instead of being interface procedures of a separate module. The use of new compiler directives, and an extended concept of FORWARD procedures, support the use of compiletime libraries. Small system procedures have been put into a standard compiletime library which is automatically utilized during the compilation of any module which invokes one of the procedures.

cmdFile (command file) and logFile (logging file) are now the standard input and output files used by the compiler and the runtime system where tty (i.e., the user's terminal) was previously used. Both are initially opened to tty. These files are used instead of tty so that the user can "redirect" the standard input or output stream if desired. This can be done via subcommands at the start of program execution, or via explicit opens during program execution. The system procedures ttyRead and ttyWrite still provide direct communication with tty.

The CHECK and NOCHECK compiler directives have been implemented. CHECK directs the compiler to henceforth emit code to check certain conditions (such as array subscripts and NULLPOINTER's) at runtime which cannot be determined at compiletime. NOCHECK can be used to turn such checking off. These directives have been in the language (in a slightly different form), but they had never been implemented in the code generators.

MAINSAIL has previously guaranteed ASCII character codes. This requires a translation on machines with other character codes (e.g., the IBM-370 uses EBCDIC). Our experience shows it is not difficult to write programs independently of the character codes if certain minimal assumptions are in effect. For example, the characters A...Z are guaranteed to be in alphabetic order, but they are not necessarily contiguous. We have also introduced eol (end-of-line) in place of the previous ASCII-dependent crlf (carriage-return-line-feed), and eop (end-of-page) in place of the ASCII form-feed. eol and eop are defined as implementation-dependent string constants. Character incompatibilities among machines is a difficult problem to deal with, and may ultimately complicate MAINSAIL's implementation on machines with "deficient" character sets (e.g., a CDC with a 6-bit character set).

Compiler Design

Two steps have been taken with the goal of getting the compiler to execute in a small address space. First, it has been broken into smaller modules. It used to consist of about 10 modules, but now consists of about 60. This allows a more accurate working set of modules to build up in memory since no one module is so large that it displaces most of those currently resident. However, there is more overhead involved in initializing so many modules on machines which have sufficient memory for the entire compiler. The second step has been the size reduction or elimination of many of the compiler's data structures. Where possible data is maintained on a file rather than in memory.

To save the space required by the text of string constants, the compiler error messages have been placed on a file. Calls to error message procedures specify the location on the file of the appropriate error message. A program has been written to generate a new compiler error message file by combining the messages on an existing error message file with any new messages specified as string constant arguments to error procedures in the compiler modules.

The code generators have been modified to output additional information (in a separate file) for debugging use during execution. This debugging file is created only if requested as a subcommand to the compiler. It contains a symbol table and a cross reference between the generated code and the source files.

Documentation

A new MAINSAIL manual is nearly complete. The former manual was an alphabetically ordered reference manual written primarily for internal use, that is, for those who were either already familiar with MAINSAIL or who at least knew SAIL. The new manual is a more readable reorganization and expansion of the reference manual information, organized by topic (e.g., data types, procedures, modules), and incorporating numerous examples.

Concurrent with the writing of the manual was the development of a formatting program to input the manual as written (with encoded section numbers and index references, no table of contents, etc.) and output a complete, ready-to-print manual.

An invited paper entitled "The MAINSAIL Project: Developing Tools for Software Portability" was delivered at the First Annual Symposium on Computer Application in Medical Care, given in Washington, D.C. in October, 1977. This has resulted in a number of inquiries from researchers interested in MAINSAIL's portability.

Emulation Research

The goal of the emulation research is to determine efficient means of representing MAINSAIL programs for interpretive execution. If the interpreter can be written in the microcode of the host machine, the resulting emulation should be more efficient than execution of MAINSAIL translated into a standard machine code. This approach simplifies compilation, and allows the efficient monitoring of program execution, so that debugging capabilities and performance measurements can become an integral part of program execution.

Statistics gathered from programs written in MAINSAIL have been used to guide and justify the design of a language representation suitable for emulation. An interpreter has been developed, and the static and dynamic properties of the resulting high level MAINSAIL interpretation are now under study. Its properties will be compared with conventional machine language implementations of MAINSAIL. The generated code appears to be about a third the size of standard machine code. It is more difficult to measure execution time differences since the processor design must be taken into account.

The characteristics of a suitable host processor to support the emulation are being examined in detail. It appears that a "universal host" (i.e., a processor not designed with a particular representation in mind) will not be able to execute a tailor-made representation as fast as a conventional processor can execute a standard machine language representation. Thus a "poor" representation on a processor designed to execute that representation seems faster than a "good" representation on a processor not designed for that representation. For this reason a microcode and processor organization which are oriented toward execution of the MAINSAIL representation are under design.

ALGORITHMIC LANGUAGE IMPLEMENTATIONS OF MYCIN-LIKE SYSTEMS

Production systems (PS) have been used extensively for knowledge representations for a number of AI applications such as MYCIN (2). Traditionally these systems have been implemented in various dialects of LISP. This has been so partly because LISP contains several "natural" representations for PS's, and partly because of the unique development and debugging environment offered by systems like INTERLISP.

Because of its generality and power, however, INTERLISP tends to be a expensive system to run, requiring a large amount of computational time and a large address space. While the expense of such a system is justifiable in a research and development environment, it may not be in more operational environments where these programs are to be used. One way to overcome this cost is by designing more economical LISP systems - several groups are working on this. These systems are typically built around special-purpose "LISP machines". Over the past year we examined another alternative, converting a MYCIN-like system into an algorithmic programing language (such as SAIL or MAINSAIL). This approach may offer advantages in being able to run versions of MYCIN-like systems on existing laboratory computers. The objectives of this study were to find ways to trim the resource requirements of the system while preserving as much of the knowledge representation clarity and modularity of the production system approach as possible. One of the major hurdles to be overcome is the difference in program and data representations between LISP and algorithmic languages. Several different approaches and languages were explored. Some are implementable in almost any algorithmic language, while others exploit features unique to certain languages.

The principal design features of MYCIN include (3):

1) A rule based consultation system. The knowledge is represented as collection of production rules. The consultation is driven by a goal directed search of the knowledge base.

2) An examination program which will explain the "line of reasoning" the system has gone through to produce the current consultation.

3) A question-answer system to query the system on parts of the consultation, or to ask general questions of the knowledge base.

4) A method of updating the knowledge base, by adding new rules or changing or deleting incorrect rules.

The knowledge base for the system is stored as a collection of production rules in the form of PREMISE-ACTION pairs. The consultation is driven by a goal-

---

(2) See for example, Davis, R., Buchanan, B., and Shortliffe, E., "Production Rules as a Representation for a Knowledge-Based Consultation Program," Artificial Intelligence, Vol 8, No 1, February 1977.

(3) see for example, Shortliffe, E.H., "Computer-Based Medical Consultations: MYCIN," Artificial Intelligence Series 2, Elsevier, New York, 1976.

directed search of these rules, i.e. if the PREMISE of a rule depends on the value of a given parameter and its value is not known, rules which conclude something about that parameter value are evaluated. The flow of control depends on an interaction between goals stated in the IF clause of one rule, and the THEN clause in others where this interaction changes as the rule base is changed. This can be readily achieved in LISP by storing the executable PREMISE-ACTION routines as properties of the various rules and linking them in a list. This unity between program and data is one of the key points in the difference between a LISP approach and one using an algorithmic languages.

Considerations in Non-LISP Approaches to MYCIN

MYCIN can be viewed as a very complex "IF THEN" clause, but this ignores the flexibility and modularity of knowledge representation of a PS. At the very least, the programming language should have the capability of creating LISP-like data structures, such as trees and lists. If we examine the nature of the rule interpretation, we can see that it is recursive in nature, so the language should also support recursive procedures. Many modern languages such as SAIL, PASCAL and MAINSAIL have these features.

The main problem is to represent the rules in such a way that they can be executed or interpreted in some sense, but can also be woven into the data base so that they may be fetched, examined, and modified when needed. We need to unify the control store and the data store in a way similar to that of LISP. We have looked at two approaches: one in which the rules are represented strictly as data, and interpreted as needed, and one in which the rules are represented as procedures.

Rule Interpretation Approach

In this approach the production rules are represented strictly as data, and procedures written to interpret the data structure. This is equivalent to writing a small, special purpose LISP interpreter. Any language which fills the requirements outlined above can be used to write the interpreter. The general procedures needed for this approach are:

1) A procedure to insure that atoms are unique

2) Procedures to read and write the data structures.

3) A procedure for each of the LISP functions to be executed, e.g., logical operations.

4) A procedure equivalent to LISP's EVAL, which will examine a list and invoke the correct procedures to interpret it.

This scheme has many of the advantages of the LISP version. It is easy to add new rules to the system that use only the currently defined functions. The hope would be that this interpreter would be smaller and faster than equivalent more general LISP machinery. Some disadvantages are that if a new function is needed, the EVAL section of the program must be rewritten and recompiled. The effort involved on this can be minimized by proper modularization, however.

Procedural Approaches

In these approaches we generate procedures which perform the following functions for each rule in the knowledge base:

1) Execute the PREMISE of the rule, and return a value indicating whether or not the PREMISE is true. In addition there must be a mechanism for marking which clause of the premise failed, if any.

2) Execute the ACTION of the rule.

3) Return a list of the parameters appearing in the PREMISE of the rule (also marking which clause they occur in).

4) Return the parameter (or list of parameters) referenced in the ACTION of the rule.

5) Print an English version of the rule.

These procedures can be generated automatically either translating LISP rules from MYCIN or translating an English input. In this way, we have replaced the interpretation of the rules as data structures with procedures that return truth values and carry out the actions. Besides the procedures, there is also a data structure which represents the interconnection of the rules. This approach could be faster than the interpreter for rules since we effectively have interpreted the rules once and for all at compile time.

Case Statement Procedures

In this implementation, a procedure is constructed for each of the functions mentioned above. The procedure takes an integer representing the rule number, and executes the proper subsection for that rule. For example, we might have:

```
BOOLEAN PROCEDURE premise ( INTEGER ruleNo );
        CASE ruleNo OF
                BEGIN
                [ 1 ]   < PREMISE of RULE001 >
                [ 2 ]   < PREMISE of RULE002 >
                        .
                        .
                        .
                END;
```

By marking a parallel data structure, we can trace which rules and which clauses of which rules have been executed.

A disadvantage to this method is the relative inflexibility of the rule base once it has been written. This can be alleviated to some degree by proper modularization.

MAINSAIL Implementation

This approach exploits some of the unique features of MAINSAIL. A MAINSAIL program is broken up into a number of MODULE's, which communicate with each other by means of interface fields. Each rule is represented as a different module, each with the same interface field definition. The interface field represents the values and procedures outlined above. The rule modules are stored in libraries for execution as needed. The fact that the data section of a module may be assigned to a pointer variable gives us the ability to unite the control store and the data store.

There are two possible methods of evaluating the rules. The first is to create instances of all the rules, and save the pointers to the correct MODULE as part of the data structure. The second is to create an instance of a rule only when it is necessary to evaluate that particular rule, and to dispose of the rule when it is no longer needed. The first method is much faster than the second, since at the time of the consultation the code for all rules is in core. The second method is extremely core efficient, however, since only a few rules will be active at any given time. This method is particularly suited for a small machine environment.

One of the advantages of this MAINSAIL approach is that there is no linking step in compiling modules as in other algorithmic languages (e.g., SAIL). Thus rule modules may be changed at will without relinking the entire system. The cost for the flexibility MAINSAIL offers is the overhead of intermodule calls in a dynamic memory environment - modules do not always load at the same address.

SAIL/LEAP Implementation

The LEAP package of SAIL offers yet another approach (4). Among the LEAP facilities are procedures which "assign" an ITEM (the basic element of LEAP) with a SAIL procedure and "apply" executes the procedure associated with an ITEM. Since ITEM's are part of the data store, and can be manipulated in data structures, this allows the needed interactions of control store and data store.

The strategy in SAIL is to associate an ITEM with each of the basic procedures for each rule. These ITEM's are stored so they can be retrieved when the appropriate rule is invoked. There are several points where special care must be taken. The program to modify the knowledge base must update several files in this implementation; 1) a header file which contains the declaration of all ITEM's used in the system, 2) an initialization procedure which creates all the triples, performs all the assigns, etc. needed, and 3) the actual code for the rule.

The advantages of the SAIL version is a somewhat more direct mapping of the data to a single procedure representing a rule. Adding a rule will be more cumbersome. The new rule can be stored in a file by itself, and required in the initialization module as a REQUIRE'd LOAD!MODULE. Thus only the new rule and the initialization need to be recompiled but the whole system must be relinked.

---

(4) LEAP is a facility added to SAIL for the associative storage, retrieval, and manipulation of objects. See Feldman, J.A. and Rovner, P.D., "An ALGOL-Based Associative Language," CACM 12, 8, August 1969.

Summary of Preliminary Results

A highly stripped down, seven rule "MYCIN" system has been successfully emulated using each of these approaches. These systems merely indicated the workability of each approach but the system was not complex enough to draw any quantitative conclusions about relative efficiency. Each of the methods has its advantages and disadvantages. SAIL, like LISP, requires a large core image and currently runs only on PDP-10 systems. The MAINSAIL version seems well suited to a small machine environment through the high modularization and dynamic memory management. By use of a virtual data structure, the rule interpretation approach could be made to run in a smaller core image as well.

It must be noted that no implementation of MYCIN in an algorithmic language will maintain the full flexibility of the INTERLISP version for rule changes, control structure experimentation, and debugging. One must be willing to trade the flexibility of a development system requiring large resources for a more fixed production oriented system with substantially smaller demands.

2.1.2.6 USER SOFTWARE AND INTRA-COMMUNITY COMMUNICATION

We have continued to assemble and maintain a broad range of utilities and user support software. These include operational aids, statistics packages, DEC-supplied programs, improvements to the TOPS-10 emulator, text editors, text search programs, file space management programs, graphics support, a batch program execution monitor, text formatting and justification assistance, and magnetic tape conversion aids. Over the past year we have made changes and updates to more than 60 programs in this stable. While many of these changes were maintenance bug fixes, major improvements were made to SPELL, MACRO, BACKUP, DIABLO, tape service programs, VIEW, and the user spooler interface. In addition we have brought up a number of new programs including PASCAL (DECUS), EMACS (a display oriented editor from MIT - installed by McMahon at SRI), overload control information programs, MACLISP (MIT), and FAIL. Changes are in progress to the bulletin board system to allow string searches in the "subject" and "body" of bulletins to find information of interest and to allow general wild card specifications within strings.

2.1.2.7 DOCUMENTATION AND EDUCATION

We have spent considerable effort to develop, maintain, and facilitate access to our documentation so as to accurately reflect available software. The HELP and Bulletin Board systems have been important in this effort. As subsystems are updated, we generally publish a bulletin or small document describing the changes. As more and more changes occur, it becomes harder and harder for users to track down all of the change pointers. We are in the process of reviewing the existing documentation system again for compatibility with the programs now on line and to integrate changes into the main documents. This will also be done with a view toward developing better tools for maintaining up-to-date documentation.

2.1.2.8 SOFTWARE COMPATIBILITY AND SHARING

At SUMEX-AIM we firmly believe in importing rather than reinventing software where possible. As noted above, a number of the packages we have brought up are from outside groups. Many avenues exist for sharing between the system staff, various user projects, other facilities, and vendors. The advent of fast and convenient communication facilities coupling communities of computer facilities has made possible effective intergroup cooperation and decentralized maintenance of software packages. The TENEX sites on the ARPANET have been a good model for this kind of exchange based on a functional division of labor and expertise. The other major advantage is that as a by-product of the constant communication about particular software, personal connections between staff members of the various sites develop. These connections serve to pass general information about software tools and to encourage the exchange of ideas among the sites. Certain common problems are now regularly discussed on a multi-site level. We continue to draw significant amounts of system software from other ARPANET sites, reciprocating with our own local developments. Interactions have included mutual backup support, hardware configuration experience, operating system enhancements, utility or language software, and user project collaborations. We have been able to import many new pieces of software and improvements to existing ones in this way. Examples of imported software include the message manipulation program MSG, TENEX SAIL, TENEX SOS, INTERLISP, the RECORD program, ARPANET host tables, and many others. Reciprocally, we have exported our contributions such as the drum page migration system, KI-10 page table efficiency improvements, GTJFN enhancements, PUB macro files, the bulletin board system, MAINSAIL, SPELL, SNDMSG enhancements, our BATCH monitor, and improved SA-10 software.

2.1.3 RESOURCE MANAGEMENT

2.1.3.1 ORGANIZATION

The SUMEX-AIM resource is administered within the Genetics Department of the Stanford University Medical School. Its mission, locally and nationally, entails both the recruitment of appropriate research projects interested in medical AI applications and the catalysis of interactions among these groups and the broader medical community. User projects are separately funded and autonomous in their management. They are selected for access to SUMEX on the basis of their scientific and medical merits as well as their commitment to the community goals of SUMEX. Currently active projects span a broad range of application areas such as clinical diagnostic consultation, molecular biochemistry, belief systems modeling, mental function modeling, and instrument data interpretation (descriptions of the individual collaborative projects are in Section 4 beginning on page 61).

Early this year it was announced that Professor Lederberg had been named president of Rockefeller University. Whereas the SUMEX staff at Stanford will miss the face-to-face contacts of his involvement in SUMEX-AIM, his relocation may even broaden and strengthen the biomedical research base that will be represented in our AI applications. Professor Lederberg has expressed a strong, continuing commitment to medical AI applications and to SUMEX. The network and message facilities provide a mechanism to continue his close participation in this research and AIM Executive Committee activities.

The depth of the Stanford multi-disciplinary support of SUMEX-AIM has been a key asset in being able to bridge this management transition. Professor Edward Feigenbaum, who is chairman of the Stanford Computer Science Department and has long been the co-principal investigator of SUMEX-AIM, will take over as PI. Professor Stanley Cohen, who has been PI of the MYCIN project and on the Stanford SUMEX advisory committee, will provide the biomedical ties and coordination with the Stanford Medical School and projects. The new management team is committed to sustaining the active development of the SUMEX-AIM resource and community.

2.1.3.2 MANAGEMENT COMMITTEES

As the SUMEX-AIM project is a multilateral undertaking by its very nature, we have created several management committees to assist in administering the various portions of the SUMEX resource. As defined in the SUMEX-AIM management plan adopted at the time the initial resource grant was awarded, the available facility capacity is allocated 40% to Stanford Medical School projects, 40% to national projects, and 20% to common system development and related functions. Within the Stanford aliquot, Dr. Lederberg and BRP have established an advisory committee to assist in selecting and allocating resources among projects appropriate to the SUMEX mission. The current membership of this committee is listed in Appendix IV.

For the national community, two committees serve complementary functions. An Executive Committee oversees the operations of the resource as related to national users and makes the final decisions on authorizing admission for projects. It also establishes policies for resource allocation and approves plans for resource development and augmentation within the national portion of SUMEX (e.g., hardware upgrades, MAINSAIL development priorities, etc.). The Executive Committee oversees the planning and implementation of the AIM Workshop series currently implemented under Prof. S. Amarel of Rutgers University and assures coordination with other AIM activities as well. The committee will play a key role in assessing the possible need for additional future AIM community computing resources and in deciding the optimal placement and management of such facilities. The current membership of the Executive committee is listed in Appendix IV.

Reporting to the Executive Committee, an Advisory Group represents the interests of medical and computer science research relevant to AIM goals. The Advisory Group serves several functions in advising the Executive Committee; 1) recruiting appropriate medical/computer science projects, 2) reviewing and recommending priorities for allocation of resource capacity to specific projects based on scientific quality and medical relevance, and 3) recommending policies and development goals for the resource. The current Advisory Group membership is given in Appendix IV.

These committees have actively functioned in support of the resource. Except for the meetings held during the AIM workshops, the committees have "met" by messages, net-mail, and telephone conference owing to the size of the groups and to save the time and expense of personal travel to meet face to face. The telephone meetings, in conjunction with terminal access to related text materials, have served quite well in accomplishing the agenda business and facilitate greatly the arrangement of meetings. Other solicitations of advice requiring review of sizable written proposals are done by mail.

We will continue to work with the management committees to recruit the additional high quality projects which can be accommodated and to evolve resource allocation policies which appropriately reflect assigned priorities and project needs. We hope to make more generally available information about the various projects both inside and outside of the community and thereby to promote the kinds of exchanges exemplified earlier and made possible by network facilities.

2.1.3.3 NEW PROJECT RECRUITING

The SUMEX-AIM resource has been announced through a variety of media as well as by correspondence, contacts of NIH-BRP with a variety of prospective grantees who use computers, and contacts by our own staff and committee members. The number of formal projects that have been admitted to SUMEX has more than doubled since the start of the project; others are working tentatively as pilot projects or are under review.

We have prepared a variety of materials for the new user ranging from general information such as is contained in a SUMEX-AIM overview brochure to more

detailed information and guidelines for determining whether a user project is appropriate for the SUMEX-AIM resource. Dr. E. Levinthal has prepared a questionnaire to assist users seriously considering applying for access to SUMEX-AIM. Pilot project categories have been established both within the Stanford and national aliquots of the facility capacity to assist and encourage projects just formulating possible AIM proposals pending their application for funding support and in parallel formal application for access to SUMEX. Pilot projects are approved for access for limited periods of time after preliminary review by the Stanford or AIM Advisory Group as appropriate to the origin of the project.

These contacts have sometimes done much more than provide support for already formulated programs. For example, Prof. Feigenbaum's group at Stanford has initiated a major collaborative effort with Dr. Osborn's group at the Institutes of Medical Sciences in San Francisco. This project in "Pulmonary Function Monitoring and Ventilator Management - PUFF/VM" (see Section 4.1.6 on page 93) originated as a pilot request to use MLAB in a small way for modeling. Subsequently the AI potentialities of this domain were recognized by Feigenbaum, Nii, and Osborn who have submitted a joint proposal to NIH and have a pilot status at present. This summer Dr. John Kunz from Dr. Osborn's laboratory is planning to spend half time at Stanford to learn more about AI research and to participate more closely in the development of the PUFF/VM program.

The following lists the fully authorized projects currently comprising the SUMEX-AIM community (see Section 4 for more detailed descriptions). The nucleus of five projects that were authorized at the initial funding of the resource in December 1973 are marked by "<*>".

National Community -

1) Acquisition of Cognitive Procedures (ACT); Dr. J. Anderson (Yale University)

2) Chemical Synthesis Project (SECS); Dr. T. Wipke (University of California at Santa Cruz)

<*> 3) Higher Mental Functions Project; K. Colby, M.D. (University of California at Los Angeles)

4) INTERNIST Project; J. Myers, M.D. and Dr. H. Pople (University of Pittsburgh)

5) Medical Information Systems Laboratory (MISL); J. Wilensky, M.D. and Dr. B. McCormick (University of Illinois at Chicago Circle)

6) Pulmonary Function Project (PUFF/VM); J. Osborn, M.D. (Institutes of Medical Sciences, San Francisco) and Dr. E. Feigenbaum (Stanford University)

<*> 7) Rutgers Computers in Biomedicine; Dr. S. Amarel (Rutgers University)

8) Simulation of Comprehension Processes; Drs. J. Greeno and A. Lesgold (University of Pittsburgh)

Stanford Community -

1) AI Handbook Project; Dr. E. Feigenbaum

<*> 2) DENDRAL Project; Drs. C. Djerassi, J. Lederberg, and E. Feigenbaum

3) Generalization of AI Tools (AGE); Dr. E. Feigenbaum

4) Large Multi-processor Arrays (HYDROID); Dr. G. Wiederhold

5) Molecular Genetics Project (MOLGEN); Drs. J. Lederberg and E. Feigenbaum (Stanford) and N. Martin (University of New Mexico)

<*> 6) MYCIN Project; S. Cohen, M.D. and Dr. B. Buchanan

<*> 7) Protein Structure Modelling; Drs. E. Feigenbaum and R. Engelmore

As an additional aid to new projects or collaborators with existing projects, we provide a limited amount of funds for use to support terminals and communications needs of users without access to such equipment. We are currently leasing 6 terminals and 4 modems for users as well as 4 foreign exchange lines to better couple the Rutgers project into the TYMNET and a leased line between Stanford and U. C. Santa Cruz for the Chemical Synthesis project.

2.1.3.4 STANFORD COMMUNITY BUILDING

The Stanford community has undertaken several internal efforts to encourage interactions and sharing between the projects centered here. Professor Feigenbaum organized a project with the goal of assembling a handbook of AI concepts, techniques, and current state-of-the-art. This project has had enthusiastic support from the students and substantial progress made in preparing many sections of the handbook (see Section 4.2.1 on page 123 for more details).

Weekly informal lunch meetings (SIGLUNCH) are also held between community members to discuss general AI topics, concerns and progress of individual projects, or system problems as appropriate as well as having a number of outside invited speakers.

2.1.3.5 AIM WORKSHOP SUPPORT

The Rutgers Computers in Biomedicine resource (under Dr. Saul Amarel) has organized a series of workshops devoted to a range of topics related to artificial intelligence research, medical needs, and resource sharing policies within NIH. Meetings have been held for the past several years at Rutgers and another is planned for this summer. The SUMEX facility has acted as a prime

computing base for the workshop demonstrations. We expect to continue this support for future workshops. The AIM workshops provide much useful information about the strengths and weaknesses of the performance programs both in terms of criticisms from other AI projects and in terms of the needs of practicing medical people. We plan to continue to use this experience to guide the community building aspects of SUMEX-AIM.

2.1.3.6 RESOURCE ALLOCATION POLICIES

As the SUMEX facility has become increasingly loaded, a number of diverse and conflicting demands have arisen which require controlled allocation of critical facility resources (file space and central processor time). We have already spelled out a policy for file space management; an allocation of file storage is defined for each authorized project in conjunction with the management committees. This allocation is divided among project members in any way desired by the individual principal investigators. System allocation enforcement is implemented by project each week. As the weekly file dump is done, if the aggregate space in use by a project is over its allocation, files are archived from user directories over allocation until the project is within its allocation.

We have recently implemented system scheduling controls to attempt to maintain the 40:40:20 balance in terms of CPU utilization (see page 14) and to avoid system and user inefficiencies during overload conditions. The initial complement of user projects justifying the SUMEX resource was centered to a large extent at Stanford. Over the past five years of the SUMEX grant, a substantial growth in the number of national projects was realized. During the same time the Stanford group of projects has matured as well and in practice the 40:40 split between Stanford and non-Stanford projects is not ideally realized (see Figure 11 on page 47 and the tables of recent project usage on page 50). Our job scheduling controls bias the allocation of CPU time based on percent time consumed relative to the time allocated over the 40:40:20 community split. The controls are "soft" however in that they do not waste computer cycles if users below their allocated percentages are not on the system to consume the cycles. The operating disparity in CPU use to date reflects a substantial difference in demand between the Stanford community and the developing national projects, rather than inequity of access. For example, the Stanford utilization is spread over a large part of the 24-hour cycle, while national-AIM users tend to be more sensitive to local prime-time constraints. (The 3-hour time zone phase shift across the continent is of substantial help in load balancing.) During peak times under the new overload controls, the Stanford community still experiences mutual contentions and delays while the AIM group has relatively open access to the system. For the present, we propose to continue our policy of "soft" allocation enforcement for the fair split of resource capacity.

Our system also categorizes users in terms of access privileges. These comprise fully authorized users, pilot projects, guests, and network visitors in descending order of system capabilities. We want to encourage bona fide medical and health research people to experiment with the various programs available with a minimum of red tape while not allowing unauthenticated users to bypass the advisory group screening procedures by coming on as guests. So far we have had

relatively little abuse compared to what other network sites have experienced, perhaps on account of the personal attention that senior staff gives to the logon records, and to other security measures. However, the experience of most other computer managers behooves us to be cautious about being as wide open as might be preferred for informal service to pilot efforts and demonstrations. We will continue developing this mechanism in conjunction with management committee policy decisions.

2.1.4 FUTURE PLANS

This next year will be the first of the 3 year renewal grant term. The principal goals of our work are outlined below. Objectives for the individual collaborating projects are discussed in their respective reports (see Section 4 on page 61).

1) RESOURCE OPERATIONS

We will continue to make available to the SUMEX-AIM communities an effective, state-of-the-art facility to support the development of medical AI programs and to facilitate collaborations between community members. Goals include:

a) Assure a smooth transition in project management as Professor Lederberg moves to Rockefeller University.

b) Continue development of the existing KI-TENEX facility to maximize effectiveness for community use. We expect to continue improving system efficiency, allocation controls, subsystem software, documentation facilities, and communications facilities. We will complete the evaluation of the TELENET network as a more cost-effective source of communication services. Another key issue will be how best to maintain software compatibility between TENEX and the newer releases of DEC's TOPS-20. This may entail another "compatibility package" to translate system calls from one system to the other.

c) Recruit new applications and projects to broaden the range of high quality medical AI applications. We look forward to Prof. Lederberg's efforts at Rockefeller University to try to stimulate new projects as well as others that might be suggested by advisory group members or other contacts.

d) We plan to work closely with other AIM resource nodes, such as the one being implemented at Rutgers this summer, to ensure effective community support between the facilities and to take advantage of expertise in various user groups for system and user software development.

e) We plan to finish the preliminary evaluation of the new DEC 2020 system and to make a recommendation for acquiring one by the end of calendar 1978. The council-approved budget allocation for this machine in year 07. We expect the technological rationale and community need for the use of such a machine for increased capacity and an effective software export mode to mature in year 06. Thus it would be desirable to move this expenditure forward. This may not be possible within NIH appropriation limitations and so we would have to defer delivery until year 07. As part of the acquisition of the 2020 system, we expect to investigate the many issues that will arise from the decentralization such machines will bring. The availability of these machines bodes many advantages for effective support of community computing needs but dangers as well of decreased sharing and software compatibility.

2) TRAINING AND EDUCATION

Within our resources, we will continue to assist new and established user projects in gaining access to SUMEX-AIM facilities. Collaborating projects will provide their own manpower and expertise for the development and dissemination of their AI programs.

a) We will continue to provide a high standard of system documentation and limited staff assistance for user problems.

b) Council disapproved our plan to support a "visiting scientist" position to bring selected investigators in contact with on-going AI projects. Funds were approved to support "collaborative linkages". These will continue to be used to facilitate project communications with the resource.

c) We will provide continued support for the AIM workshop activities in the form of demonstration support, participation in workshop discussions, and assistance for potential pilot users in understanding the SUMEX-AIM community.

3) CORE RESEARCH

Our core research efforts for the next year will emphasize the research work discussed in our proposal but the level of effort will reflect the budget cuts recommended by Council. This effect will be particularly hard felt in the MAINSAIL project.

a) We will provide core research support to about 1.6 staff FTE's and 1 graduate research assistant for the documentation and generalization of AI tools developed in the context of particular applications projects. This work will complement the on-going project developments by providing a link to make results available to the entire community. We plan to partially support the ACE project and the AI handbook project. The detailed research goals of these projects are summarized in Section 4.2.1 and Section 4.2.3.

b) Within the council-approved manpower level for MAINSAIL (2 FTE's), we will only be able to complete a demonstration of the MAINSAIL system. A wide distribution of the language, credible support of a user community, and investigation of implementations for other target machines are well beyond this level of effort. For the next year we plan to complete a debugged compiler that will run on a 28K PDP-11, tutorial and reference manuals (including procedures for bringing up MAINSAIL on new target machines), an interactive symbolic debugger (providing breakpoints, variable examination, single stepping, etc.), and documentation of the key design issues encountered in defining a machine-independent language. Also over this year we will investigate ways in which this demonstration version of MAINSAIL could be transferred to an environment with the necessary resources to extend, distribute, and maintain it properly.

2.2 SUMMARY OF RESOURCE USAGE

The following data give an overview of SUMEX-AIM resource usage. There are five subsections containing data respectively for 1) system loading, 2) system efficiency, 3) resource use by community, 4) resource use by project, and 5) network use.

2.2.1 SYSTEM LOADING

The following plots display several different aspects of system loading over life of the project. These include total CPU time delivered per month, the "peak" number of jobs logged in, and the "peak" load average. The term "peak" refers to the peak of the monthly diurnal loading curve for each variable which in turn is the average of the individual daily diurnal curves. Thus, "peak" values are quite representative of average monthly peak loading and do not reflect individual days. These data show well the continued growth of SUMEX use and the self-limiting saturation effect of system load average. Since late 1976, when the dual processor capacity became fully used, the peak daily load average has remained at about 6. This is a measure of the user capacity of our current hardware configuration and the mix of AI programs.

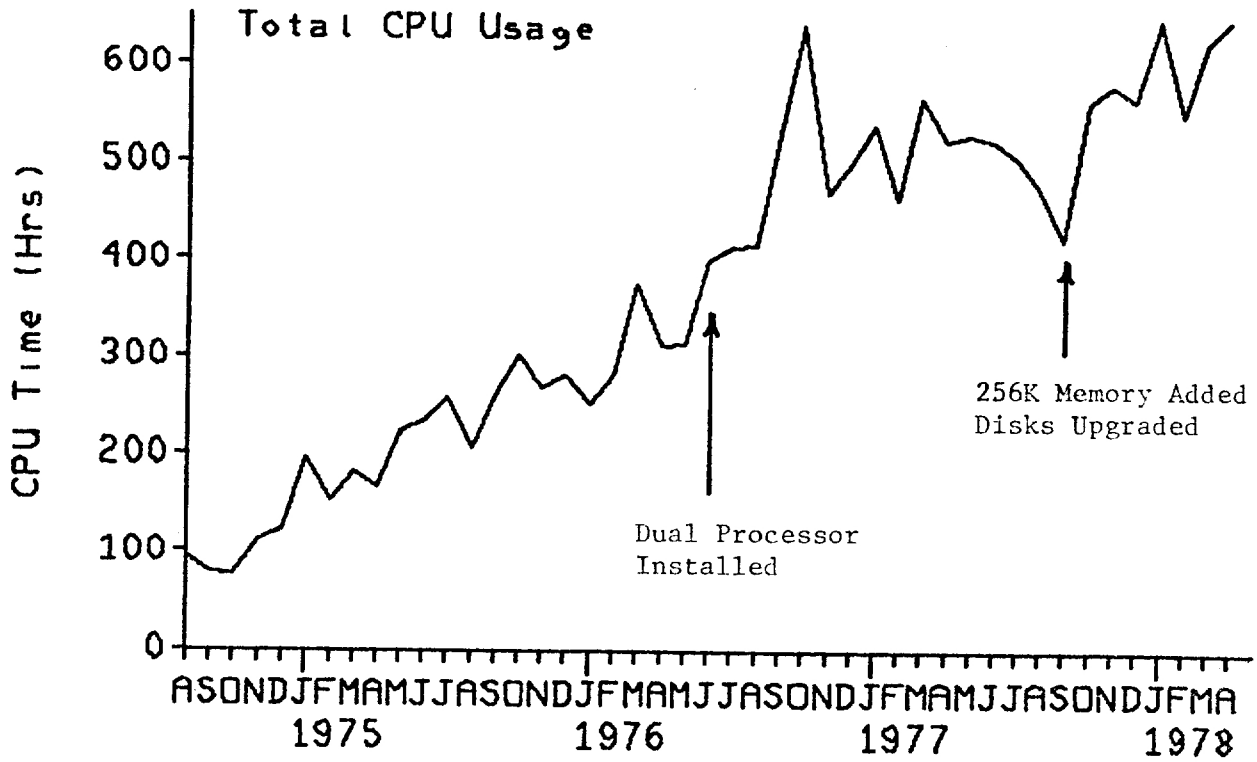

Figure 6. CPU Time Consumed by Month

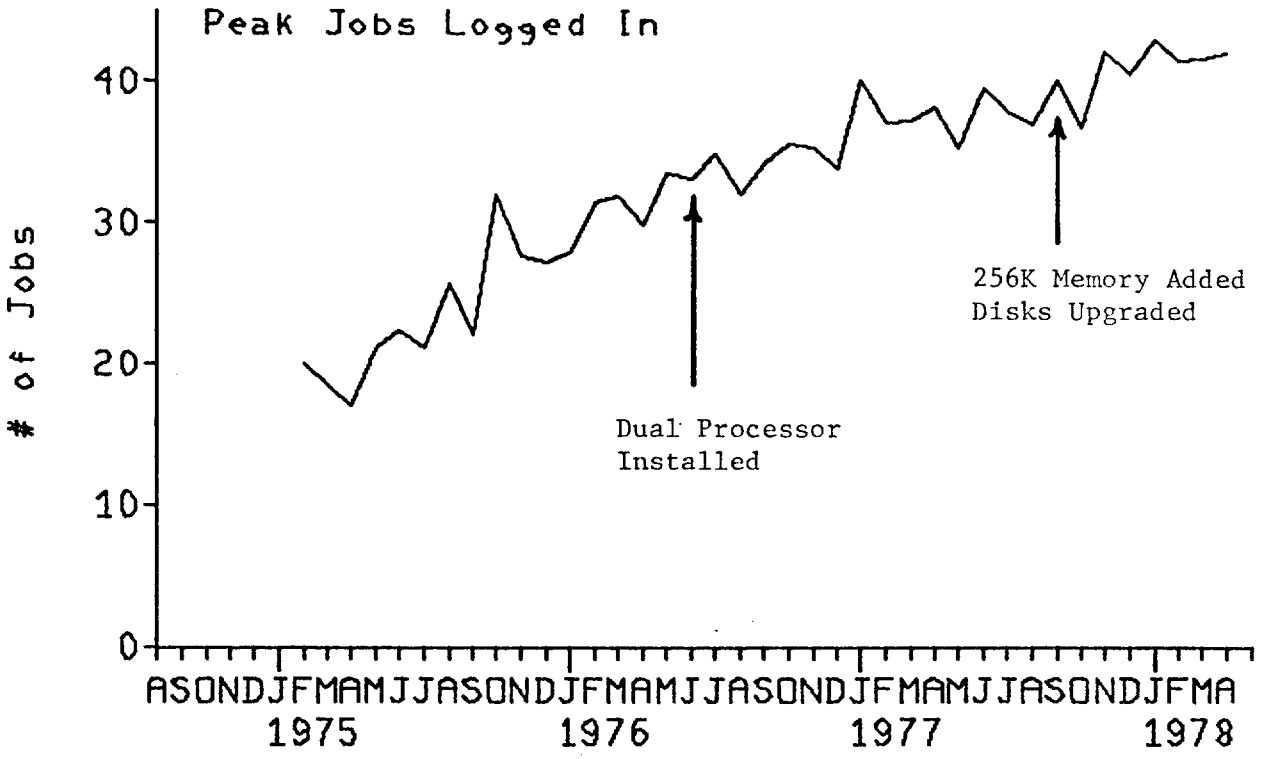

Figure 7. Peak Number of Jobs by Month

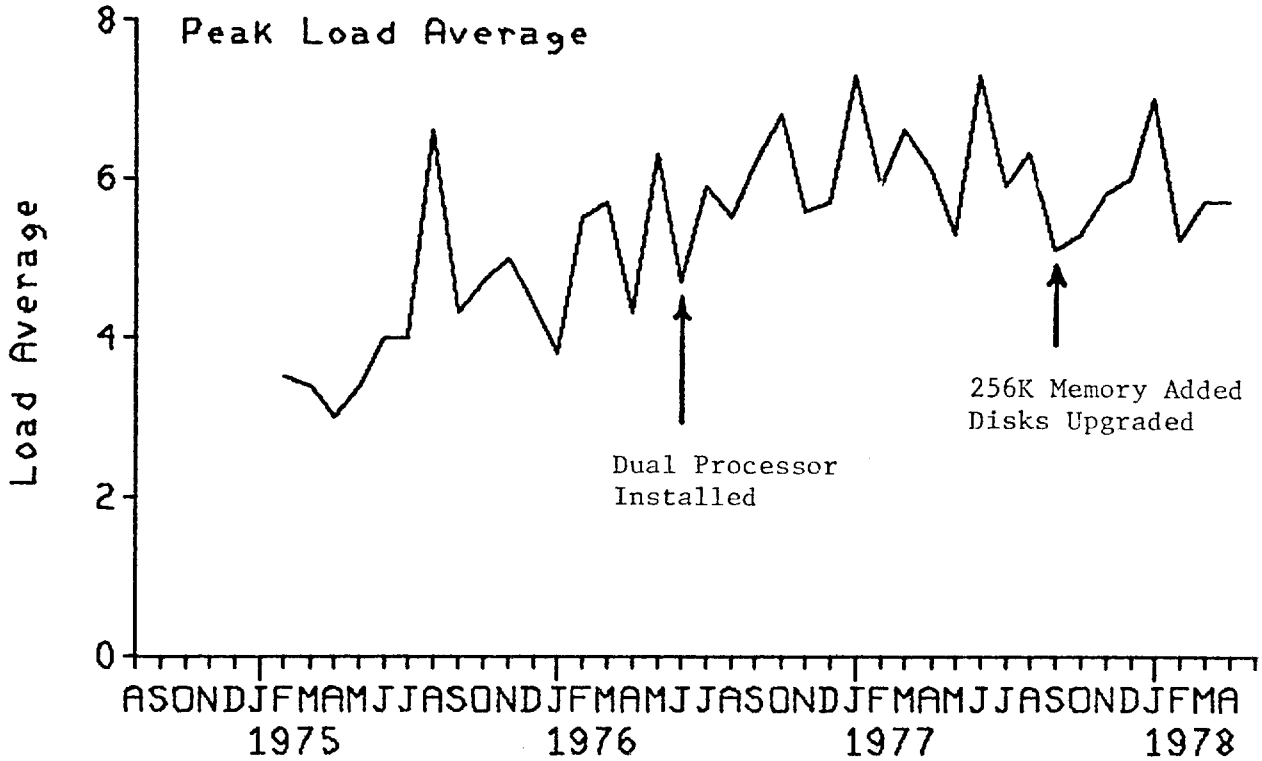

Figure 8. Peak Load Average by Month

2.2.2 SYSTEM EFFICIENCY

The following plots show two measures of system overhead and the influence of hardware augmentations on them. The first is "total overhead" which includes scheduler time, I/O wait, and core management time. The second shows "page trap" time which is charged to user run time but reflects lost time in working set management for each job. Note the sharp rise in overhead with the introduction of the dual processor caused by the increased memory contention. This overhead drops back to single processor levels after we doubled memory. The peak around February/March 1975 is anomalous and reflects testing of the drum and disk systems during installation.

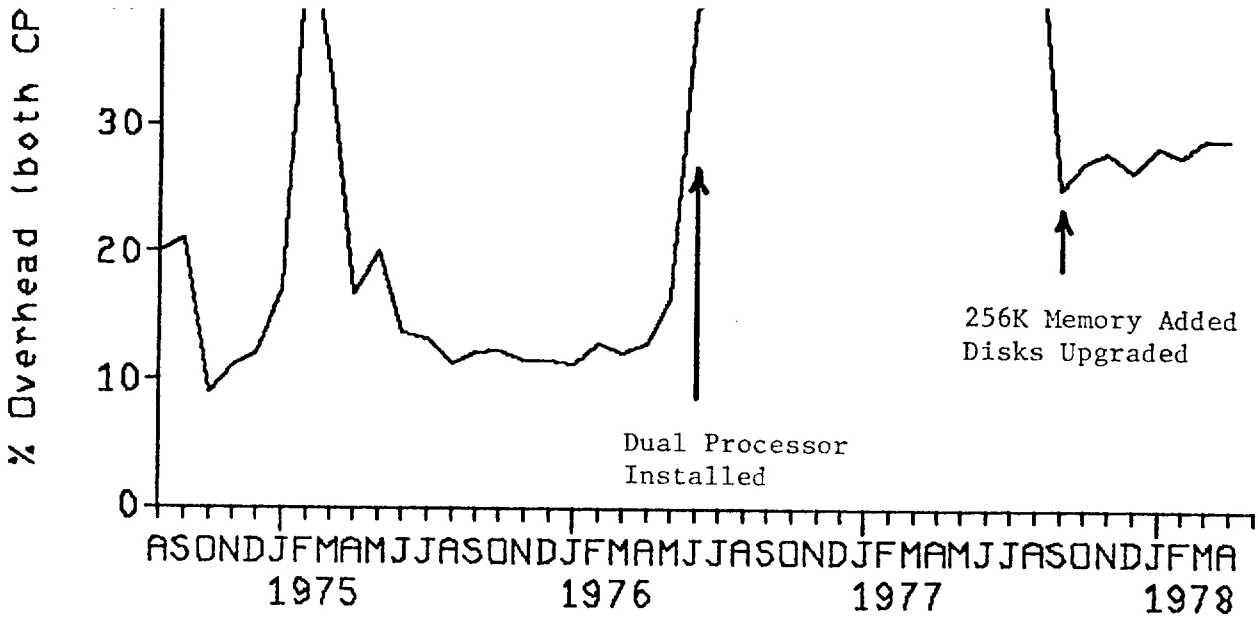

Figure 9. System Overhead by Month

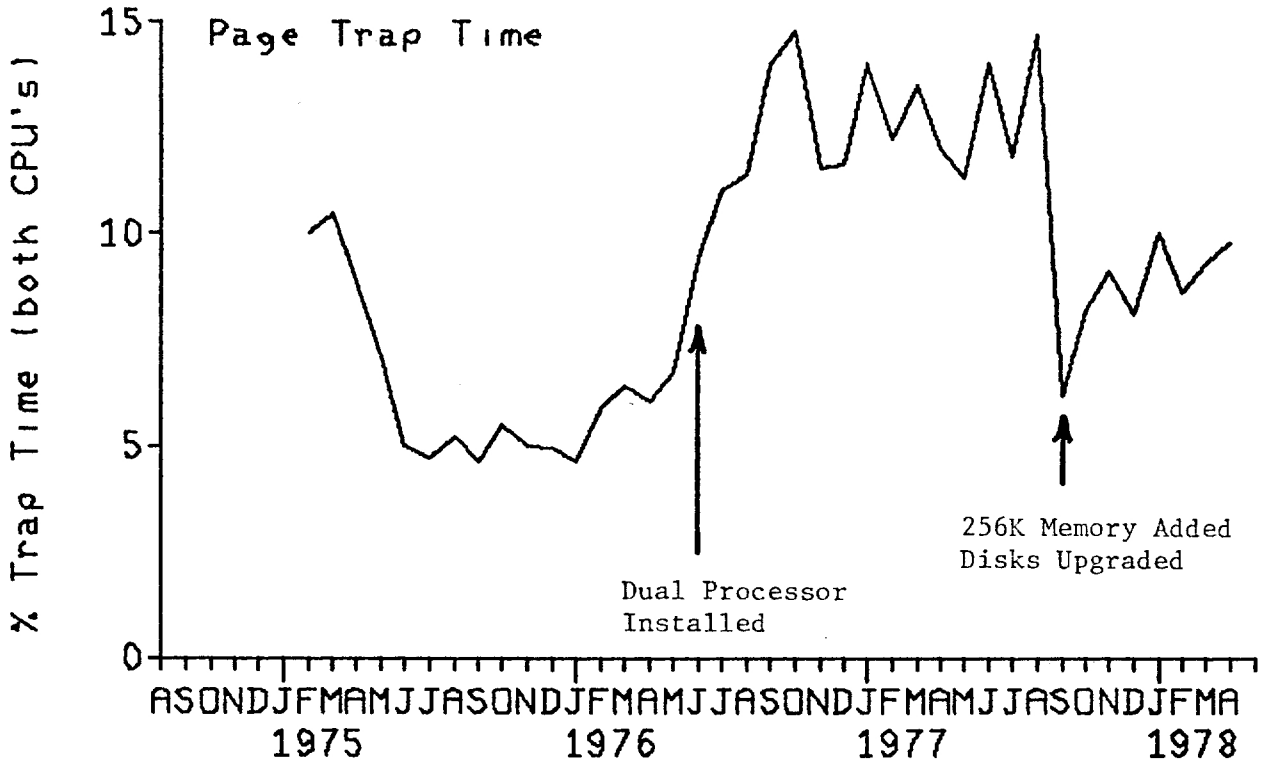

Figure 10. Page Trap Time by Month

2.2.3 RELATIVE SYSTEM LOADING BY COMMUNITY

The SUMEX resource is divided, for administrative purposes, into 3 major communities: user projects based at the Stanford Medical School, user projects based outside of Stanford (national AIM projects), and common systems development efforts. As defined in the resource management plan approved by BRP at the start of the project, the available system CPU capacity and file space resources are divided between these communities as follows:

Stanford	40%
AIM	40%
Staff	20%

The "available" resources to be divided up in this way are those remaining after various monitor and community-wide functions are accounted for. These include such things as job scheduling, overhead, network service, file space for subsystems, documentation, etc.

The monthly usage of CPU and file space resources for each of these three communities relative to their respective aliquots is shown in the plots in Figure 11 and Figure 12. Terminal connect time is shown in Figure 13. It is clear that the Stanford projects have held an edge in system usage despite our efforts at resource allocation and the substantial voluntary efforts by the Stanford community to utilize non-prime hours. This reflects the development of the Stanford group of projects relative to those getting started on the national side and has correspondingly accounted for much of the progress in AI program development to date.

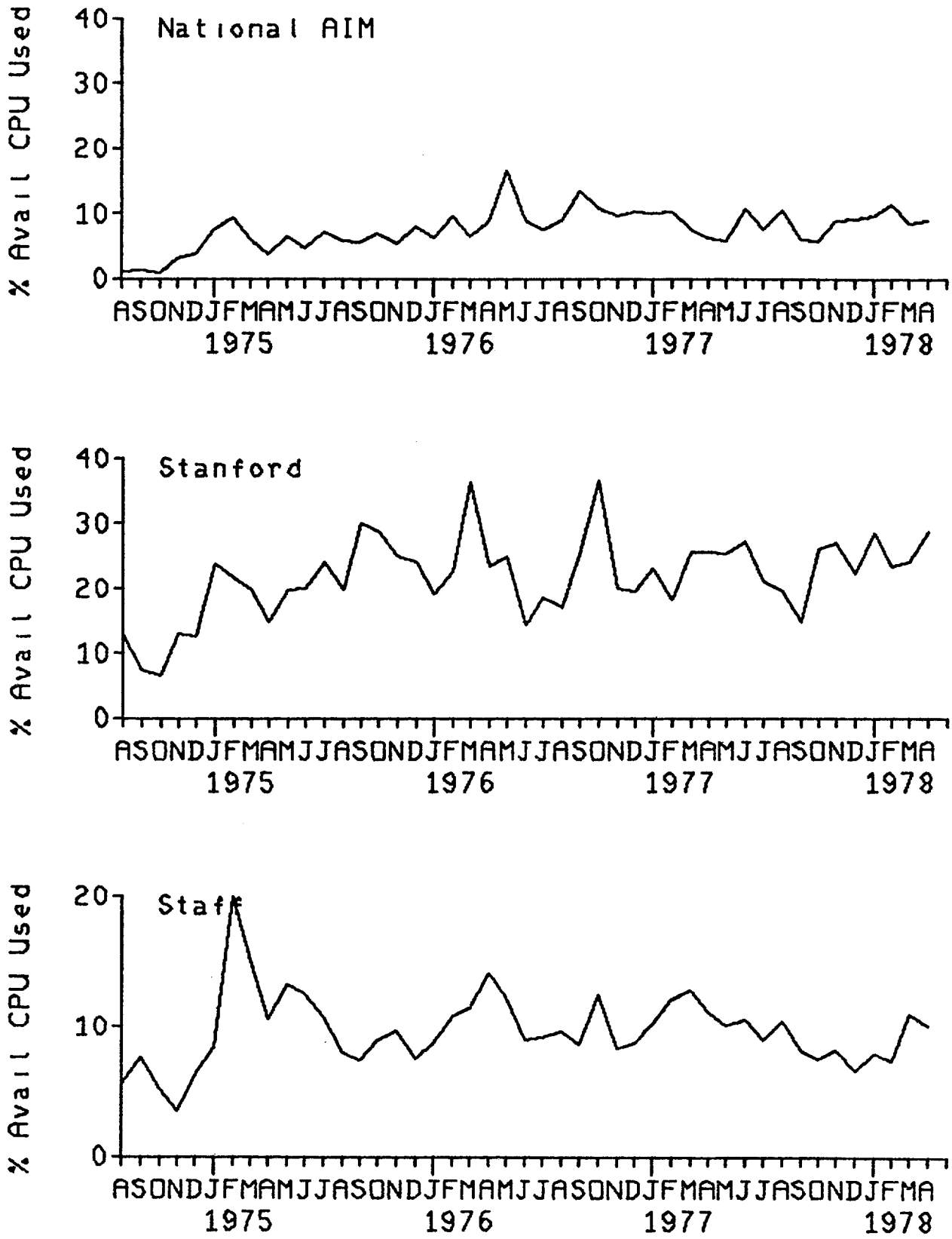

Figure 11. Monthly CPU Usage by Community

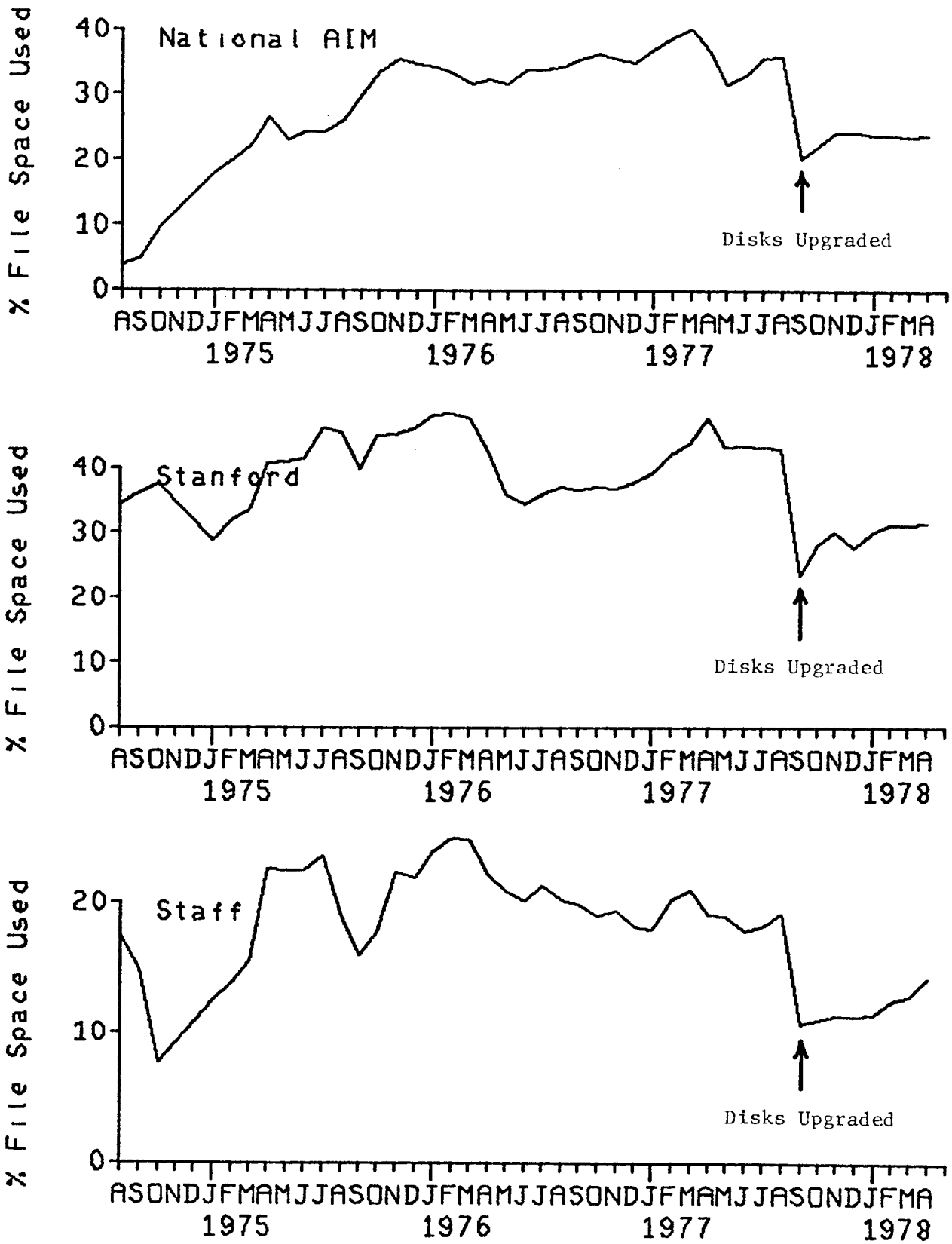

Figure 12. Monthly File Space Usage by Community

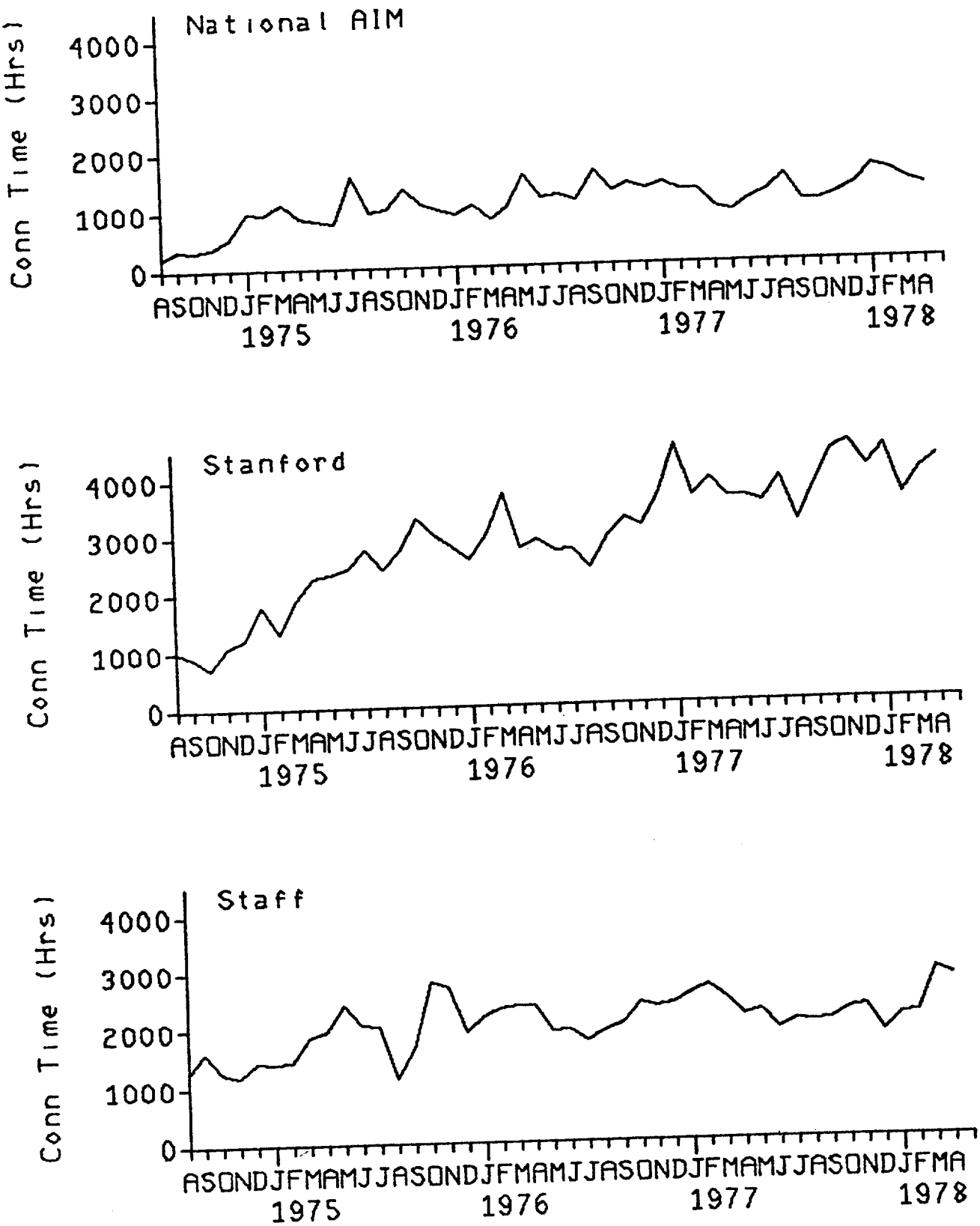

Figure 13. Monthly Terminal Connect Time by Community

2.2.4 INDIVIDUAL PROJECT AND COMMUNITY USAGE

The table following shows cumulative resource usage by project in the past grant year. The data displayed include a description of the operational funding sources (outside of SUMEX-supplied computing resources) for currently active projects, total CPU consumption by project (Hours), total terminal connect time by project (Hours), and average file space in use by project (Pages, 1 page = 512 computer words). These data were accumulated for each project for the months between May 1977 and April 1978. Again the well developed use of the resource by the Stanford community can be seen. It should be noted that the Stanford projects have voluntarily shifted a substantial part of their development work to non-prime time hours which is not shown in these cumulative data. It should also be noted that a significant part of the DENDRAL and MYCIN efforts, here charged to the Stanford aliquot, support development efforts dedicated to national community access to these systems. The actual demonstration and use of these programs by extramural users is charged to the national community in the "AIM USERS" category, however.