of the things that required special data management routines in J70. Hence, the need for specialized assembly language coding was minimized in J80. Depending on how it is assessed, some 20-65% of the J70 operating system was written in assembly language, while less than 1% of J80 coding is assembly language.

The J80 system is getting its first significant usage by customers (Reactor Physics, Reactor Technology, and NPR personnel) since its initial development. In general, the system is performing well and has received favorable comment from the users. Most problems that have occurred have been traced to causes unrelated to either the J80 system or the conversion of J70 codes to that system (see below regarding the GLASS code). Initial documentation of the J80 system[7],[8] is being updated.

Responding to feedback from new users of the J80 system, numerous system changes have been made to improve the user friendliness of the system and to add several new features. A major enhancement was made to the J80 terminal system to include the Reactor Map function which is necessary for creating the input records for GRIMHX. This function is a text-based graphic equivalent to one now existing in J70.

2.2.2.1.2 *J80 Improvements on the IBM Mainframe*

2.2.2.1.2.1 <u>Permissible Problem Size</u>

The size of problems that can be run in J80 was greatly increased by taking full advantage of virtual memory features. Jobs can now be run on either the IBM or the VAX using up to 2 gigabytes of memory. This is much larger than possible with J70.

2.2.2.1.2.2 <u>Faster Execution Speed</u>

One of the reasons for much of the assembly language coding in J70 was to obtain satisfactory performance of I/O to disc access. Special steps had to be taken in J80 to obtain comparable performance. In the initial version of J80 for the IBM, the CPU time used by an applications code was about the same as in J70, but the elapsed time used by J80 was much greater than that used in J70. (No comparison of timings is available for the VAX because J70 runs only on an IBM.) This increase in elapsed

---

[7] W. H. Reed, et al. The JOSHUA User's Manual. DPSPM-GEN-36 (Sept., 1987).
[8] J. T. McCort. The JOSHUA Programmer's Manual. Draft (Aug., 1988).

time was so onerous that conversion of users from J70 to J80 would have delayed critical reactor physics calculations required for reactor startup.

The first step to speeding up the IBM version was to replace the J80 dynamic allocation utility routines with IBM's VS FORTRAN supplied routines. This allowed the use of multiple I/O buffers which improved the efficiency of J80 I/O processing. Also, these changes made it possible to use the latest code analysis tools and thus determine that the long running times for J80 were due to differences in the way direct access I/O is handled between the two systems.

IBM FORTRAN requires direct access files to be formatted before they are used which means writing the format pattern to every record in the file. Each J80 job has associated with it two scratch data sets: one for the system and one for the user . Because each J80 data set consists of two direct access files (a tree file and a data file), the two scratch data sets require four files to be allocated and formatted. This is very time consuming. The scratch data sets are inherently temporary, so they were moved from disk to VIO (virtual I/O). The scratch data sets must still be allocated, but the formatting is now I/O to memory which is much faster than doing physical I/O to a disk drive.

Also, standard FORTRAN direct access on the IBM requires the disk files to be fixed length and unblocked. This means that for each record read or written to the file, a physical I/O is made to the disk. If a job does a lot of I/O, then much of the time is spent simply waiting on the disk. Permanent data sets were moved to VSAM (virtual storage access method) files which are effective blocked. This means that when one record is read, a block of records are actually read off the disk. When the next record is read, the system checks to see if that record is already in memory. If it is, the record is returned to the program without having to do a physical I/O to disk.

These changes, along with several other minor changes (such as increasing the number of buffers allocated to each file) have reduced the running time of J80 on the IBM to be comparable to that for J70.[9] With the improvement in computing timings, there is no longer any reason to delay the conversion of codes and users from J70 to J80.

2.2.2.2 *Code Conversion*

As stated above, J70 consists of a data management system, or operating system, and a set of applications codes that operate under that system. A key step to fully implementing the J80 system is to convert to J80 format all the J70 codes. The newer language standards are more restrictive than those existing for the original development. Furthermore, because of the complexity of the J70 codes and, sometimes, ingenuity of the original J70 programmers, the task of conversion does not lend itself to a rote type conversion. However, there are some guidelines[10], the most obvious of which are as follows:

[9] A. O. Smetana. Improvements to New JOSHUA (J80) on the IBM/MVS Operating System. WSRC-TR-91-118 (March, 1991).

[10] J. H. Key. Conversion of Old JOSHUA to New JOSHUA . SCS-CTG-900032 (August 9, 1990).

1. Data Typing:
   - Data typing must be separated from data initialization.
   - Data types must be renamed (e.g., 'REAL*4' becomes 'REAL', 'REAL*8' becomes 'DOUBLE PRECISION').
   - Some data types must be changed (all integers are 4 bytes, and 'INTEGER*2' is not allowed.

2. Character variables:
   - Character variables must be declared as such (vice 'REAL' in J70)
   - Character and numeric variables can't appear in the same COMMON statement.

3. Equivalence statements must be checked for validity after data typing and character variable changes

4. Calls to some system routines must be replaced to use new J80 routines.

5. Miscellaneous syntactical changes

The application of the above guidelines to convert the J70 codes is well underway. However, progress in this area is being severely hampered by manpower restrictions because most J70 code proprietors who would normally be doing the code conversion are heavily involved in reactor restart activities. Specific progress is noted below for some of the more important codes. It should be noted that many of the tools used in debugging the converted codes are finding errors in coding that exist in the original J70 codes. Generally, these errors are remnants of code lines which are never accessed, or data not initialized. But it is possible that some of the bugs may be more serious. The net result is that the J80 versions represent the best available version of each respective module. These latest versions have also been supplied to offsite contractors involved in either verification and validation work for SRS or design work for the New Production Reactor.

2.2.2.2.1 *GLASS Code*

Reactor Physics used the NJOY code (developed offsite) to produce a new MULTIGRP library for use with GLASS. Use of the new library uncovered an error (now corrected) in the CRISP module of GLASS. The error caused CRISP to create an incorrect energy band structure which then caused an obvious fatal error in another part of the GLASS execution. The error occurred only with the new library and had no effect for the standard libraries in use at SRS

All modules of GLASS have been converted to J80 and checked out with the exception of the Monte Carlo option. Checkout of the latter coding is awaiting verification of the random number generator by the Criticality Methods and Analysis Task Team (see Section 2.2.2.2.3). The CRISPE and FISTH modules have been installed on the IBM J80 system and are also being checked out. These modules are executed by GLASS but only when GLASS is executed by the criticality module KOKO.

2.2.2.2.2 *GRASS Code*

The GRASS modules necessary for running the GRIMHX code have been debugged and made available under the J80 system. To make the GRASS modules conform to the FORTRAN 77 standard, the dynamic memory allocation functions were replaced with statically dimensioned arrays. This meant that every job, regardless of size, would require as much memory as the largest job that would ever be run. This is normally of no concern on virtual memory computers, but the IBM requires special steps to gain access to memory beyond approximately 8 megabytes. The GRASS modules (as well as the J80 system - see above) were rebuilt to take advantage of this extra memory. This means that jobs can now be run that use up to two gigabytes of memory available on the IBM as well as the VAX.

2.2.2.2.3 *Criticality Codes*

J80 versions of criticality modules ANISN, KOKO, HRXN, POOHBAH, KENO, and KENOCJ, are currently being checked out by the Criticality Methods and Analysis Task Team. Suitability of the random number generators (used the Monte Carlo codes) on various J80 platforms is also being assessed. The remaining criticality modules should soon be available under J80.

2.2.2.2.4 *JASON Code*

The JASON code is converted and ready for initial testing by the code custodian in the Reactor Physics Group. Installation of J80 on the classified computer is underway and will facilitate checkout of the converted code.

2.2.2.2.5 *Transient Analysis Codes*

The AA3 code was converted to the J80 system and initial testing of the converted code is complete. Formal approval by the code proprietor is underway. The code has been installed in the Software Configuration Management System and final testing of this SCMS version is underway.

The modules necessary for running the module TRIMH2 were installed under J80. These modules have undergone extensive debugging, and the code now appears to be converted satisfactorily. Final testing is underway before installation in the SCMS.

2.2.2.2.6 *SHIELD Code*

All the SHIELD modules have been converted to J80, and enough of them have been installed under IBM J80 to enable one computation path through the SHIELD system. A test problem that exercises that path was provided by the code proprietor and the results are equivalent to those obtained with the J70 version.[11] Verification of the remaining J80 SHIELD modules is underway.

2.2.2.2.7 *PORAD Code*

Assistance was provided Reactor Physics in converting the application code PORAD from J70 to J80. With only a few hours of help from ASCENT, the Reactor Physics code proprietor was able to convert the code within a few days. This is much less than the originally estimated conversion time and offers the hope that conversion of other codes by the code proprietors might also proceed much faster than earlier estimates when they are able to consider tasks other than those directly related to reactor restart.

2.2.2.3 *QA Procedures*

QA documentation for the J70=>J80 conversion of applications codes was developed and approved to define a procedure to obtain formal acceptance by the code proprietors that the conversion is successful if the J80 results are within acceptable agreement of the results obtained with the J70 codes.[12] Code proprietors are, in general, responsible for further certification of the code, and for all associated QA documentation.

2.2.2.4 *Training*

Several training sessions for J80 were held for Reactor Physics, Reactor Safety Research, New Production Reactor, and Reactor Technology personnel to facilitate the transition from the J70 to the J80 system. A specific topic was to demonstrate how to use J80 and run codes such as GLASS under J80. A videotape was made of the last training session for later use by the resource center as initial instructional material for users of J80.

A procedure was developed by CTM to provide a record of the daily usage (codes and users) of J70. These data are being analyzed to determine the most important needs for further J80 training and J70-to-J80 code conversion. Also, a login message similar to that used in J80 is being displayed for J70 users to inform them about impending cutoff dates for usage of J70 codes. Such dates will be set after the J70 users of those codes have been given appropriate J80 training and their J70 data sets have been successfully exported to the J80 system.

As new graphics applications are added to J80 (see below) additional training will be provided to facilitate their efficient use.

2.2.2.5 *Platform from which to move forward to new features*

A key feature of the J80 product is that the compliance with modern language standards provides a base on which to build a variety of new structures. The

[11] Identical agreement may not be possible because of differences in systems libraries.

[12] J. P. Church. Conversion of JOSHUA Applications Codes to J80. Computational Task Problem Statement and Solution Proposal (PSSP) Identification, Approval and Distribution Form. Task 91-016-1, Revision 0 (February 22, 1991).

portability of the standard language permits easy porting to other platforms. This, in turn, opens new avenues to graphics applications, and generally allows the SRS scientific analyst to better take advantage of the power of emerging hardware and software technologies. Two areas of development are the J90 system discussed next in Section 2.2.3, and new graphics applications discussed in Section 2.3 below.

2.2.3 *J90 (Unix Based JOSHUA for the 90's)*

One way to greatly increase the computing capability at SRS is to move to a totally distributed computing system (parts of a problem being run on different hardware, accessing information from local and/or global databases). The first step in developing such a system is to focus on an essentially hardware transparent system. A result of this functionality is to be able to immediately take advantage of new compute cycles as soon as they become available. No coding or data changes by applications code users will be necessary.

A prototype of a new version of the JOSHUA scientific data management system was completed.[13] This prototype, named J90, is targeted for the UNIX operating system based family of computing platforms that are expected to comprise our scientific computing environment in the 1990's. The implementation uses UNIX-supplied capabilities to replace the internal J80 system operations. The emphasis is on simplicity and standards such that JOSHUA implementation will not negate the emerging power and usefulness of advanced computing technologies.

The goal is to run any J80 application, without changing the present IBM and VAX source code and without moving and/or changing the existing JOSHUA data base, on any UNIX OS based platform having direct LAN connection and support for common network application protocols. Aspects of performance, robustness, reliability, and security are deferred for later production versions of J90 that may be developed.

The J90 system is also a point of demarcation from normal JOSHUA concepts and opens the way, eventually, for a smooth transition to an environment in which a network of heterogeneous platforms provides the compute cycles and a distributed database is accessible from any node on the network.

2.2.3.1 Programming Considerations

Six major programming constructs are addressed: (1) dataset hierarchies and name trees, (2) formatting of records, (3) module execution, (4) file manipulation, (5) library routines, and (6) command language.

2.2.3.1.1 *Dataset Hierarchies and Name Trees*

The UNIX file system for management of JOSHUA named records was used. The concept of a dataset hierarchy, a basic component of J80, is a subset of the function provided by UNIX file systems extended with "viewpathing" such as Sun's Translucent

[13] R. N. Sims. J90, A Prototypic Unix-Based JOSHUA System. WSRC-TR-91-421 (in progress).

File System (TFS) or AT&T's 3-D File System. Eduardo Krell of AT&T Bell Laboratories, an author of the 3-D File System, has been contacted regarding source code licensing for possible future enhancement/extension of J90 and related systems.

2.2.3.1.2 *Formatting of Records*

Sun's XDR (eXternal Data Representation), the source code for which is publicly available, was used for encoding and decoding of JOSHUA's binary data records as portable UNIX files.

2.2.3.1.3 *Module Execution*

The J90.1 implementation uses a simple UNIX execution model which restricts execution to a single CPU; i.e., the single machine where the JOSHUA monitor is started.

2.2.3.1.4 *File Manipulation*

The file utility package distributed by the Free Software Foundation was changed to provide a good library of file manipulation utilities for copying, moving, renaming, etc. Of special interest is the capability of these utilities to provide backup by means of file versioning.

2.2.3.1.5 *Library Routines*

A set of library routines was obtained to provide a "database" of capabilities much like that of Unix's termcap facility. This allows the generic handling of relatively static but externally specified information such as the association of dataset names with directory names.

2.2.3.1.6 *Command Language*

An embeddable command language processor was obtained. This will be used to provide what used to be thought of as the JOSHUA input stream where dataset/record manipulation and execution could be specified in a COMMAND, PARAMETER(S), S/E syntax.

2.2.3.2 Execution of Applications Programs IMPORT and GLASS on J90

The first application to be ported was the IMPORT code, which was then used to transfer datasets from the VAX under J80 to a SUN workstation under J90. All of STD (a major JOSHUA dataset) that is necessary to run GLASS was "imported". The source code for J80-supported GLASS modules was moved to a SUN workstation, precompiled, and compiled. The compiled pieces, i.e., individual subroutine object files, were linked to build GLASS executables (modules) and the system was then used to thoroughly test the core of J90 provided functions and to gauge the level of performance that can be achieved with the J90 methodology.

The results duplicated that obtained with J80/VAX/VMS within machine precision. Some specific considerations about the precompiler and data typing are noted below.

2.2.3.2.1 *Precompiler Changes*

Command-line parsing and file handling of the J80 precompiler were changed for J90 to make the precompiler more convenient to use in the UNIX environment . (The raw precompiler assumes a very crude environment for its execution, and the changes made to the J90 version are similar to changes made to the raw precompiler in the J80-VMS version for the same purpose.) These changes had no effect on the precompiler-generated FORTRAN code

One other change to the precompiler (the only one that changes the FORTRAN output) changes the MAIN FORTRAN program into a SUBROUTINE. J90 has a standard main program that encapsulates what is commonly regarded as the FORTRAN module, i.e., the applications code. In J90 the FORTRAN entry to the module is always named SUBROUTINE JSYAJS (external symbols prefixed by 'JS' are reserved for exclusive use by the J80 or J90 systems.) This naming is in acknowledgement of "yet another JOSHUA system".

2.2.3.2.2 *Data Typing*

Moving the current J80 GLASS to J90 uncovered a problem that, temporarily at least, affects the ability to run applications without changing the J80 source code.

In the latest version of J80 GLASS, INTEGER and REAL data are equivalenced for use by MANAGE (a set of subroutines used by each GLASS module to manage the in-core representation of the data array and to manage the I/O for the LIDS records). The coding for MANAGE had to be changed because J90 requires separate handling of differing data types. Although some GLASS modules read/write LIDS blocks directly and the same equivalences may appear outside of MANAGE, problems other than those identified in MANAGE were ignored.

A sample GLASS problem has been run successfully with J90. Because significant code restructuring would be required in GLASS to remove data type "lies" (equivalencing integer and real variables), the "NO_XDR" option in J90 was used to run the present GLASS without changing any source code. This option tells J90 to not encode/decode the data in a portable binary format.

2.2.3.3 Extension of J90 to Other Platforms and Performance Testing

The J90 system, which was developed on a Sun SPARCstation running SunOS, was then ported to an IBM RS-6000 workstation running AIX and to the Cray XMP-EA running UNICOS®. For all these ports, the GLASS code was successfully run with the source code essentially unchanged from its current J80 IBM and VAX coding. It should be noted that data portability doesn't always mean encoding/decoding or translation. The Sun and IBM RS-6000 platforms have a relatively standard and common (to many workstations) representation of binary data, i.e., char's, int's, and float's. This permitted using a copy of the Sun's binary database directly on the IBM RS-6000 without exporting/importing any data.

The test problem was a simple bi-sep pattern comprising a control cluster with a Mark 5E. Most of the problem time is spent in CRISP, CREEP, and TRAMP, with TRAMP getting around 80% of the CPU time. All platforms produced essentially identical results. The J80 times are shown below for reference for execution on the SLLAB2 node of our VAXcluster, i.e., a VAX 8810 running VMS, and on the IBM mainframe, i.e., an IBM 3090 running MVS.

		Timing Results in MIN:SEC*		
Platform	Version, Jvv	CPU	SYSTEM	ELAPSED
Sun SPARC2 (local)	J90	6:06	:55	10:44
Sun SPARC2 (remote)	J90	6:03	:27	9:05
IBM RS-6000/530	J90	4:07	:08	4:50
Cray	J90	6:25	:10	9:30
VAX 8810	J80	17:23		24:33
IBM 3090 (MVS)	J80	2:36		9:54

* See text for discussion regarding caveats for these timing results

Runs on the Sun and IBM workstations were for completely free machines; i.e., there were essentially no competing processes. The Cray appeared to be getting some 60-70% of the machine (an early morning run and probably quite atypical of normal daytime elapsed times). The system conditions in which the VAX problem was run are unknown.

All of the executions except for "Sun SPARC2 (remote)" were for configurations in which the JOSHUA records (and module binaries) resided on local disks of the machine. For "Sun SPARC2 (remote)", only the paging space resided on the machine on which the problem ran. Everything in the remote case was "mounted" on the executing machine from a remote SPARC2 using NFS (network file system). The times suggest that performance of distributed file systems for support of these computations may be quite good, and that such remote operation incurs no penalty.

The Sun and IBM workstation runs were with full optimization. Available current and valid information on how to properly interface C and FORTRAN, what libraries to use, etc., was barely enough to make the Cray port run . Also, the J90 routines on the CRAY have some optimization and/or configuration error and are currently running there with no optimization.

2.2.3.4 *Status of J90 Development*

J90 provides easy sharing of JOSHUA data among networked platforms, direct access of binary JOSHUA data from platforms of diverse architectures, and opens the

JOSHUA data management mechanism so that all programs and utilities other than specific JOSHUA-coded ones can easily access JOSHUA data.

J90 is operable as a batch-like system on the Sun, RS-6000, and Cray XMP-EA platforms and has been demonstrated successfully in execution of a typical GLASS problem unchanged from its current IBM and VAX coding. Terminal facilities are not available for this J90.1 version (see Section 2.3.2.3 for discussion about the graphics desktop under development for both J80 and J90). The application code, IMPORT, will import to the J90 environment any IBM or VAX exported JOSHUA dataset.

All goals of the J90 effort were achieved. The more significant are:

- Elimination of almost all internal JOSHUA system operations in favor of native operating system supplied services
- Easy sharing of JOSHUA data among networked platforms
- Direct access of binary JOSHUA data from platforms of diverse architectures
- Opening of the JOSHUA data management mechanism such that all programs and utilities other than specific JOSHUA-coded ones can easily access JOSHUA data.

Documentation of the J90 development is underway[14]. The reader is reminded, however, that the present version of J90 is a prototype rather than a production system. It would take a major effort to turn this prototypic development into a robust production environment, and it would be premature to do so. For example, only one test problem for GLASS has been run on J90. Although this test problem is typical and representative of a significant usage of the code, it does not exercise all possible paths through the GLASS module. Thus, no claims are asserted about complete validation and verification of the J90 system for the GLASS module.

Further, it appears that it may take a significant effort (~ 1 man-year) to rewrite the JOSHUA applications codes to ensure that all modules would be truthful about data typing (see Section 2.2.3.2.2). This, too, would be inappropriate because there is a separate program underway to develop improved replacements for many existing JOSHUA applications codes.

Additional development of the J90 system will continue as appropriate to test various concepts (see Section 2.2.4). As this development proceeds, and as new applications codes and methods are being developed, many of the benefits of the J90 program can be incorporated into the J80 system. The J80 system is already a production system with extensive testing and usage and already has an existing text-based terminal system. Thus, when the J70-to-J80 conversion is completed, J80 will be extended to include the file-sharing and distributed computing features developed for J90 as discussed next.

[14] R. N. Sims. op. cit.

2.2.3.5 *Extension of J90 Features to J80 and Future Directions*

As stated above, many of the benefits of the J90 developments can be incorporated into J80. For example, coding from J90 can be included in J80 to permit using a single copy of a distributed database for templated records and to enable a limited implementation of a distributed computing system. This implementation would require no changes in the applications codes if the data typing in templated records is truthful.

A limitation of this extended J80 system would be that a sequence of modules that share data as untemplated records, e.g. the LIDS records, would have to run on a single architecture (albeit any one of the CRAY, VAX, or IBM platforms). To obtain full implementation of a distributed computing environment would require the same data typing "truths" in J80 as J90, and hence require the same recoding effort of the JOSHUA applications codes as discussed above for J90.

This change from the program plan will result in more rapid introduction of some of the J90 advantages into the SRS scientific computing environment. J80 is already a robust production environment, and the extensions proposed would not demand the same effort to bring to production status as would a complete J90 system.

Also, note that the extended J80 would run on the CRAY, VAX, and IBM systems with their respective operating systems, while J90 would require a UNIX operating system on each machine. (Implementation of the J90 concepts is not as readily achieved in J80 on the IBM as on the VAX. However, the J90-UNIX system on the IBM mainframe may not be an efficient environment. That remains to be determined.). This extension of J80 will begin after the conversion from J70 to J80 is complete.

2.2.4 *Future Development Plans for J90 Computing*

Although J90 will not presently be implemented as a production system, further development will continue to test various concepts of an advanced scientific computing environment. For example, the present version of J90 is limited to executing on a single CPU. The next phase for development of J90 will produce a more generalized prototype for network (multiple CPU) distributed and parallel execution of modules. As J90 is extended to include these features, it will of necessity become more robust, just as occurred due to porting it from the Sun to the RS-6000, and Cray XMP-EA. That extension required reviewing all, and re-doing some, of the coding used for the initial development of the Sun Implementation. Just getting J90 to run on the three separate platforms resulted in a much more robust product.

As development progresses, attention will also be paid to improving the error handling (i.e., messages, codes, etc.) presently implemented in J80.

Some of the most exciting extensions to J90 can also be implemented into J80. These are the graphical interfaces discussed in the next section.

2.3 Graphics Tools and Applications

The ASCENT Team is developing a core set of graphics tools to be used in the scientific computing arena at SRS. These tools are intended to make application development easier, human interfaces more intuitive, and application codes more portable by separating the calculations from their input and output. The graphics tools are being developed using industry standards such as the C Language, X Window System, X Toolkit, Motif™ Graphic Tool Kit, and Unix. Each tool is intended to be the standard graphical user interface for the site and to provide the capability for applications output to be viewed from anywhere onsite.

Development of the first tool, a reactor facemap tool, is complete as discussed next in Section 2.3.1. This tool was then used in the subsequent development of two applications: the RMS/ARMS (Reactor Monitoring System, and its successor, the Advanced Reactor Monitoring System), and the FM function for both J80 and J90. These applications are discussed in Section 2.3.2.

2.3.1 *FaceMap Tool*

The first graphic tool completed, FaceMap, displays a reactor facemap[15, 16]. Coding for the FaceMap tool and a first draft of an Applications Programmers Interface (API) for that tool have been completed. Also, a patent disclosure has been filed for the FaceMap tool. The FaceMap tool has been evaluated by SRL-Human Factors including tests with certified Senior Reactor Operators to identify so-called 'human engineering discrepancies'.

The prototype was developed using the Motif™ Graphic Tool Kit, but the production version of the tool was written as a widget based on the Xtoolkit Intrinsics . This facilitated packaging the tool as a separate reusable entity that is distinct from the application. The resulting tool is independent of Motif™ and, thus, is more portable.

2.3.1.1 *FaceMap Features*

The FaceMap tool provides 'pointer tracking' and 'enter notify' capabilities.

'Pointer tracking' places a crosshair on the facemap at the center of the hex that the mouse pointer is in. The crosshair runs the length and width of the facemap. This makes it easier to determine which x,y location the pointer is in. Pointer tracking also highlights the particular hex by drawing a line around the outside of the hex.

'Enter notify' notifies the application when the mouse pointer moves to a new hex position. An example of a use of this feature would be an application which displays the online computer number corresponding to the position of the pointer.

---

[15] J. C. Roberts. "Interactive Graphical Reactor Facemap Tool. Patent Disclosure No. SRS-91-230 (May 23, 1991).

[16] J. C. Roberts. Reactor FaceMap Tool: A Modern Graphics Tool for Displaying Reactor Data. (To be presented at the 1991 Westinghouse Computer Symposium, Pittsburgh, Pa., October 21-22, 1991).

The tool can outline portions of the reactor facemap (positions, clusters, gangs, sectors and systems), display the facemap in grayscale as well as color, and produce PostScript output for printing the facemap. (To print a facemap in color without PostScript output formerly took about an hour. The ability to print in PostScript has reduced this time to five minutes. Also, using PostScript instead of printing via screen dump has greatly improved the readability of printed text.) The facemap will also print on an inexpensive black-&-white PostScript printer and will simulate grayscale.

2.3.1.2 *FaceMap Tool Documentation*

Documentation of the FaceMap tool is underway. This includes QA documentation, programmer's manual, user's manual, and an applications programmer's interface manual. The initial QA documentation (PSSP[17]) has been approved, and a draft of the Programmer's Manual is undergoing review. The Programmer's Manual consists of four chapters. The first chapter is an Overview of the X Window System and discusses everything from the X Window System architecture to building an X Windows application. The second chapter is an introduction to the X Toolkit, on which the FaceMap is based. The third chapter describes in detail what a widget (graphical object) is and how to use them in the X Toolkit. The fourth chapter documents each widget in the SRS Widget Set. Currently the only widget in this set is the FaceMap. Other widgets are planned and will be documented as separate sections to this chapter.

2.3.1.3 *X Window Platforms*

We continue to acquire, install, and test software and hardware which runs the X Window System. Presently, the FaceMap tool runs on DEC RISC, HP 700, and Sun SPARC workstations running their individual versions of Unix; VAX systems operating with VMS; and a Mac IIci with A/UX (the Mac X Window System must also be installed). FaceMap can be displayed on any box (including Mac's and PC's) having X server software.

2.3.2 *Applications*

ASCENT's charter is to develop production versions of a full range of graphical tools that can be integrated into applications codes by the respective code proprietors, developers, and analysts. But such integration requires new skills for most code proprietors. To facilitate this process, and to provide additional testing of the tools, ASCENT plans to implement each tool into at least one application of immediate interest to SRS. New applications that use the FaceMap tool are discussed below in Sections 2.3.2.2 - 2.3.2.4.

As part of the process of developing applications, graphical user interfaces (GUI's) need to be created. Software for facilitating this process is called a 'GUI builder'.

[17] J. C. Roberts. Computational Task Problem Statement and Solution Proposal (PSSP) Identification, Approval, and Distribution Form. Task Title: Reactor FaceMap Tool; Task Number 90-054-1; Revision 1 (October 30, 1991).

Such software is commercially available and is being evaluated as part of the ASCENT graphics task as described next.

2.3.2.1 *Graphical User Interface Builders*

A graphical user interface (GUI) builder is extremely useful software that facilitates construction of applications codes using graphic widget sets. GUI builders are being evaluated for use at SRS with the focus on enabling the use of the graphics tools being developed here. The GUI builders include ExoCODE®, XBUILD from Siemens Information Systems, Quest UIM/X from Quest Systems, and Builder Xcessory from Integrated Computer Solutions. The latter was used to design the interface for the new Joshua FM function discussed below (Section 2.3.2.3). Documentation of the evaluation is underway to describe the perceived strengths and weaknesses of the GUI builders and to recommend the best one(s) for use at SRS.[18]

2.3.2.2 *Reactor Monitoring System (RMS/ARMS)*

The Reactor Monitoring System (RMS) facilitates monitoring and interpreting reactor operation by providing for collection, storage, and retrieval of reactor operating data.[19] This system will be replaced by the Advanced Reactor Monitoring System (ARMS) now being developed by the ASCENT Team and the Reactor Physics Methods Development Team. ARMS will have new hardware and software to provide increased computational capabilities using a commercially available relational database that complies with proposed site computer architecture standards. When completed, ARMS will provide scientists, engineers, and managers with near real-time reactor operating data for monitoring current operations and predicting future operating conditions. Also, the ARMS QA documentation will comply with the present site requirements.[20],[21]

The display of the RMS data is presently done with text-based tables without any query capability. The FaceMap RMS application prototype provides a full graphic display of a facemap of temperatures, flows, powers, or any other dimension of the reactor assemblies. The RMS application code uses an early version of FaceMap that was not a distinct tool, i.e., in this prototype the FaceMap coding and RMS application coding packages are not cleanly separated from each other. This was consistent with the initial intent of prototyping the development The production code for the ARMS application will be rewritten to use the latest version of the FaceMap tool. This latest version is a reusable coding package distinct from the applications codes using this tool.

---

[18] J. C. Roberts. An Evaluation of Four X Interface Builders. WSRC-TR-91-471 (in progress).

[19] A. O. Smetana. Reactor Monitoring System User's Manual - VAX Version. DPST-86-621 (August 18, 1986).

[20] Nuclear Reactor Technology and Scientific Computations Department Quality Assurance Manual (U). 1Q-34 (January 1, 1991)

[21] A. A. Zagrodnik. QA Procedures for the Nuclear Reactor Technology and Scientific Computations Program Management Team. DPSTM-88-700-9 (August, 1988).

2.3.2.3 *Facemap Movie For K Reactor Data*

Another application using the FaceMap was built to analyze data obtained from tests of 'K' reactor. This also served as a tutorial for Reactor Engineering personnel who sat in on the development. (This was part of the training task implemented to disseminate to the SRS scientific personnel the ASCENT developments as described in the Program Plan.) The application was a simple movie generator which read input snapshots of K Reactor test data and displayed it at a rate of 2 frames per second. It is currently being enhanced by Reactor Engineering to display date and time and to have a color scale to display the values that correspond to particular colors.

2.3.2.4 *Graphic Reactor FaceMap (FM)Function in J80*

The latest FaceMap tool application under development is intended to replace the present text-based JOSHUA RM function. The RM function creates an image of a reactor facemap during charge design. That image is then processed to point to desired data records to describe a specific reactor charge for physics codes calculations. The present J80 RM function provides the same capability as the original J70 function. It is based on a text-based pseudo-graphic that is difficult to use, imposes unnecessary constraints on the user, and is prone to user error.

A new graphics function, named FM, is being developed to replace the RM function. The new FM function will be usable in either the J80 or J90 environment. Although the RM function will continue to be available in J80, it is expected that user productivity obtained with the new FM function will be so greatly increased, that the RM function will be obsolete less than 24 hours after implementation of FM. The goal is to enable a user to set up a charge design much more rapidly, by a factor of 10-20 for simple problems and as much as 100 times faster for very complicated problems, than now possible. Early tests show that goal is achievable.

The new function permits multiple axial-level reactor maps, creation of assembly types through an assembly editor, mapping between 2-character mnemonic label and corresponding GLASS record, cutting and pasting between axial levels, and cutting and pasting of positions, clusters, sectors, systems and gangs.

The new FM function has five panels controlled by the user:

1. FaceMap Panel to display the facemap and status information.
2. Assembly Palette Panel to display a list of the current assemblies that can be put into positions on the facemap
3. Assembly Editor Panel to create assemblies by specifying a name, type, mnemonic, color, and GLASS record.
4. Preferences Panel to specify if boundaries (gang,sector, or system) are to be shown; if hexes are to be outlined; if assemblies are to be labeled (OLC, mnemonic, or type); if the pointer is to be to tracked with crosshairs and hex highlighting; etc.
5. Selector Panel to identify specific groupings of reactor positions.

These panels are discussed more fully below.

2.3.2.4.1 *FaceMap Panel*

The FaceMap Panel, shown in Figure 4, is the main working area for the FM function. It is used to display the current charge, select positions and it contains the application menus. Shown in Figure 4 is a fictitious example charge with the options menu activated. When a user presses the mouse button on a particular position, that position becomes selected. A selected position will have a white outline around it. Once the position is selected the user may place an assembly into that position or perform an Editing function such as Cut, Copy and Paste.

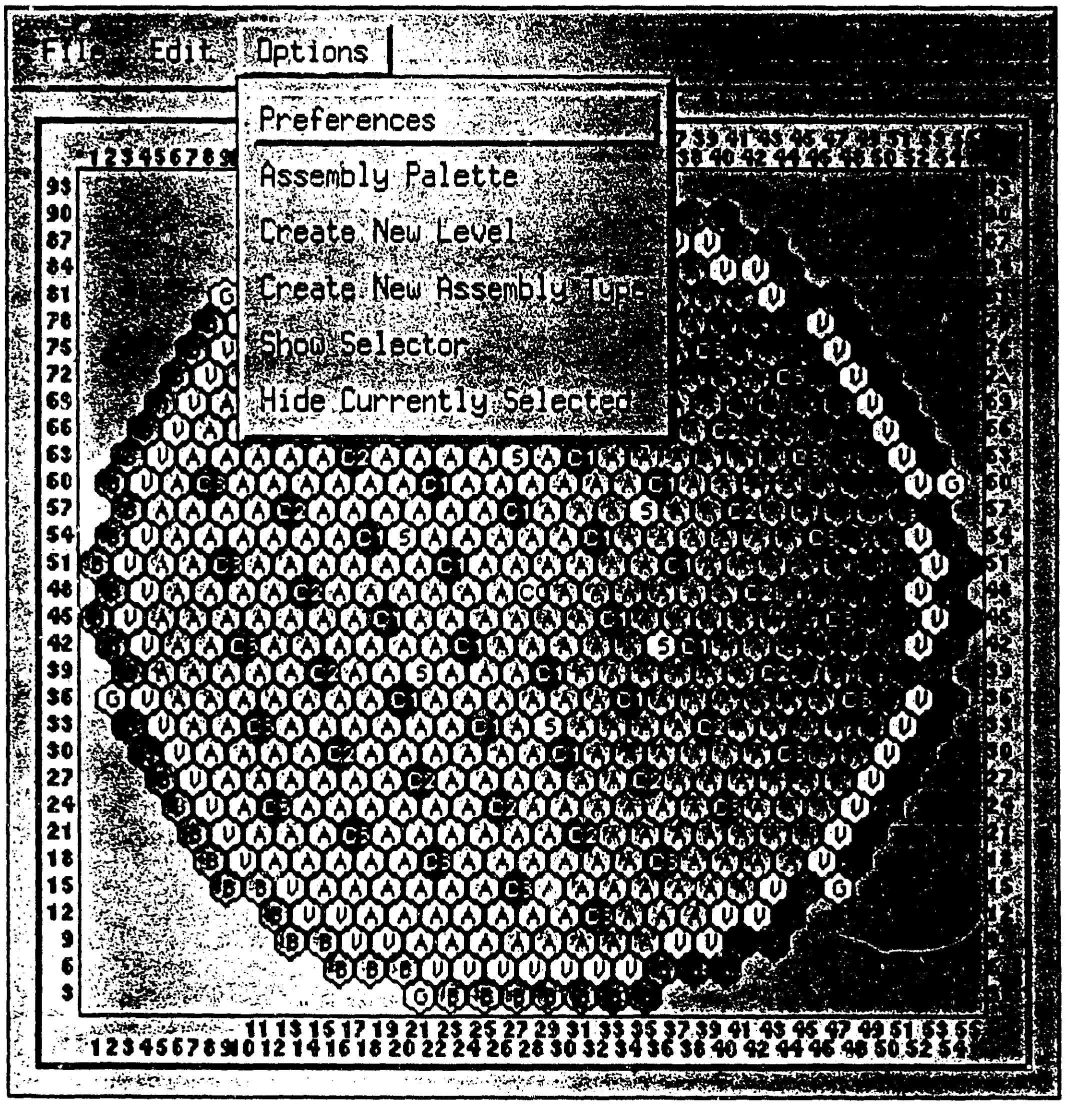


Figure 4. FaceMap Panel for the JOSHUA FM Function

2.3.2.4.2 *Assembly Palette Panel*

The Assembly Palette Panel is shown in Figure 5. The Assembly Palette contains a menu of assemblies that already exist, i.e., have already been created and for which complete JOSHUA data records exist in the read-path hierarchy. The user selects assemblies from this list by clicking (with the mouse) on an assembly name in the Palette. The response to this select procedure depends on the state of the application. If positions in the reactor map were previously selected, then the assembly type from the Palette will be placed in each of those reactor facemap positions. If no facemap positions were selected, then the assembly from the Palette will be loaded into the Assembly Editor.

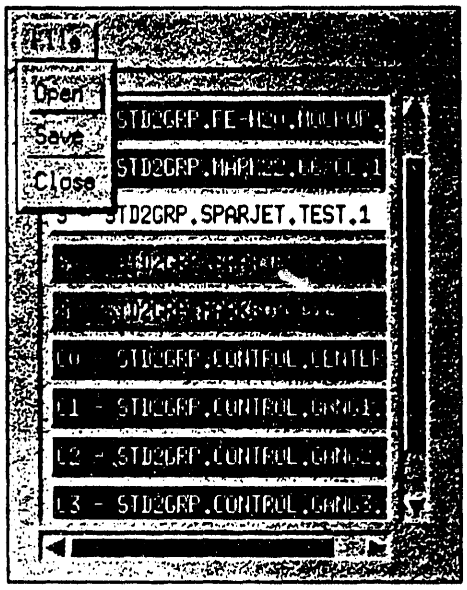

File
Open
Save
Close
STD2GRP.FE-H2O.NUCFUP
STD2GRP.MAR
STD2GRP.SPARJET.TEST.1
C0 - STD2GRP.CONTROL.CENTER
C1 - STD2GRP.CONTROL.GANG1
C2 - STD2GRP.CONTROL.GANG2
C3 - STD2GRP.CONTROL.GANG3

Figure 5. Assembly Palette Panel for JOSHUA FM Function

2.3.2.4.3 *Assembly Editor Panel*

The Assembly Editor Panel, shown in Figure 6, is used to define (i.e., create and edit) assemblies.

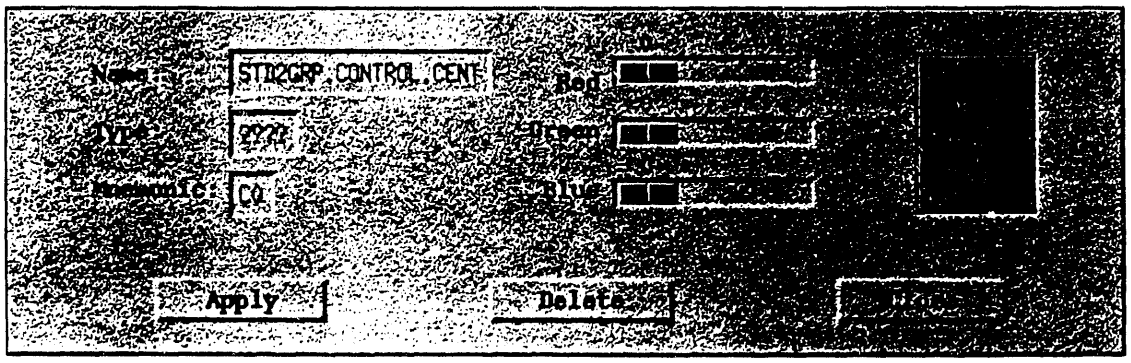


Figure 6. Assembly Editor Panel for JOSHUA FM Function

An assembly definition consists of a name, type, mnemonic and color. The name identifies the desired GLASS record of cell averaged cross section data. The type and mnemonic are used only for compatibility with the current set of codes and make the FM function backwards compatible with existing RM-created records. The color permits the charge designer to graphically differentiate between assembly types. When the user creates an assembly type and presses the apply button, the assembly is placed in the Assembly Palette.

2.3.2.4.4 *Preferences Panel*

The Preferences Panel , shown in Figure 7, is used to set the application display to the users choice.

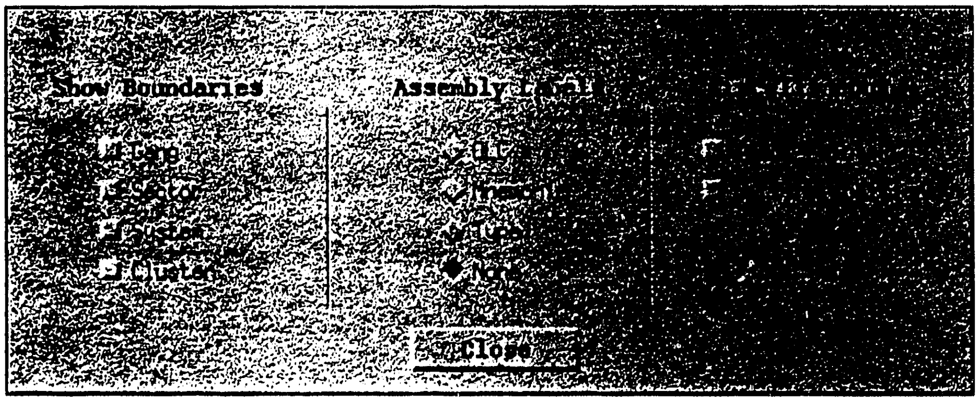


Figure 7. Preferences Panel for JOSHUA FM Function.

Sector, system, gang and cluster boundaries may be toggled and, when 'on', will be displayed on the FaceMap Panel as white lines. The positions of the FaceMap can be displayed with OLC#, mnemonic, type or no label. Pointer tracking, a crosshair that follows the mouse and spans the length and width of the FaceMap Panel, can be toggled. A black outline can be placed on the positions of the FaceMap Panel to help differentiate two similar colored adjacent positions.

2.3.2.4.5 *Selector Panel*

The Selector Panel, shown in Figure 8, is used to specify symmetry options and identify specific groupings and type of reactor positions. This panel consists of four small facemaps, each of which represents a specific grouping of reactor positions; namely, gangs, sectors, systems, clusters. The user selects a grouping by pressing the mouse button while the pointer is inside a particular grouping. To copy sector #1 to sector #3, the user 'selects' sector #1 and chooses *copy* from the *Edit* menu on the FaceMap display. Then the user selects sector #3 and chooses *paste* from the same Edit menu. Extended selections can be made by holding down the shift key while pressing the mouse button.

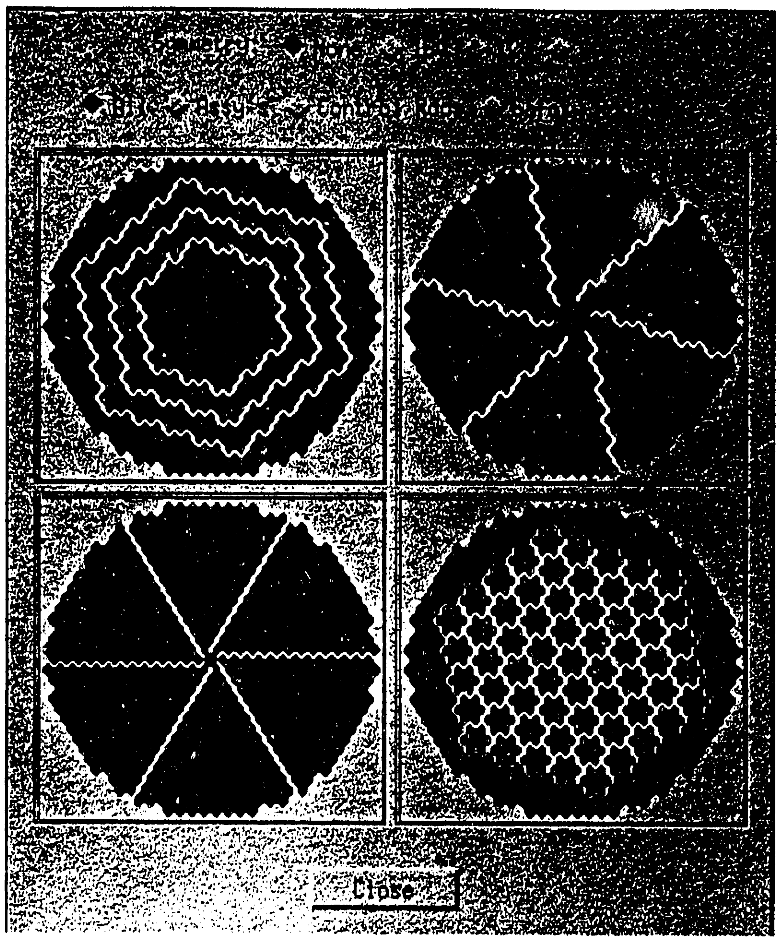


Figure 8. Selector Panel for JOSHUA FM Function

The selector also has two sets of toggle buttons which affect symmetry (e.g., 60°, 120°, 180°, 360°) and position type. The symmetry option changes the way selecting works. With 60° symmetry a selection is replicated in each sector. The position type toggles affect which type of positions (assemblies, control rods, safety rod assemblies) are being referenced with the selector panel. For example, if gang 2 and the *Control Rod* toggle were selected then the positions referenced would be all the control rod positions in gang 2.

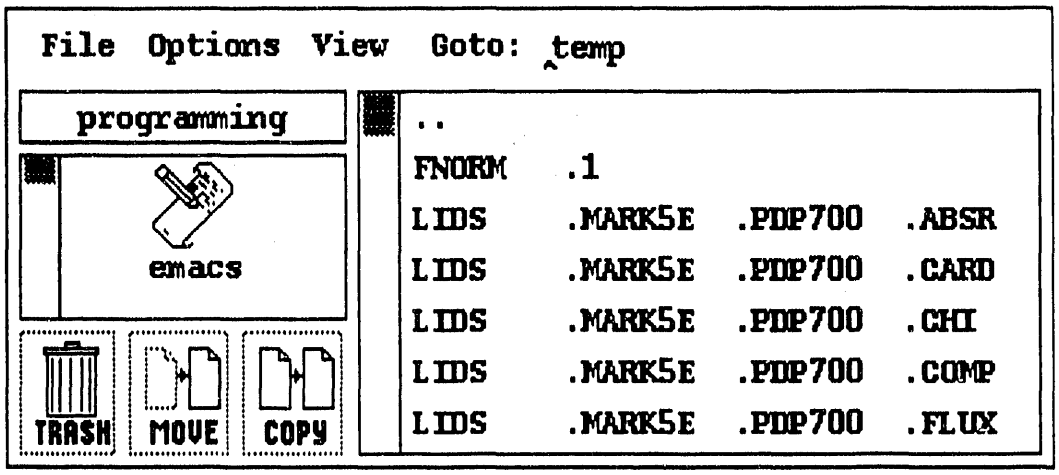
2.3.2.4.6 *Coding of FM Prototype Completed*

The completed prototype of the J80/J90 FM function has been evaluated by the Reactor Physics Analysis Task Team. To facilitate the evaluation a test Mac was equipped with the appropriate software and Ethernet connection to run the new X based FM. It was obvious from the initial installation that a few features of the new FM were just too slow on the Mac. (The development platform is a Sun SPARCstation 2.)

This was remedied using a caching scheme to speed up the FaceMap tool. This and other optimizations have improved the performance of the FaceMap tool, and consequently the FM application, to be faster on the Mac than the unoptimized version on the Sun SPARCstation 2. The Reactor Physics Group, which is expected to be the major user of the FM application, is providing additional feedback that will be used to enhance the usability and utility of the production version.

2.3.2.5 *Desktop GUI (Prototype Graphical Terminal System) for J80*

A prototype graphical terminal system is under development which gives users a *Mac* -like interface to the J80 system. This code is being developed from an application previously written in X Windows for the Unix file system. JOSHUA subtrees are represented as folders and data records are represented as documents similar to the Mac. Folders and files can be opened by double clicking them. The current functionality of the system is the List Subtree (LI) command shown in Figure 9, and the List Brothers (LB) command shown in Figure 10.



```
..
FNORM    .1
LIDS     .MARK5E   .PDP700   .ABSR
LIDS     .MARK5E   .PDP700   .CARD
LIDS     .MARK5E   .PDP700   .CHI
LIDS     .MARK5E   .PDP700   .COMP
LIDS     .MARK5E   .PDP700   .FLUX
```

Figure 9. List Subtree Command

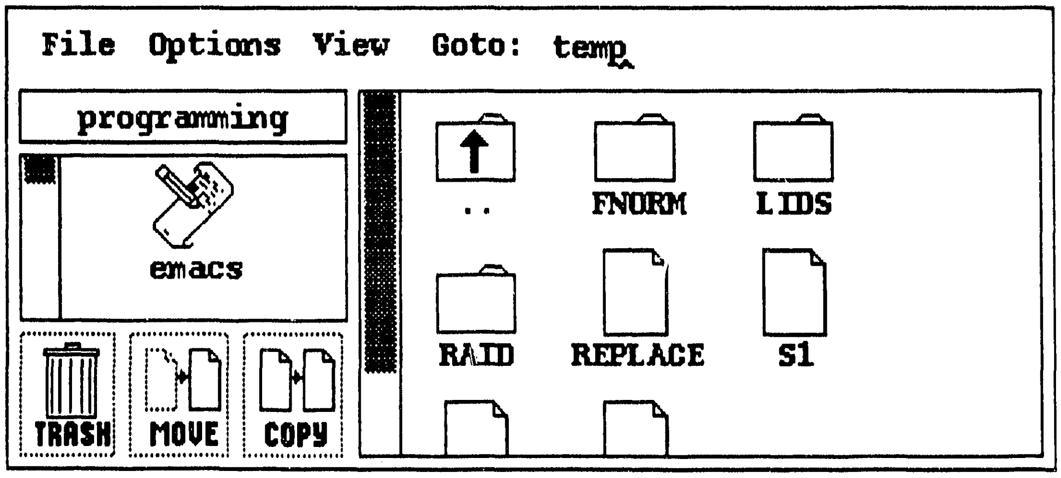


Figure 10. List Brothers Command

Other J80 functions to be included are copy, rename, delete. Other features planned include multiple listing formats such as those in Mac's System 7, double clicking to display either a record (DD) or an existing FaceMap file (RM), different icons for different record types, drag-and-drop metaphor to manipulate files, and more.

2.4 Future Directions

The increased computing power available with the latest workstations requires seriously considering their capability as an alternate resource for supercomputing cycles. In particular, for those problems that lend themselves to distributed or parallel computing methods, the use of a group of workstations offers the possibility of relatively low-cost, high-performance computing capability. While perhaps applicable only to certain groups of problems, it appears that those groups form a major part of the computational workload at SRS.

There are various software environments purported to enable implementation of parallel computing. (Parallel computing may be considered as an exacting subset of distributed computing concepts, and the remaining discussion will focus on the former.) Among these are: Linda, a parallel-programming language that uses simple C-language statements that provide for all process spawning, data sharing, and synchronization; PVM (Parallel Virtual Machine), a software package developed at Oak Ridge National Laboratory to use a heterogeneous network of parallel and serial computers as a single computational resource; ***Express*** , by *ParaSoft* Corporation, a set of tools and utilities designed for parallel processing. These software systems, and others, will be studied for suitability to SRS applications codes and problems.

At the same time, ASCENT plans to form strategic alliances with existing National Science Foundation and Department of Energy centers for supercomputing research and development. The benefit of these alliances will be to facilitate rapid incorporation of the latest cost-effective technology into the SRS scientific computing environment.

An initial target for testing these methods is the GLASS code, a transport code for unit cells used to develop fewgroup cross-sections for use in reactor diffusion codes. The intriguing goal is to develop a parallel algorithm for using GLASS for each cell, or group of supercells, and implement a transport theory solution for the reactor charge.

An important part of achieving success in this endeavor will be the necessity to maintain a secure network capable of providing access to all users, and providing mass storage systems and distributed file systems that make mass storage capacity available to any cpu that needs it..

Consistent with ASCENT's charter, the Team will continue to focus on the following three interconnected data management tasks:

a. I/O data management or symbol manipulation, which comprises the interface between the user and the I/O data base.

b. Computational data management, which comprises the interface between the numerically intensive part of the computer code and the corresponding computational, or 'internal', data base.

c. Interface, or flow of data, between the I/O data base and the computational data base.

END

DATE FILMED

4/23/92