Proposal for Continuation of Research in Information Processing

Submitted to
Advanced Research Projects Agency
of the
Department of Defense

Allen Newell
Anita Jones

Department of Computer Science
Carnegie-Mellon University
Pittsburgh, Pennsylvania
September 1975

Principal Investigators:

Allen Newell
University Professor

Joseph F. Traub
Professor and Department Head
Department of Computer Science

Duration: July 1975 to June 1976
CMU Proposal Number: 03027
Submitted: March 1975

The research discussed in this proposal is supported by the Defense Advanced Research Projects Agency of the Office of the Secretary of Defense (contract number F44620-73-C-0074) and monitored by the Air Force Office of Scientific Research.

Abstract

This document is the substantive part of the proposal submitted by the Computer Science Department of Carnegie-Mellon University to the Advanced Research Projects Agency of the Department of Defense for continuation of research in information processing during 1975-76. It contains a description of the major on-going research projects: Artificial Intelligence; Speech Understanding Systems; C.mmp, the multi-mini-processor; SMCD, the symbolic manipulation of computer descriptions; and newly initiated work on multiple microcomputers.

CONTENTS

PREFACE

This document contains the substantive part of the proposal to the Information Processing Techniques Office of the Department of Defense for the period July 75 to June 76. We have reproduced it for more general distribution, since it gives a good overview of some of the current research in the Computer Science Department.

The present version differs from the original in two respects. First, the budget portions of the proposal have been removed, except the part relating to major equipment, where the information is of substantive interest. Second, the original proposal was prepared under considerable time pressure. Thus, the present version has been edited for errors and infelicities of writing. References to our own publications, which did not make it into the original version, have been added. However, the proposal was written primarily to describe our own ongoing projects rather than to provide a discussion of the general state of the art. There were essentially no references to work outside of CMU; we have not remedied this since it would have required substantial modification. Our work, like that of all other scientists, depends on and grows out of work done elsewhere in the scientific community. Our publications, cited herein, give specific credit to these other efforts.

The research going on under APRA is only part of the total research in the Computer Science Department at CMU. It is, in volume terms, a rather large part of it, and thus what is described in these pages is much in evidence. It is important to realize, however, that the research done under ARPA represents a particular style of research: large experimental projects standing at the center of each area, with well defined mileposts and with scientific success tied closely to the success of specific systems. Around these major systems there is much individual effort, but the dominant flavor is imparted by the large experimental systems. Though there are oft-noted dangers to this style of research, we believe that it represents an important style and one that leads to progress on major scientific issues.

However, an inference to the totality of research in computer science at CMU cannot be made from the sample of this report. For much of the rest of the research follows other styles. There is work in operating systems that is experimental, but relatively small (Habermann); there is work in security and protection that is primarily theoretical (Jones); there is work in real complexity theory that is primarily theoretical (Kung, Traub); there is work in the development of programming languages and programming methodology (Habermann, Jones, Shaw, Wulf); There is work on data base design (Eastman, Schkolnick). There is all the experimental and theoretical work in Psychology on cognitive processes, which is closely related to Artificial Intelligence.

We take the trouble here in the Preface to enumerate some of these other research efforts, with their somewhat contrasting styles, to be sure that the reader understands that the extensive description of research given in this document is still only a part of the total research in our environment and, further, is cast in a particular large-system experimental mold.

We would like to acknowledge the many members of the department who helped to create this document and to improve its accuracy. Especially we would like to thank Richard Johnsson and Philip Karlton for assisting with the final document production, and Beverly Howell and Mildred Black for secretarial help.

1. INTRODUCTION

This proposal is for basic research in computer science by members of the Computer Science Department of Carnegie-Mellon University for the Information Processing Techniques Office (IPTO) of the Advanced Research Projects Agency (ARPA) of the Department of Defense under Contract F44620-73-C-0074, monitored by the Air Force Office of Scientific Research (AFOSR).

The proposal covers the eighteen month period from 1 July 75 to 31 December 76. It is for the third year (and half of the fourth) of the existing contract, and is a continuation of a long term research program.

This introduction provides a brief description of the scope of the research. It also provides brief overviews of recent results and our specific research targets for the immediate future. To fully appreciate these results and our immediate goals requires a more robust treatment than the overviews provide. Extended discussion will be found in the sections on each research area. However, it seems useful to bring the overviews to the front, where they can act as quick summaries of where our research program has been making progress and where it is laying its bets.

1.1 General Scope and Organization of the Research

The research proposed herein may be organized under the following headings:

(1) Artificial Intelligence: To discover the nature of intelligent action and to realize such action by computers.

Specifically, as a focus for the coming period:

> In research on total system organization: To determine whether production systems are an appropriate organization for realizing intelligent action in a system designed to operate in a general environment.
>
> In heuristic search: To determine the basic components that make heuristic search possible in realistic complex situations, especially how knowledge is used for strategic guidance.
>
> In automatic programming: To automate data base management and access tasks.
>
> In vision: To determine how to locate and identify semantic objects in large naturalistic scenes.

(2) Speech Understanding Systems: To recognize by computer freely uttered speech.

Specifically, as a focus for the coming period:

> To construct systems capable of understanding connected utterances in a restricted task domain (and meeting the ARPA Speech Understanding specifications by Nov76).

(3) C.mmp (A multi-miniprocessor): To determine effective ways to organize computations on systems with multiple processors.

Specifically, as a focus for the coming period:

> To bring C.mmp to a fully operational state with respect to the hardware configuration, the operating system, the software facilities, and the accessibility to users.
>
> To investigate a series of benchmark programs to analyze the total-system characteristics of multiprocessor systems of the C.mmp class.
>
> To flex the operating system with respect to its ability to support specialized sub-environments, with an emphasis on security and efficiency.

(4) SMCD (Symbolic Manipulation of Computer Descriptions): To describe computer systems so that the full range of analyses, syntheses, and evaluations of interest to computer science can be performed on these descriptions.

Specifically, as a focus for the coming period:

> To create a basic set of tools for the description of computers, including a computer description language capable of expressing mixed levels of abstration, a simulator capable of mixed level simulations, and a global data base.
>
> To analyze the task of a compiler-compiler that works relative to the description of the target computer; and to design and implement such a compiler-compiler capable of producting quality code.
>
> To analyze the task of the design of computer systems from sets of modules; and to design and implement such a computer design system capable of specifying both control and data flow and of working with arbitrary module sets.

The research just listed above is relevant to a substantial fraction of modern computer science: to artificial intelligence, and within that speech understanding, to several areas of software systems, to computer architecture and to performance analysis. It represents research interests of ten faculty members in Computer Science, and several programs within IPTO. From the viewpoint of diversity and breadth, it has several major thrusts, each focussed on independently important scientific questions.

The research is also strongly interdependent, despite its breadth. Each of the major projects represented in the proposal has active ties, and in some cases strong dependencies, on work going on in the other projects. The major personnel in the program are involved generally in more than one of the major projects. This is a reflection of the breadth of research interests and talents of these particular scientists, but it is also a reflection of the unified nature of the CMU ARPA research effort that it has encouraged and permitted this. Indeed, it is our contention that we have found a way to bring to bear adequate concerns and expertise for computer architecture, performance analysis, software systems, and artificial intelligence on research problems of basic significance to computer science. We believe strongly that the present state of computer science, both technologically and conceptually, is

such that all of these areas are jointly essential to making progress on many fundamental scientific problems.

The four projects listed above serve to organize the research environment, with specific faculty members responsible for each area, continuing research seminars in each area, etc. However, this organization does not extend to an administrative structure. Administratively, we consist simply of faculty, research associates and graduate students of the Computer Science Department, with the facilities divided into an Engineering Laboratory and a Programming Group (which is also responsible for computer operations).

The project-orientation of the various components of the research effort is genuine. We have formulated what we believe are important scientific goals in terms of systems to be developed and performances to be obtained. Riding piggy-back on these major goals, of course, are many scientific issues which can only be addressed given the existence of the main system and in some cases achievement of system performance goal.

As is well known, all research is not, and cannot be, so organized into projects. Though computer science lends itself to project formulation (given a certain amount of taste and care in the selection of projects so that their accomplishment truly advances the science), much exploration and individual theorizing and analysis must also go on. Within our environment, this occurs across all the areas covered by the research described herein, and beyond them. It is out of such explorations that our new research efforts arise. Indeed, many of the projects of the present proposal arose in just such a way.

1.2 Overview of Recent Results

Each of the four major projects in this proposal has a different maturity and hence a different standing with respect to its goals. Thus the type of recent results that have been obtained for each project area have a somewhat different flavor. We list each of the major results in each area without trying to make their full context clear. A more extensive treatment will be found in the appropriate section (though not all the particular facts are necessarily repeated). The results are all for 1974, unless specifically noted.

1.2.1 Artificial Intelligence

Production systems: We have several production system languages, exploring different architectural assumptions, and have coded several tasks ranging up to 250 productions. We have produced several simple production systems that grew themselves, i.e., learned.

Heuristic Search: New theorems were obtained for optimal search strategies in types of task environments quite different from those of previous theorems about search.

Vision: Segmentations of 600*800 pixel three-color (24 bits/pixel) natural scenes have been obtained using a new technique (region splitting) that has not been much explored.

Automatic Programming: A program has been created within the Buchanan-Luckham automatic proving system (created at Stanford) that creates relational data bases defined according to the CODYSYL Data Base Definition.

1.2.2 Speech Understanding Systems

Hearsay-1.X. The system has not been appreciably improved over its capabilities in the Nov73 mid-course evaluation, though its performance has been analyzed. Current performance (in words correctly recognized) is 39% for acoustics alone, 70% for acoustics+syntax, 93% for acoustics+syntax+semantics (Chess task only). Sentence recognition is correspondingly lower, e.g., 90% for the all three knowledge sources. The figures are for small vocabularies (20-50 words); the system is insensitive to vocabulary variation at this size, but degrades appreciably at 250 words (33% word recognition). The recognition runs in about 10 times real time on a KA10 (and potentially at real time on a dual processor KL10).

DRAGON (alternative scheme based on Markov representation of knowledge) became operational with acoustics+syntax. On same five tasks as Hearsay-1 DRAGON obtained 83% word recognition, in about 50 times real time.

Hearsay-2 became operational and recognized its first sentence using 9 knowledge sources (Nov74).

Hearsay-2 on C.mmp: L*, the implementation system for Hearsay-2 on C.mmp, became operational (Jan75).

SPS-41 High Speed Microprocessor: The first faster-than-real-time program for analyzing speech signals using the LPC spectra was demonstrated (Apr74).

The full dynamic range 16-bit ADA Conversion device, specially adapted for speech, was demonstrated (Apr74). It should replace the analog adaptation techniques currently in use.

1.2.3 C.mmp: the Multi-miniprocessor

The 16*16 switch became operational in spring 74.

In Dec74: (1) The hardware was operating with 5 PDP11/20s and 512K primary memory. (2) The HYDRA kernel was operating routinely, and the initial subsystem was in a very early operational state. Regular user periods of 3 hours/day were initiated, though total (hardware + software) mean-time-between-failure became unusably short as the number of simultaneous non-expert users grows to 3 or 4.

An analysis based on static and dynamic measures of code size and speed on the KA10 and C.mmp indicated that a full 11/40 version of C.mmp is about 6 Mips (millions of instructions per second) while a KL10 is about 1.4 Mips; and that the cost-performance of C.mmp (measured in Mips/Mega$) is about 2-3 times that of a KL10. (Memory costs and assumptions about primary memory capacity required to support effective use of processing power have strong effects.)

The C.mmp Hardware Monitor designed and constructed at CMU became operational (Oct74).

The PDP11/40 extended at CMU to include a writeable microstore became operational (Oct74). This programmable processor is destined to be the the main processor type on C.mmp. It was developed with partial support from DEC and NSF.

1.2.4 SMCD: Symbolic Manipulation of Computer Descriptions

(Research effort initiated in Jul74.)

ALPHARD, a new programming language for structured programming being developed at CMU, will be the basis for the new computer description language. ALPHARD itself is at an advanced specification stage, usable for manual analysis.

Related to the Machine-relative compiler-compiler: A book-length monograph on the BLISS11 optimizing compiler structure was finished (published Jan75).

Related to Machine Design with Module Sets: EXPL, a system capable of considering variation in control flow in designing structures using RTMs (Register Transfer Modules, a commercially available module set) was completed in early 74 (supported by an NSF grant). An extension to EXPL that designs structures using macromodules (Washington University's module set) is in the advanced debugging stages. EXPL is the nucleus system for the SMCD effort.

1.3 Overview of Research Targets

We give here the main definite targets for our research efforts for each of the four projects. In the body of the proposal we enunciate for each subproject its own special targets and the higher scientific context into which it fits. Some targets are strictly means to our own projects, but most relate to independent scientific issues. We do not reproduce all this here, for the elements in it are of variable interest and specificity. Rather, we give here the main targets for which we are prepared to be specific as to accomplishments expected and dates.

A cautionary note is in order: Such a list favors the scientific targets that can be tied to the development and completion of systems (e.g., to C.mmp) or that will arise inevitably from straightforward study (e.g., the comparison of two programmable microcoded computers). That science which we have not yet thought of will not be represented -- yet we would hope the latter will account for a significant fraction of our scientific productivity during the coming period. But assuming the following list is recognized for the truncated representation that it is, we present it.

1.3.1 Artificial Intelligence

Production systems: By Aug75 we expect to have finished a study that compares (operational versions of) production systems for many classic AI tasks against the original implementations within other program organizations.

Production systems: By Aug76 we expect to create a production system with 1000 productions to perform some frontier AI task. (Production systems, computer system, and task are to be chosen by Aug75.)

Heuristic search in complex tasks: By Dec75 we expect to have a lemma system for guiding search working in our chess program.

1.3.2 Speech Understanding Systems

Hearsay-2: By Nov75 we expect it to be working at 90% word recognition level, and 70% sentence level on the 200 word vocabulary Associated Press News Retrieval Task. We expect it to be working, but not well, on a 1200 word vocabulary task. By Nov76 we expect it to meet the APRA 5 Year Performance Specifications.

DRAGON: By Nov75 we expect to have a fast (SPS41) version working on a 1200 word vocabulary, but with no performance expectations. By Nov76 we expect it to meet the APRA 5 Year Performance Specifications.

Hearsay-2 on C.mmp: By May75 we expect to make a decision on proceeding with the current implementation. Given a positive decision, we expect the system to become a faithful replica of Hearsay-2. By Nov76 we expect to be able to show the computational tradeoffs for a C.mmp-like multiprocessor implementation.

1.3.3 C.mmp: the Mini-multiprocessor

By Sep75 we expect the total C.mmp system to be in routine and reliable operational use by multiple non-expert users.

By Dec75 we expect to have in operation a system with 16 processors (4 11/20s and 12 microprogrammable 11/40s) and 1 million words of primary memory. This will be a system at about seven eighths of the maximum 5.95 Mips possible with sixteen 11/40s.

By Sep75 we expect to have initial versions of a collection of benchmark tasks on C.mmp, to be able to assess its computational character.

1.3.4 SMCD: Symbolic Manipulation of Computer Descriptions

By Dec75 we expect to have a first implementation of ALPHARD usable for SMCD purposes.

By Dec75 we expect to have a first specification of the machine-relative compiler-compiler, with a possible expectation of a running version by Aug76. This implies the completion of a series of studies on the components of the compilation processes to abstract them to work from computer descriptions.

By May75 we expect to have a specification for an register-transfer level computer aided design system, which works with arbitrary module sets (at the register-transfer level) and designs over both control and data flow. Its expected date of completion depends on the specifications.

By Apr75 we expect to have a comparison of the MLP900 and the microprogrammable 11/40, to select a path to follow for the high speed portion of the variable level simulator associated with ALPHARD.

1.4 Organization of this Proposal

The proposal consists of eight sections including this introduction as Section 1. Sections 2 through 5 describe each of the four research projects. Section 6 describes the facilities.

Sections 7 and 8 describe two supplementary proposals, one for completion of the physical components of C.mmp, the other for a support in the area of multiple computers.

Thus the main content of the proposal is given in the four research sections. For each of these we describe the goals of the scientific research, both general computer science goals and specific objectives of the project. We then give the present state of the research, noting the accomplishments we have attained within the last year (1974). Finally, we describe our plans for the research for the coming year and note any specific resources needed.

2. ARTIFICIAL INTELLIGENCE (AI)

2.1 Introduction

CMU has a long history as a center for research into the nature of intelligence, both in humans and computers. It is often referred to, along with the laboratories at Stanford, MIT and SRI, as one of the four ARPA AI Laboratories (though the work under this contract is much broader than AI, as can be seen from this proposal). The totality of research at CMU that should be discussed under the rubric of Artificial Intelligence actually occurs in three separate places: the work on human cognition, going on in the Psychology Department and supported by other than ARPA funds; the work in Speech Understanding Systems, which is described in a separate section of this proposal; and the work described in this section, officially labeled "artificial intelligence".

The work described herein has several distinct concrete objectives, representing approaches to different aspects of the total puzzle of artificial intelligence. [Newell 72a, Simon 71] To put them into perspective we will provide in this introduction a brief discussion on the general scientific goals of artificial intelligence.

2.2 The Scientific Goals of Artificial Intelligence

Artificial Intelligence is the scientific specialty that attempts to understand the nature of intelligent action and to construct systems that are capable of intelligent action.

"Intelligent action" is shorthand for describing the capability of some systems to bring knowledge to bear in the service of ends in an effective way. As is true of any science-defining term, precise definition is not required. Plentiful examples of intelligent action, varying in many aspects, are all that are required. Science, when successful, evolves adequate theories of its domain; it never starts with them. The behavior of some humans some of the time, and (by now) the behavior of some computers some of the time, provide the empirical base on which to erect the science.

As with all sciences, the goals of AI at a particular historical moment reflect three sources: (1) the universal questions asked by any science; (2) the state of scientific knowledge at that time; and (3) the techniques available to answer questions.

AI's version of the universal scientific questions:

Q1. What are the phenomena: What are the types of intelligent action?

Q2. What gives rise to the phenomena:

Q2.1: What are the characteristics of systems that permit (or enable) them to show intelligent action?

Q2.2: What are the characteristics of the environments of tasks that require intelligent action?

Q3. What is the genesis of the systems that produce the phenomena: How do systems capable of intelligent action arise?

The following propositions summarize very globally the experience of the last twenty years of work (which in fact covers essentially the total relevant scientific history). They induct over investigations in many task environments. Being empirical generalizations, they make no claim to be exhaustive or complete.

P1. An essential condition for intelligent action of any generality is the capability for the creation and manipulation of symbolic structures.

> To be a symbolic structure requires both being an instance of a discrete combinational system (lexical and syntactic aspects) and permitting access to associated arbitrary data and process (designation, reference or meaning aspects).
>
> It is this proposition that makes AI a subdomain of Computer Science, since the computer is most appropriately viewed as that man-made system which is capable of symbolic representation and manipulation.
>
> This is perhaps the most profound discovery of AI (and Computer Science): That what was implicitly created in digital computers was a physical symbol system, one which satisfied the fundamental prerequisite for intelligent action.

P2. Intelligent action requires bringing to bear very large amounts of highly diverse knowledge.

> Stated as an impossibility axiom, there is no way to obtain more than limited intelligent action from a limited data base, no matter how much computation is applied. Therefore, large intelligence requires large amounts of knowledge.
>
> The problem of representation is fundamental. The problem has many facets besides simply encoding knowledge: the acquisition of new knowledge, its assimilation to existing knowledge, the accession to knowledge that is relevant to the task at hand, the conversion of the knowledge so accessed to increase knowledge about the immediate task, and the derivation of new knowledge from the body of knowledge already acquired.
>
> The representation of knowledge in symbolic structures gives rise to the fundamental dual nature of information processing systems into a processing structure and content (i.e., particular symbolic structures).

P3. The fundamental response of a system to uncertainty -- to how to proceed with a task or how to attain a goal -- is to create a space within which a resolution to that uncertainty must lie, and to search that space.

> The problem of search is fundamental. It shows up in many guises and with many variations. Almost all the basic methods used by intelligent systems can be seen as some variation of search, responsive to the particular knowledge available.

P4. The act of perception -- of the formation of an intelligible representation of an external environment -- requires knowledge equal in breadth and extent to that involved in subsequent use of the resulting representation.

> Thus, perception is not to be distinguished from cognition as if it were a separate preprocessing stage of simpler design. The full knowledge available in the system must be able to inform the act of perception.

P5. The control of behavior towards ends can be obtained by the use of goals -- i.e., symbolic structures that encode the variety of knowledge pertinent to ends: under what conditions it will be obtained, what aspects of the task environment are relevant, what methods are relevant and have been used, etc.

Finally, the current art specifies rather strongly the techniques available for obtaining new scientific knowledge.

T1. AI's major technique for discovering new knowledge and for verifying existing knowledge is the construction of systems that exhibit intelligent action in specific task environments.

> To discover new forms of intelligent action, specify a previously unexplored task environment and construct an intelligent system to perform its tasks.
>
> To verify that a collection of mechanisms for intelligence have certain properties, construct a system embodying these mechanisms and see whether they produce the intelligent action predicted.
>
> The constructed systems of AI (in the current art) are accessible for exhaustive examination and rational analysis. Hence, verification of scientific knowledge can often be accomplished by detailed examination of a single instance. This is a striking property vis a vis most sciences, whose systems are much less accessible.

T2. Humans, as the existing naturally intelligent systems, can be studied to discover the types of intelligent actions they perform and the mechanisms used to perform them. The major source of knowledge is the experimental study of human behavior to discover mechanisms, followed by construction of simulation systems to verify whether the discovered mechanisms do indeed have the properties inferred.

> Currently, the complexity of human structure (the anatomy and physiology) precludes direct insight into the structure of these mechanisms. The human system is a prime example of one with inaccessible structure, so that elaborate experimental techniques and designs are often required to exclude alternative explanations of phenomena.

T3. The current state does not provide formal theories of any power that can be used to structure the main course of scientific development of AI, though formal theories do exist for fragments of the field. Scientific knowledge is most well developed at the level of specific mechanisms of intelligent action and their effects in task environments of (partially) specifiable characteristics.

AI (and computer science generally) is both constructive in method and basically concerned with system behavior that attains ends. Thus, unlike natural sciences, there is little separation between the pure and applied aspects of the science. To produce systems capable of intelligent action is to produce systems capable of attaining ends, which is to say, systems that are useful according to someone's criteria. The distinction between pure and the applied aspects rests primarily on the particular task environments studied. Pure scientists choose task environments according to the requirements of discovery or verification of various scientific propositions. Applied scientists choose according to the usefulness as determined by by externally stipulated goals and criteria.

With this overview as a background, we can describe in a succinct way the particular goals of the research currently being conducted in AI within the present proposal.

2.3 Production Systems: An organization for intelligent action

2.3.1 Scientific goals

What is the appropriate organization for a system capable of general intelligent action? This is one of the fundamental questions in AI. We have learned, both in computer science and in AI, that a system can be divided into a control structure, on the one hand, and the content (i.e., the symbolic structures) that resides within that structure, on the other. The prime implication of such a division is that, in an important sense, the organizational structure is contentless, consisting of extremely general processing and encoding mechanisms. As we observed in the general discussion above, this arises from the essential nature of symbolic representation itself.

The extraordinary usefulness of programming languages rests squarely on this principle, as does the very organization of computers into an architecture and a program residing within that architecture. Two of the most important advances in AI have come from taking this view: the development of list processing at the very beginning of the field, and more recently the development of the so-called Planner-like languages (Planner, QA-4, Conniver, Poplar). There is, of course, no indication that the question of organization has a unique answer, nor is that important. What is important is that we discover and then understand the basic organization required for intelligent action.

A candidate organization is that of a production system [Newell 73a] and it is a specific goal of our current research to determine whether that is the case. We will first describe briefly this type of organization, and then return to pose the scientific question in sharper terms.

A basic production system consists of a set of elements (the productions) and a set of symbol structures (the working memory). Each production consists of a condition part, which is sensitive to the contents of the working memory, and an action part, which can modify the working memory. The basic operation cycle is "recognize and act" -- namely, recognize which productions have their conditions true of the current state, select one of these, and then take the corresponding action. Since the action modifies the working memory, indefinite repetition of the cycle moves the system through a trajectory of processing behavior.

A production system is thus a parallel recognizer and a serial actor. The conditions are taken to be computationally simple enough so that the selection of a production happens in unit time and independently of how many productions exist in the system (tens, thousands or

millions). The actions are also relatively simple: all conditionality in the system occurs through the recognizer, so that actions are not complex subprograms capable of executing arbitrary sequences of conditional subactions.

All programming systems have conditional expressions, often of the form "if C1 is true, do A1, if C2 is true, do A2 ...", which is similar to the structure of a production system. Similarly, the notion of having sets of conditional actions sit around being continuously sensitive to the body of data has been receiving considerable attention; they are often called "demons" after an early organizational scheme of Selfridge's called Pandemonium. Demons are a production system like organization. The characteristic feature of production systems is to take this type of control organization to the limit, rather than have it be simply one control mechanisms among many. The recognize-act cycle becomes the most elementary cycle in the total system, replacing the fetch-execute cycle basic to most other systems. This has profound effects on the system and affects how it might solve a number of its basic problems.

Several important characteristics of productions systems should be mentioned. First, they provide a model of the basic human information processing organization. The production memory is analogous to the human so-called Long Term Memory; the working memory is analogous to the human verbal Short Term Memory [Gilmartin 75]; the recognition-act control cycle corresponds to the human action cycle of 50-100 ms. This model has achieved some success in describing a variety of human behavior. This correspondence to human organization is of prime importance and is serving as an important guide to determine additional organizational details of production systems.

Production systems provide a homogeneous encoding of knowledge in an active form. [Newell 72] They represent all control explicitly in terms of the content of the Working Memory, a feature which may prove to be extremely important to the self programming and debugging of productions systems. [Waterman 74] Production systems provide an openness to interruption, to error detection and corrections, and to bringing to bear the total knowledge available in the system. Other control structures tend to isolate structurally distinct operational environments, controlling unwanted interactions, but not admitting unplanned but useful knowledge. Production systems provide a simple mode of growth and augmentation (i.e., simply add the production to the set), though no one has yet really capitalized on this apparent advantage.

The consequences of real success in determining an appropriate organization are very substantial, both in theoretical and in applied terms. This is true whichever of the various organizations now being explored (or some new one) turns out to be effective. But we can use production systems to illustrate the impact, since these are the scheme of interest here.

If production systems prove effective, we will have a standard way within which to accumulate knowledge in the service of a task. In the construction of applied knowledge-based AI programs, we will have a standard way to structure them, so that the job of building such programs can look much more like a knowledge-engineering problem than like a system-design problem. This is already the approach being taken by others workers in AI (such as Feigenbaum at Stanford), and their work is beginning to provide a test of this notion. (That one should be able to begin applying an idea even before it is very thoroughly worked out should not be a surprise.)

Trading on our preliminary understanding of production systems, one can see that such an organization might really change the terms on which programs operate, via the system's own continuous monitoring and recovery from error and difficulties, by in some sense

understanding the computation it was working on. [Waterman 75] That is, productions embodying error detection and correction knowledge would continuously be sensitive to the ongoing computations. Production systems would also lead to a form of programming, which can be called "incremental programming", in which one starts by adding productions corresponding to the main functions of a program and then filling out the rest of the program (simultaneously debugging it) by adding productions that deal with the failure of the partial program to operate. This is similar to the notions being explored at the MIT AI Laboratory (by Sussman notably). With production systems, one can see how incremental programming may well become the standard way of programming. The processing assumptions underlying productions are sufficiently specialized that they clearly would lead to machine organizations that realize them directly (see the subsection below on efficiency).

Productions systems are only one candidate among several currently under intensive investigation in AI. Two other widely investigated organizations are Planner-like systems and Semantic-Net systems; the most recent scheme is that called frames. All of these explore important variations in basic organization, which will be tested only by extensive experience and development of each basic scheme.

2.3.2 Specific Goals

We can now turn to the specific subgoals that currently drive our attempt to discover the properties of productions systems and to assess their worth as an organization.

Goal: To discover the appropriate specific form for production systems. Although we can enunciate some interesting and useful features of production systems, the fact that they seem to model the human organization is of key importance. For the question is how we should complete the design of the basic architecture -- beyond those few features (such as the recognize-act cycle) which now seem clear. We would wish to complete it in a way that captures the flexibility of the human for dealing with partial knowledge and for handling error. Thus, from a research point of view we wish not simply to pick a specific total design (we have of course created several operational systems for experimental purposes already), but to explore the space of production systems and to attempt to find the regions in that space which correspond most closely to the human system. Though not part of this proposal in terms of support, the parallel investigations into cognitive psychology going on at CMU are a requisite part of this stragegy.

Goal: To construct production systems for a range of intelligent systems. The main line of approach is the basic AI experimental strategy of constructing systems and analyzing their behavior. Our goal here is one step more analytical than usual, in that we will construct versions of many existing systems so that we can do a comparative analysis. We thus wish not simply to develop production systems pell-mell, so to speak, but to spend substantial effort in analyzing them. At some point, of course, this goal calls for creating substantial production systems to accomplish a major task requiring intelligent action.

Goal: To discover efficient implementations of production systems. A moments consideration shows that the obvious way to realize production systems on a serial machine is to iterate through the productions at each recognition cycle. This is not a tolerable (much less efficient) implementation for large sets of productions. One wants the cycle to be very short (ultimately at the microsecond level for machines) and the number of productions to be very large (in the millions). Thus, an intensive analysis of how to realize production systems, both in software on serial computers and in hardware in parallel systems, is a necessity.

Goal: To discover how productions systems can grow themselves. Though we now know how to get productions to make additions to themselves (i.e., to learn and grow) with deliberate production-construction actions and for task situations which lead to rather simple uniform modes of growth, we still do not know how to obtain general growth. This is one of the major challenges of production systems and one that will be a focus of effort until we finally solve it.

2.3.3 Current status and Recent Results

Basic production systems are not difficult to create for exploratory purposes, given programming systems with adequate facilities (e.g., Lisp or L*). We have constructed several to try variations in architectural assumptions (two, PSNLST implemented in LISP, and PSG, implemented in L*, are essentially public systems, the latter also permitting substantial variation of the architecture through parametrization).

We now have constructed several production systems in the 200-250 production category. These include a system that performs essentially identically to STUDENT [Bobrow 68], a program developed some years ago to solve word problems in ninth grade algebra, and two systems which perform visualization tasks, one in which a person imagines a structure in his mind's eye [Moran 73] and the other in which a person describes an object he can see and scan only through a small peep-hole. [Farley 74] (These latter efforts were attempts to model the behavior of humans on these tasks.) In addition, we have done a large number of smaller tasks, such as elementary psychological experiments, puzzles, parsing, and some learning tasks (see below). We are in the midst of doing some detailed analysis on some of these program (especially the ones that perform identically to an existing AI program), but are not finished yet.

As noted, the question of how production systems learn or grow is an important issue, especially since some attention has been devoted to it for quite awhile without striking success. This year we have developed some schemes for growing productions and have created growing systems: one that mimics EPAM, a system for learning verbal materials that is a psychological theory of human learning, some that learn simple sequential concepts, and some that grow a semantic net. These must be seen as exploratory exercises, since the learning schemes themselves seem much too deliberate in the way they construct new productions and salt them away in the existing system. But they represent genuine progress over a year ago.

We have developed several computational schemes for realizing productions systems efficiently. One scheme (a variant of PSG called PSH) capitalizes on the immense time-redundancy of the truth of conditions; namely, the features of the working memory that make a production condition true change only slowly with time. We are currently testing this system and doing an analysis of its algorithm.

2.3.4 Plans

The study of the nature of production systems that we have been carrying on for the last year should be completed by Aug75. This study is based partly on a thesis (involving PSNLST) and should be a fairly substantial analysis of the problem.

Assuming for the moment the general conclusion of the Aug75 study, we will select a substantial task to use as a major driver for the subsequent work in production systems. The key design parameter on the production system itself (and implicitly on the complexity of the

task demands) is the number of productions. We cannot yet calibrate this measure against others one might use elsewhere (e.g., code size, or fact size of a data base). We do have, for the STUDENT production system, a characterization of how many propositions are required to hold all the knowledge implicit in the production system itself. The same order of magnitude is required: 218 propositions vs. 257 productions, though the mapping is many-many with several productions reflecting each proposition and several propositions required to support a production. Thus, we can take a production to correspond very roughly to a proposition.

We intend to construct a system of about 1000 productions. This should correspond to a program about the size of Winograd's natural understanding program on the toy blocks world. The selection of the task and the production system in which to create it should be accomplished by Aug75. We should have the system itself up by Summer 76.

Our general plans are to set goals for increasingly large production systems, since the issue underlying production systems as an organization (and all other candidates as well) is how to operate with large amounts of knowledge.

It is important that we be able to run large production systems in real time. Thus along with the construction of production systems will go the attempt to make them very efficient. We plan to explore the implementation of production systems on parallel computing structures, such as C.mmp. These plans are not so advanced that we are able to set targets. We do, however, expect to have the analyses of the efficient realization of such systems by programming means on uniprocessor completed by Jul75.

2.4 Heuristic Search in Complex Spaces

Search permeates all attempts at intelligent action, and the study of search will be a fundamental area of AI indefinitely. There is no simple single "problem of search", such that one could hope to solve it, thus putting the matter to rest. For the general question is how to apply whatever knowledge one has, however imperfect and obscure, to generating the directions through which search might proceed and then guiding the choice of path. As the structure of the task environment and the types of knowledge available change, the "problem of search" will require distinct scientific treatment. Thus the study of search in AI is a scientific subfield, in which we can expect a whole body of knowledge to develop that provides a basis both for understanding the contributions which search makes in any particular task and for designing effective search components of total systems which behave intelligently.

The early work on search (which in fact established AI as a scientific field) did produce some general notions, which are now embodied in the basic methods of AI, such as Generate and Test, Hill Climbing, Means-ends Analysis, Heuristic Search, etc. These methods show up repeatedly in all AI programs and applications, and form the basic tool kit of general techniques. No sooner was heuristic search used in a clear way in the first theorem proving programs (LT), than it was applied to theorem proving in other areas (geometry), then to symbolic mathematics (integration), then to some management science problems (assembly line balancing, and production scheduling), then to a variety of common sense reasoning tasks and puzzles. The very way these search methods swept through a wide range of tasks indicated their fundamental nature.

Additional methods are being discovered less frequently, but recent work of Walz at MIT on a way of manipulating multiple sets of constraints in a visual scene interpretation task

appears to be another one. We have called it the Range Restriction method, and have discovered that it is the method operating in several tasks (where one thought there was only an ad hoc search scheme).

We are currently working in two areas which are directly addressed to important general questions of search. One is how to apply knowledge in a complex situation to control the search (the early work, and so also the basic methods, are all bare-bones schemes and do not touch the problem at all). The other is how to obtain some theoretical basis for characterizing search situations. We take these up separately.

2.4.1 Guiding Search by Strategic Knowledge

The problem is how, in a task situation of real life complexity, the diverse knowledge gets brought to bear to guide the search for solutions. Such a question cannot be approached in the abstract, since one needs to investigate situations where detailed and highly interrelated knowledge is available to the problem solver, and to discover how such knowledge might be used -- the very antithesis of an abstract situation. One of course hopes to discover some of the basic processing that is being used, so that this can be extracted and then used in other task situations. One approaches such questions in what has become the hall mark of AI research: selecting a specific task, constructing a program for it, and then analyzing that program to see why it was able to function. It is often possible (recall the remarks in the overview of AI goals) to obtain genuine new scientific knowledge from the analysis of such single cases. (They are never quite single cases, for variation of program structure and of task structure is always used to help in the analysis.)

We are using chess as our task environment. It has been a scientific intuition of the AI community that chess is a task situation that will yield a rich harvest of scientific results for AI. It combines a situation in which the search components are prominent (we all search out the consquences when choosing a move) with an immense amount of deep knowledge (which is why the novice often cannot understand why a master moved the way he did). We have not been disappointed, as the development of alpha-beta shows. Thus, chess has become a classical task in AI for the study of search, which has by now a background of scientific results that makes it an exceedingly useful task. [Berliner 73] [Berliner 75b] [Chase 73]

Let us illustrate the situation by discussing a new mechanism for search, which we call lemmas. [Berliner 75a] We are still developing this mechanism, so that our treatment here has the form of a scientific expectation, not of a verified result.

An acute problem in chess searches is that one repeats the same search over and over again. The same complex threat and counter threat situation is examined many times (sometimes hundreds or thousands in actual chess programs), with almost always the same result (the defender is safe, say). Almost all the total search effort may be devoted to such redundant analyses. The situation is not actually identical (that would be easy to deal with); rather, some small variation has occurred (for chess programs, the moving of any piece at all on the board), and the consequences of this variation must be worked out. The idea of lemmas is to extract from a situation the characteristics upon which the key action depends (the threat), to demonstrate that only specific changes in these actions can change the evaluation of the situation, and then to establish that lemma as a piece of knowledge that can be used in examining all further search, so that reexamination never takes place until it is necessary. There is, in effect, no more repetition, though one has paid for this by an act of analysis in developing the lemma.

When to re-examine a changing situation is a ubiquitous problem, not limited to chess or even to game-like situations. Likewise, the fundamental notion behind lemmas -- to extract a characterization in a form appropriate to avoiding reexamination -- has equally wide scope. It is of course easy to construct trivial abstract situations in which a notion of lemmas can be developed, but without solving the problem of lemmas in realistic situations. Chess genuinely avoids that. It is in fact the case that no non-trivial lemma-like scheme has been developed for heuristic search. We expect that if we are successful in this attempt to develop lemmas in search that it will lead the way to the use of the mechanism generally in AI programs.

2.4.2 Theory of Search

Some new theoretical results have been obtained relating to search in structured spaces for all-or-none goals. [Simon 74] These are quite different search problems than the least-cost or shortest-path search problems that have received most theoretical attention in artificial intelligence. The main result is the construction of an evaluation function on the elements of the space that tells the order in which nodes of the space should be searched. The given data is the probability of finding a goal at a node and the theorem reflects the expected value per node taking into account the structure of the space in terms of what nodes can be reached from what other nodes

This work is an instance of the attempt to construct a theory of search. Theory in AI, as it currently exists, is a patchwork. The fundamental difficulty should already be apparent. Search occurs in an immense diversity of situations (we have argued in the overview of AI goals that it occurs in all attempts at intelligent action). It is controlled by knowledge, and both the details of how it proceeds and its ultimate efficacy depend on the content of the knowledge. But theory most easily develops by abstracting from all such details to some general feature of the situation which has a determining effect on the success or speed of search. Normally this just abstracts away from what is in fact important. Thus, only slowly, does the AI field find ways to build a theory.

The present work is such a contribution and we cannot tell yet where it will lead to. We know it has some applications in some research management situations (for we have discovered similar formulations being developed there). We shall be searching for others, as well as trying to extend the formulation. But this is a good example of an area in which developing concrete expectations for future results is extremely difficult.

2.5 Automatic Programming

The ability to construct programs automatically, i.e., by means of other programs, is an important technical goal, shared by artificial intelligence and by programming generally, and leading to the emergence of a field of automatic programming in its own right. Several systems for constructing programs automatically now exist at varying levels of competence and taking varying types of initial specifications. We have used one of these systems (the Buchanan-Luckham system, developed at Stanford in the AI Laboratory) to conduct an investigation of automatically programming sophisticated forms of data bases for management information systems. [Gerritson 75]

Explicitly, we have taken the CODASYL model for a data base with its DDL (Data Description Language) and DML (Data Manipulation Language) as ways of describing a data base. These were formulated in terms of the primitives of the automatic program generation system. It is then possible to generate automatically the programs for various generalized

retrieval requests within the context of a specified data base system. These queries constitute in fact the types of programs that data base programmers do write (e.g., in COBOL). It is also possible to start from the queries themselves and induce the relations that should form the appropriate data base to answer the queries efficiently. The system is a research vehicle, not a production program, but it is operational.

This work has the appearance purely of an application -- of taking some results in AI (in this case an actual system embodiing a set of notions for how to synthesize programs) and finding an arena in which they might do a task of real world interest. The effort certainly is that (or at least a major step in that direction). But it is also more. The critical scientific questions in AI revolve around how knowledge is brought to bear to get intelligent action. To investigate the question requires investigating situations of various structures of knowledge -- abstract simplified situations will not do, not because they are not interesting, but because they simply have different knowledge available. Each so-called application area (here management data bases) is an arena in which the mechanisms discovered in AI can be tested against new patterns of knowledge. Thus, we we view this work as making important scientific contributions as well as having important applications.

Current plans for the continued work at CMU in this topic are in momentary abeyance. The faculty member involved (Buchanan) will be on leave this coming year, and the graduate student whose thesis was the development of the system (Gerritson) has joined another university, where he will continue development of the system.

2.6 Vision

Perception must occur whenever there is an external environment that provides a source of knowledge of interest to a system trying to perform some task. Unity in diversity operates here, as it does in many other areas of science. Each environment has its own characteristic structure -- the rates of information influx, the types of laws that relate one part of that influx to another, the levels of invariants that can be extracted and the types of noise that frustrate attempts at extraction. Each environment, then, is diverse and must be dealt with on its own terms. But equally, perception exhibits a set of common functions that must be performed and common mechanisms and methods that can perform them (if adapted) in all perceptual domains. So there is unity, as well. It is the goal of a work in AI in perception to discover and understand the total collection of perceptual functions and mechanisms.

At CMU, speech has provided the main area for investigating perception (as described in the section on Speech Understanding Systems). But we have carried along a small effort in vision as well. By having an active investigation of a similar but different perceptual domain, we provide ourself with an important perspective on our speech work. It helps to abstract from the speech work what is fundamental to perception generally, and to avoid attaching the wrong importance to things which are unique to speech. That our formulation in speech of the system and its task is not so embedded in the particularities of knowledge about speech (e.g., phonology, co-articulation effects) is attributable in part to our maintaining a small counterbalancing effort in vision.

The motivation just described justifies having an investigation on vision. It also justifies as a general focus considering the problem of image understanding in direct analog to speech understanding, namely, how to bring to bear all the sources of knowledge relevant to a visual environment to interpret it relative to a task to be performed. (See the discussion in the SUS section, all of which is relevant to vision as well.) However, it does not determine its specific scientific content, which must arise from the state of the art in visual perception research.

Research in scene analysis has made its most thorough progress in plane-bounded scenes and the time is ripe to move to working with natural scenes, which abound is soft edges and textures. There is some consensus on this among AI workers in visual perception. The difficulties are well known: the size of image increases (the input data-rate problem is a problem of all perceptual systems), edges become much less important, textures, (which are a class of not well understood visual patterns) become very important. Thus our goal is to discover how to interpret natural colored scenes.

2.6.1 Current state and recent results

We have a set of programs for working with images. These are formed into a total system for image understanding as a man-machine scheme. This allows us to investigate particular aspects of the processing in the context of a total system.

A major emphasis in our recent work has been the segmentation problem. [Ohlander 75] Given the raw image, which comes as an array of picture elements (600*800 pixels, each with 24 bits of color and intensity information), the image must be decomposed into regions corresponding to the entities of interest in the image. In one of the natural scenes we are using these are doors, windows, roofs, shrubs, sidewalks, etc. of a photograph of an ordinary a city house. Recently we have been able to perform such segmentation with substantial success using a technique called region splitting, which subdivides a given region by looking at the changes in the histograms of each visual feature taken over the hypothesized subregions. The scheme requires an hypothesizer, so that it makes sense within the context of a total image understanding system. By this technique one is able to successively segment an image, breaking it into a variety of subregions of interest.

2.6.2 Plans

Our image understanding research has advanced to the place where we think it is appropriate to undertake a major attempt at an image understanding system. We are preparing a proposal for such an effort to be presented later this Spring (75). Consequently, it is not appropriate to describe any plans here.

However, it is appropriate in the context of this proposal to note the issue of scientific style involved. The vision work has matured over the last several years (about 3) under a very low head of steam. Now that we have built up a base of expertise, and have demonstrated it by beginning to produce new research results (e.g., the above segmentation results), it is time to shift to the project-like mode of operation. We are now ready to specify distinct scientific goals to be obtained by creating systems with specific capabilities, within a reasonable delineated time scale.

2.7 Resources

The AI research involves four faculty members at differing levels of effort. Their work is supported by 2.5 research associates, one member of our programming staff and six graduate students. The work in vision requires the equivalent of an additional 2 people from the programming and engineering staffs plus 2 more graduate students.

Though all of AI is only about 16% of the personnel budget, the very nature of their large (60-100K) and computationally complex programs consume 37% of the PDP10 facility. Whereas the vision work will require specialized facilities (separately requested) most of the

other AI computing is at present adequately served by the current PDP10 facility. Some of this work would profit from a large virtual memory system and may require parallel systems for efficiency.

3. SPEECH UNDERSTANDING SYSTEMS (SUS)

3.1 Introduction

A Speech Understanding System (SUS) is a computer system that receives continuous speech and determines the meaning of the utterance within the context of a given task environment.

Since 1971 ARPA IPTO has been engaged in a substantial research program to demonstrate that SU systems are feasible. This effort comes against a background of substantial earlier work in speech recognition which led to an assessment that the problem was extremely difficult, with a general de-emphasis of the field. Continual developments, in speech science, computer science, and especially in artificial intelligence, made the reopening of the task a good scientific gamble.

The total SUS program was targeted on a specific 5 year goal of creating a demonstration system with specified characteristics (reproduced in Figure 3-1). [Newell 73b] The program is at the two thirds point and consists of three major efforts (at BBN, SDC-SRI, and CMU) plus four supporting efforts. Each of the major efforts is attempting to create a system meeting the specifications.

Cooperation with the other SUS efforts is fostered by means of frequent workshops, technical reports, and the coordination provided by a steering committee. However the early hopes for cooperation through the exchange of programs and data (with the ARPA net acting as a catalyst) have not been fulfilled. Exceptions are the use of common data for the segmentation workshops, the exchange of SPS-41 programs with ISI, and the use of Hearsay-2 by UC-Berkeley. The causes for the lack of significant cooperation are: the differences in approach, programs that are not yet operational to permit use by each other, and the differences in programming languages and systems.

The CMU effort on speech understanding is simultaneously a part of the ARPA SUS effort and a part of our work in AI. Its organization as a project on the same level as the CMU AI project is appropriate due to its participation in the ARPA SUS effort and due to its relatively large size.

3.1.1 Goals

Perception is the creation of an intelligible representation of a task environment. It is a major function of any system capable of intelligent action and can be avoided only in highly specialized systems whose environments have been simplified to the extreme (e.g., a theorem prover whose external world gives it only already well-formed axioms and theorems).

Thus to discover the requirements of a system able to perceive is a major standing goal of AI, one that is to be approached repeatedly at successive levels of specification. As indicated earlier, it already seems clear that perception involves bringing to bear all relevant knowledge available in the system, and that there is not a "perceptual front end" that is isolated from the rest of the system. Since perception does sit logically at the skin of the system and certainly does contain specialized capabilities to transduce information from the external world, this has not been an easy lesson to learn, nor is it completely accepted.

Figure 3-1: Final Specification of SUS 5 Year Goals

The system should:

(1) accept continuous speech

(2) from many

(3) cooperative speakers of the general American dialect,

(4) in a quiet room

(5) over a good quality microphone

(6) allowing slight tuning of the system per speaker,

(7) but requiring only natural adaptation by the user,

(8) permitting a slightly selected vocabulary of 1,000 words,

(9) with a highly artificial syntax,

(10) and a task like the data management or computer status tasks (but not the computer consultant task),

(11) with a simple psychological model of the user,

(12) providing graceful interaction,

(13) tolerating less than 10% semantic error,

(14) in a few times real time on a dedicated system.

The main CMU effort is focussed on speech perception. Each perceptual arena offers its own challenges and opportunities for understanding and each is important in its own right. In particular there are corresponding problems of vision, i.e., of image understanding. [Reddy 73a]

All of the ARPA SUS efforts have in common the assumption that for a system to succeed in recognizing speech, it must incorporate in a knowledge base knowledge of many kinds and from many levels, e.g., phonetic, semantic and syntactic. Furthermore such knowledge originates in diverse, perhaps even independent, knowledge sources.

The main substantive hypothesis of the CMU SUS effort, and one shared by the other ARPA SUS efforts, is to demonstrate in detail the multiple-knowledge-source view for speech. This is to be done, and in the present art can only be done, by the construction of a system that both uses diverse sources of knowledge and succeeds in understanding speech. Such a demonstration, again by its nature, is only one sided; it can not guarantee that systems of different structure cannot understand speech.

The CMU SUS effort has an additional central hypothesis for how to organize such a multiple knowledge source perceptual system, called the Independent Cooperative Knowledge Sources. [Erman 75] Its central feature is the association of knowledge sources with distinct processes, each operating logically independent of each other (hence in parallel if you wish). Coordination is based on a common global data base that all can read and write. Hearsay-2, the main SUS system under construction, is an implementation of this philosophy and it will be illustrated in detail there.

Two important methodological hypotheses about the organization of research are embedded in the structure of the current CMU SUS program. The first is that in building large systems which are at the research stage, one must carry along multiple systems as long as possible until the uncertainties clear up. This hypothesis guides the total ARPA SUS effort, where there are three parallel endeavors. It guides our own as well. We have four distinct variants under development. As we will show, there are reasons for each of the four systems, and there is strong interaction between them in terms of the total development of the research.

The second methodological hypothesis is that one should always carry along a benchmark program, which has a simple uniform structure and which can provide revealing comparisons against the main systems. This is particularly true where the main systems themselves are highly complex and therefore difficult to analyze. The DRAGON system, one of our four SUSs, plays this role for us.

3.1.2 Plans

The plans for the SUS effort are dominated by the overall schedule for the ARPA SUS program. The critical dates are:

Nov75: A "dress rehersal" demonstration of the SUSs for each of the major contractors. The actual dates have not been set to within a few months.

Nov76: The official end of the 5 year program: A final demonstration of a system meeting the specifications.

A follow-on plan for SUS research is being developed by the SU Research Group

Steering Committee. Our own plans for additional SUS work beyond Nov76 must await the emergence of that plan.

3.2 Hearsay-1.X

Hearsay-1 was the first operational SUS developed by CMU and also the first one within the ARPA SUS program. [Reddy 73b, 74, Erman 74] It was initially demonstrated in June72, and again at the Nov73 mid-course evaluation. It is an early instantiation of the system organization hypothesis of Independent Cooperating Knowledge Sources. Hearsay-1 has three components: Acoustics, Syntax and Semantics. It runs currently on five separate tasks: Chess, Desk calculator, Medical diagnosis, News retrieval and Formant Tracking. Only Chess has a full semantic component, and the system is now considered to be a so-called "basic" SUS consisting only of acoustic and syntactic sources of knowledge. (Several SUSs are beginning to emerge in the literature, almost all with only these two types of knowledge.)

3.2.1 Plans

We have two goals for Hearsay-1. The first is for performance analysis. As a system that runs routinely and reliably, it can be studied and parametrically varied with comparative ease. The stucture is accessible enough so that substantial improvements can be made, for example in aspects of its search strategy, and can be evaluated. The sequence of systems so-defined are known as Hearsay-1.1, 1.2, etc., or generically, Hearsay-1.X. This part of the work in going on in conjunction with the work on DRAGON, described below, so that we are getting comparative evaluations of two systems.

An important aspect of performance analysis on Hearsay-1.X is as a dry run, so that the analysis of Hearsay-2 can proceed smoothly and rapidly when Hearsay-2 matures enough to make performance analysis worthwhile.

The second goal is as a back-up system. Under the influence of the performance analysis it keeps evolving to a system with higher capabilities. If Hearsay-2 runs into insuperable difficulties, then Hearsay-1.X offers an alternative route, one that is substantially less desirable because of its poorer organizational structure, but one that could be used.

3.3 DRAGON

DRAGON [J. K. Baker 74, J. K. Baker 75] is a SUS that represents the speech utterance as the product of a probabilistic Markov process. That is, the speaker is represented as being at any moment in one of a set of possible states (each corresponding to the intention to convey a certain meaning, to form a given syntactic phrase, to emit a given word of the lexicon, to emit a given phone, to obey a given phonological rule, etc.). The speaker moves between states according to a set of probabilities, thus tracing out a particular sequence. DRAGON finds the Maximum Likelihood solution to what sequence of states might have produced a received utterance. The meaning of the utterance can be directly understood knowing that sequence.

DRAGON represents a uniform computational approach to speech understanding. It stands in sharp contrast to the attempt to encode and deal with the many complex sources of knowledge, each in its own terms, which constitutes the essence of the main thrust of our work on intelligent systems. It does encode many sources of knowledge (it is not a pure

acoustic-phonenic recognizer), but they must all be expressed as Markovian networks. However, an important feature of the scheme is that the total system can be represented as a hierarchy of networks (corresponding to the usual levels of the speech hierarchy) and the final uniform network can be produced automatically.

3.3.1 Goals

There are two goals for DRAGON. One is to produce a functioning SUS to meet the five year specifications. DRAGON represents a genuine alternative to the present work, both here and at the other SUS sites, and it seems important to keep exploring this approach. On the basis of present evidence (see below) this is an extremely promising path and we currently expect the system to make it.

The second goal is for DRAGON to be a benchmark against which our other systems can be compared. As we asserted earlier, we believe such computationally simple schemes are important in order to understand what is being gained by the more complex approaches. To this end, the work on performance analysis of SUS systems is being conducted jointly with DRAGON and Hearsay-1.X.

3.3.2 Current State and Recent Accomplishments

DRAGON-1 exists in SAIL on the PDP10 and runs routinely and reliably. It has two levels of net: Syntax (word level) and Acoustic (phone-level), and works directly from the basic 10 ms parametrization of the speech signal (into six bands of amplitude and zero-crossings).

DRAGON-1's performance is shown in Figure 3-2 over the same five tasks as Hearsay-1.X, the appropriate comparison being with the acoustics-syntax versions. It runs somewhat better but is substantially more expensive. This expense arises from the uniform nature of the computational scheme, which examines all possible paths through the Markov network. (However, the process does not explode combinatorially; it is only bilinear with the number of states and the number of time intervals.)

DRAGON became operational in Apr74 and the performance measurements were taken during the fall of 74.

3.3.3 Plans

Improvement of DRAGON to meet the goals of the 5 year program involve improvements both in speed and accuracy, with concurrent escalation of the tasks on which the system is measured. The following steps are being taken:

A sequence of recodings are taking place, whose ultimate aim is to produce an optimized version on the PDP11 ELF system, which has part of the system running on the SPS-41, a specialized 10 Mips signal processor. DRAGON-1.1, an optimized version in SAIL; DRAGON-1.2, a recoding of 1.1 in BLISS10; and DRAGON-1.3, a version of 1.2 in BLISS11 for the PDP11, are all underway. DRAGON-1.3S, the SPS41 version, is being planned. The expected date of completion is Dec75. We may initiate a version, DRAGON-1.4, for C.mmp to compare with 1.3S, if comparison seems warranted.

We will explore variations of the algorithm and encodings of knowledge that offer major improvements in efficiency. One path leads to a class of systems, called the HARPY series,

Figure 3-2: Dragon / Hearsay-1 Comparison

	Size of Vocabulary	Accuracy (word level) % Correct		Time seconds of CPU per second of speech	
		Hearsay-1	Dragon	Hearsay-1	Dragon
Chess	24	69	94	14	48
Medical	66	49	88	9	67
Desk Calculator	37	53	63	16	83
News Retrieval	28	74	84	11	55
Formant Tracking	194	33	84	44	174

which do not explore all paths through the state network. These are actually a cross between Hearsay-1 and DRAGON, and represent explorations at the search strategy level. A second path leads to using a DRAGON system in conjunction with components of a regular SUS, to have it do only part of the total job. DRAGON-1.5 takes the segment level as input, using the Hearsay-2 segmenter; the current version in essence does it own segmenting by taking the acoustic parameters as input directly. DRAGON-1.6 will use a word level net only, taking the phone labeling from Hearsay-2. We expect all these variations to be explored prior to Spring76. They not only provide the basis for selecting an appropriate DRAGON variant to continue, but they will shed considerable light on the structure and performance of the other systems.

We expect to go to a 1200 vocabulary system in time for the Nov75 demostration. The improvements in speed expected from the above explorations will permit this. However, we do not expect the performance to be very good at that time.

We expect to continue the development of the Nov75 system right through Nov76, making it an entry for the final evaluation. We do believe we will be able to make rather definite statements about the trade-offs to be made with this style of system versus the 5-year specifications. We also will be able to put some basements on the performance in the other tasks.

3.4 Hearsay-2

Hearsay-2 is the main SUS system being developed for the SUS effort. [Erman 73, Lesser 74a] It is our main embodiment of both the general hypothesis about the requirement for multiple knowledge sources in perception (here, speech), and of the particular organizational scheme of independent cooperating processes associated with each knowledge source.

Hearsay-2 has a single common data structure (called the Blackboard), which contains a set of hypotheses interconnected by relations of which hypothesis supports another and which hypotheses are alternatives. This leads to a lattice data structure which directly reflects the time-ordered and multi-leveled character of speech. A fragment is shown in Figure 3-3. In terms of the representational conventions, all levels -- semantic, syntactic, phonological, etc -- are handled identically.

> Each knowledge source is represented by an independent process (as that term is used in multiprogramming or multiprocessing).
>
> Each process is capable of the same set of basic actions on the Blackboard (taken of course by virtue of its own special knowledge): to detect when it is relevant (i.e., has something to contribute); to create hypotheses to add to the Blackboard (and delete those there); to evaluate hypotheses found in the Blackboard (leaving evaluations for other processes to read).
>
> Each process can look at as little or as much of the Blackboard as it wishes. From a programming view, the knowledge processes also satisfy common conventions about their creation, modification, debugging and user interaction. The number of knowledge sources is expected to vary between 10 to 100, depending on how narrowly or broadly they are decomposed. The initial set is shown in Figure 3-4.

To the unwary, the general structure of Hearsay-2 may seem so general as to be unexceptional. But just out of public view is an important problem, called the Subroutine Interaction Problem, which states that (given the current art) there is no way to add substantial new knowledge to an existing system without having it interact with all the existing knowledge in the system. Thus the entire system must be modified to make each incremental addition effective. Systems which do not formally suffer from this (e.g, theorem provers where knowledge can be added simply by adding propositions), show the effect in the increased combinatorial explosion of their search, and have not proved effective. Thus, the Hearsay-2 scheme is an attempt to provide a system that can beat the subroutine interaction problem, by having each process communicate with the Blackboard and make its contribution independently of the others. Whether this organizational hypothesis is effective will be determined by whether Hearsay-2 operates effectively (and for what reasons) and, most important, whether new sources of knowledge are in fact continually added to the system during its growth period, without continual readjustment of the other sources.

The subroutine interaction problem affects the development of a system, not the final structure. It reveals that the most important aspects of large systems may well be their development and modification. The Hearsay-2 organization addresses some of the other properties of development as well: rapid configuration of new systems with arbitrary sets of knowledge sources and extensive interactive experimentation. Hearsay-2 is programmed in SAIL on the PDP10, and uses that language's associative processing (LEAP) and its facilities for asynchronous processing.

3.4.1 Goals

We have already stated the two main goals for the Hearsay-2 system: To meet the SUS specifications of Figure 3-1 by Nov76 and to prove out the system's philosophy, which, if successful, will make the Hearsay-2 organization an important contender for other intensive knowledge based intelligent systems.

Figure 3-3: Example Hearsay-2 Lattice Structure (Fragment in the Blackboard)

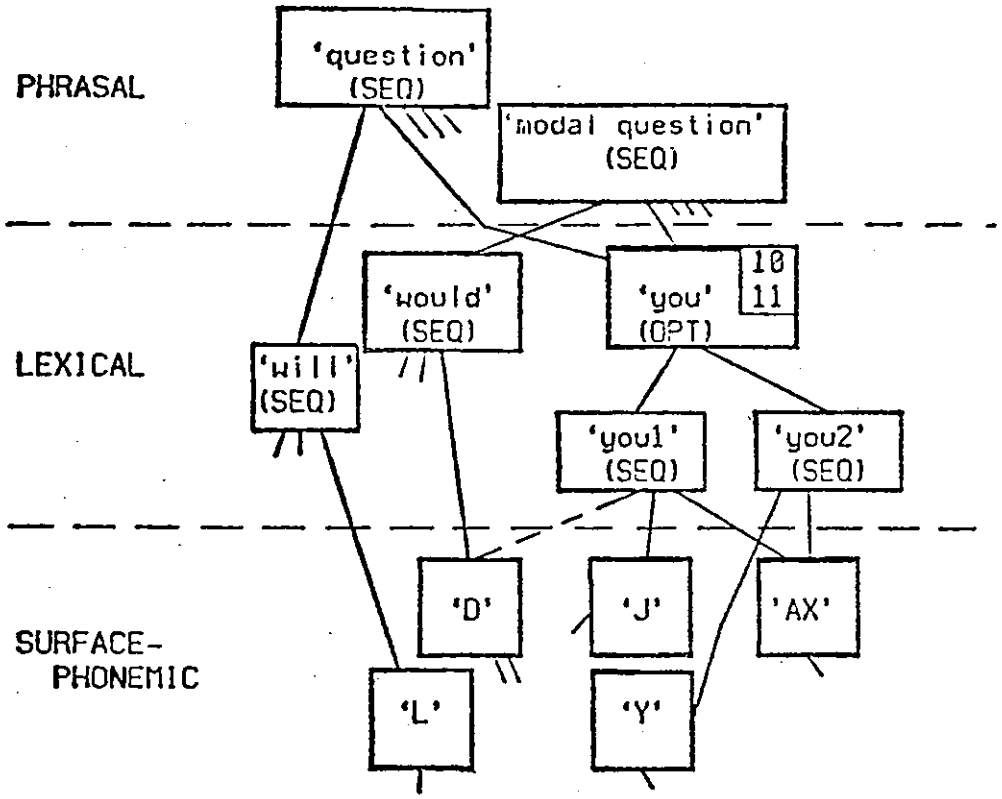
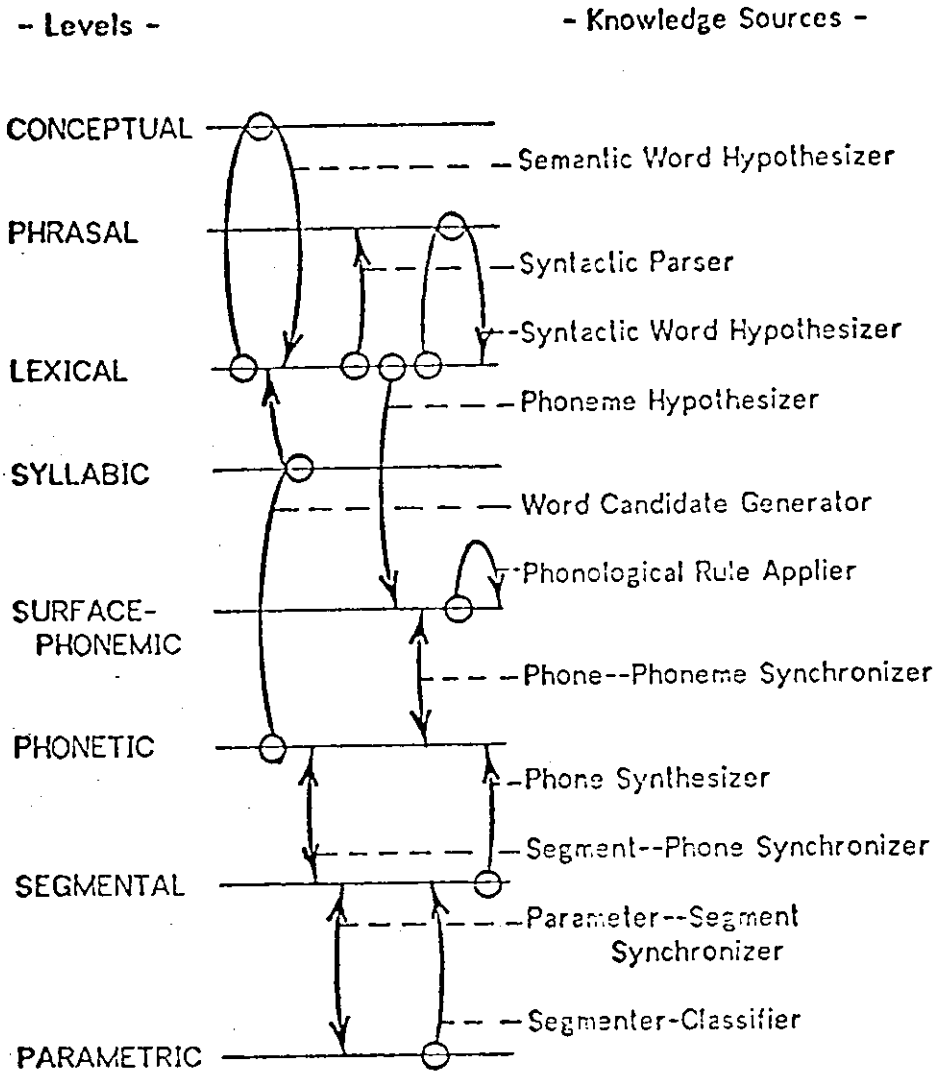


Figure 3-4: A Set of Knowledge Sources for Hearsay-2



3.4.2 Current State and Recent Results

Hearsay-2 currently exists with the knowledge sources shown in Figure 3-4. It recognized its first sentence in Nov74. Currently it is extremely expensive in both space and time, taking about 200 times real time and 180K primary memory.

3.4.3 Plans

All the major systems aspects must undergo extensive work throughout the period from now until Nov76: developing additional knowledge sources, reducing the space required, increasing the speed, and generally tuning the system. The following are our current expectations:

The system will be running on a 1200 word vocabulary by the Nov75 rehearsal, though not well (i.e., we have no performance expectations).

Performance on a 200 word vocabulary will be 90% at the word level and 70% at the sentence level on the AP News Retrieval task.

We expect the number of Knowledge Sources to be about 15 for Nov75 and 30 for Nov76. The semantic and task knowledge sources are currently the most important and they are both in progress. Semantics will look some like Shank's Conceptual Dependency model.

The space is expected to grow to about 300-500K words even with efforts at code compaction. This will require a managed segmentation system (our KA10 does not run under a paging system).

We currently expect to get the time down substantially and believe we can keep it to about 100 times real time for the Nov75 and Nov76 systems.

A critical area, especially for the Hearsay-2 structure, is how to focus attention to avoid the combinatorial explosion (which will show up rapidly in both space and time if great care is not taken). Intensive investigation of this problem is going on, though no precise expectations can be stated.

We will initiate performance analysis of Hearsay-2 by Jul75 at about the level at that for Hearsay-1.X and DRAGON. The system will not be sufficiently stable (in the performance of its knowledge sources) before then to make evaluation worthwhile. The continuous work on performance evaluation of DRAGON and Hearsay-1.X prior to that time should provide us with an appropriate methodology for dealing with Hearsay-2.

3.5 Hearsay-2 on C.mmp

The structure of Hearsay-2, in the independence and multiplicity of its knowledge sources, lends itself to realization on multiprocessor organizations. [Lesser 74b, 75] [Fennell 75a,b] However, so little experience exists with multiprocessor realizations of complex programs, that much remains uncertain. Indeed, we can expect that a multiprocessor version of Hearsay-2 will reveal much about real-time cooperation of processes in an intelligence-demanding environment. Communication with the large global data base, for instance, may impose severe restrictions on the operation of such a system.

From the viewpoint of applications, multiprocessors, especially those composed of small processors (minis or micros), appear very attractive in terms of processing cost and the mass production of specialized systems (such as SUSs need to be). Actually, the power argument is relevant in the present situation, since our multiprocessor, C.mmp, provides more Mips, more primary memory, and more flexible memory management than our KA10s (see the C.mmp section for an analysis of C.mmp and of the SUS). Thus, a successful implementation would permit much faster evolution of Hearsay. (This again harks back to the proposition that the most important aspects of systems may be their developmental characteristics and not their final configurations.)

C.mmp and, in general multiprocessing implementations are highly experimental. Consequently, we view this as a high risk, high payoff implementation. Under no circumstances can we permit this version to become the critical path to the Nov76 deadline, at least without major uncertainties being resolved.

The implementation of Hearsay-2 on C.mmp will be functionally equivalent at the level of the global data base and its operations, and the knowledge sources. We intend to provide a form of transliteration of the knowledge sources, so that no new substantive programming for them will occur in the new system.

The system will run under HYDRA, the operating system for C.mmp, and will be implemented in an interactive implementation system, L*. (See the section on C.mmp for details on both of these systems.)

3.5.1 Current State and Recent Results

The section on C.mmp gives details on the status of the hardware and operating system software for C.mmp.

L*C.(B) [Newell 71] is operational on C.mmp, having been brought up within the last six months of 74. The initial design of Hearsay in L* exists and has been coded. The primary problems lie in the small address space of the processors (32K 16 bit words), which is forcing great attention to the problem of memory management. This problem has been anticipated to be a central one, and it is inherent in the use of a multi-miniprocessor of the C.mmp type (one that does not use hardware devices to create a large homogeneous virtual address space).

3.5.2 Plans

We expect the initial version of the basic Hearsay (without knowledge sources) to be running sufficiently to test the feasibility of the design by Apr75. Critical decisions to be made at that juncture are:

> Is implementation in L* feasible? Many changes are being made relative to the operational style of Hearsay-2 in SAIL. There are basic issues of the efficiency and the addressing problems, which reflect C.mmp basic structure as much as the L* implementation system.
>
> Is implementation under HYDRA feasible? This will be a time and overhead issue, since without doubt HYDRA is logically adequate.
>
> Can Knowledge Source and focus of attention mechanisms be feasibly realized in L* and under HYDRA? Can the implementation capture enough of

the Mips to be more powerful than the KA10? Are the facilities available, e.g., a sufficiently sophisticated file system, flexible scheduling?

Assuming positive answers to these questions we will set perfomance goals for the Nov75 rehearsal.

3.6 The Four Systems in Perspective

We have now described the current four SUSs: Hearsay-1.X, DRAGON, Hearsay-2 and Hearsay-2 on C.mmp. In the introduction we gave some research-strategy reasons for wanting to work with multiple systems: That one needs alternatives where there is uncertainty; and that one needs a pacing horse, i.e., a benchmark program (DRAGON, in this case). These two principles justify in a general way the course we are taking. Further, most of the present effort would have been required even if we were building a single SUS. The fact that we can study all their alterntives by devoting less than 20% of the resources (available for SUS research) makes it an attractive option.

These principles do not make clear the actual choice of systems (except possibly DRAGON). The answers are implicit in our discussion of the systems separately; let us make them explicit.

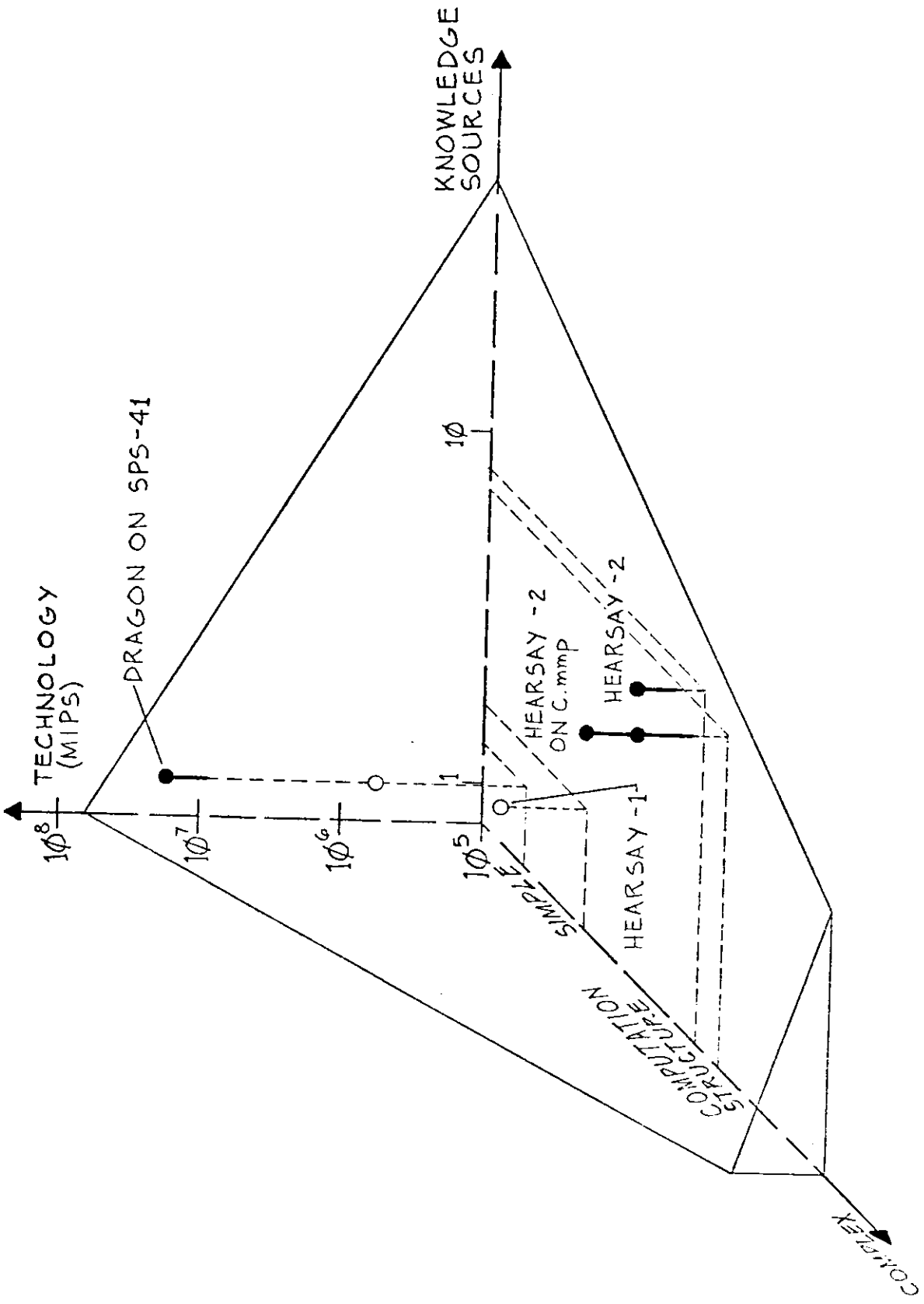
The four systems represent a concern along three dimensions of system structure: (1) The simplicity or complexity of the computational structure; (2) the amount of knowledge used by a system; and (3) the amount of computational power.

One can think of our four systems as occupying regions in a design space, as shown in Figure 3-5. Hearsay-1, Hearsay-2 and Hearsay-2 on C.mmp all are options for a complex computational structure; they represent our bet that such a structure is required. DRAGON, on the other hand, has a simple structure. This is why it is our pacing horse system.

Hearsay-1 is located at the low end of the amount of knowledge, Hearsay-2 is at the high end. One might measure this in number of knowledge sources, 3 for Hearsay-1, 9 (and rising) for Hearsay-2. Here there is no question of preferences -- one moves from Hearsay-1 to Hearsay-2 as the research develops. From this analysis, the only reason for keeping Hearsay-1 active is as a back-up in case the organization of Hearsay-2 becomes too complex. Note that Hearsay-1 is much simpler in computational structure than Hearsay-2, even though both of them are committed to the "complex" view. In terms of this dimension, DRAGON is about at the same place on the knowledge scale as Hearsay-1. Any attempt to press it harder will probably require moving it up in terms of the knowledge it uses.

The final dimension is that of computational power required. As the adequacy of the system increases, it requires more Mips and more memory. This is just as true of systems representing the "simple" view as of systems representing the "complex" view. (Each however deploys its computational resoures differently.) Therefore it is incumbent to address where the power is going to come from (we should look both at Mips and Mips/$). Technology makes clear at the moment that power comes from mini- and micro-processors, combined in specialized architectures and taken several at a time. Thus, Hearsay-2 on C.mmp is an attempt to begin this exploration into more efficient computational processes. In these terms we have in fact plotted a fifth system, which is DRAGON on the SPS-41, which is an analog of doing with DRAGON what the use of C.mmp does for Hearsay-2.

Figure 3-5: SUS System Comparison



The four systems we have picked are not the only points in this space, and indeed our notions of what the right mix of systems is will change as we get more results of our analyses.

The plane through the 3-D space in Figure 3-5 indicates an idealized notion of a plane of constant performance. It is an attempt to illustrate that similar performance can be achieved by systems with widely varying characteristics. Combinational explosion can be contained by using more knowledge or more processing power. Simple program organization may have to perform many more unnecessary tests than complex ones but can usually do so with substantially lower computational overhead. Thus what counts for a system is not its absolute coordinates, but where it resides relative to the plane of best performance attainable in a given epoch.

3.7 Support Common to all the SU Systems

Two activities are used in common by all the four systems. One of them, the Data Base, is a required adjunct to any SUS effort. The other, the parameter-independent segmentation and labeling, represents an instance of a general philosophy of not committing ourselves to specific forms of sources of knowledge (here acoustic-phonetic), but wherever possible considering a range of alternatives, against which we can do some performance analysis within the context of actual systems.

3.7.1 Data Base

Our goal is to produce a library of segmented, labeled, and cross referenced utterances which may be used to evaluate the performance of an SUS as well as for studies to determine the nature of knowledge sources. These need to be from the main task we are working with (the AP News Retrieval task) and need to be graded in the size of vocabulary used from small sizes all the way up to the 1200 vocabulary, which is our current maximum. [Shockey 74]

We expect this to be done by May75.

3.7.2 Parameter-independent Machine Segmentation and Labeling

Our goal is to use a single front-end system for all our systems. Furthermore, we wish to be able to evaluate different parameterizations -- LPC spectra, PARCOR, 1/3-octave filters -- and to use whichever seem appropriate with any of the systems.

Our approach has been to use training data (carefully segmented and labeled) and have automatic determination of label targets and segmenting thresholds. The scheme is speaker and microphone specific. Our intent is for the system to adapt to the speaker and room conditions on the fly in any operational system. This constitutes mild training of the system to the speaker, since, though there will not be any specific extensive training sessions, the system must acquire some utterances for which it knows the correct interpretation in order to learn.

The current system has undergone several iterations from its first version (Apr74) [Goldberg 74]. The segmentation scheme is in routine operation, we still expect more improvements in labeling, and we have preliminary evaluations of the various parameterizations.

By May75 we expect to be placing the correct label in the first three choices 95% of

the time, to have less than 1% missing segments from phonetic segmentation, and to less than 10% extra segments from acoustic segmentation. We feel these accuracies are what is required to produce the results predicted of the Nov75 systems.

3.8 Resources

As the current speech understanding program approaches its Nov76 performance deadlines, the manpower required to reach them has increased. All of the efforts with the exception of Hearsay-1.X are at the implementation level and so require more full and part-time programming support. We are clearly implementation, not idea, limited.

The entire area directly consumes 29% of the personnel budget exclusive of facility and administrative costs. The overall direction is assumed by one faculty member. The Hearsay-1.X system is stable and so has only one graduate student devoted to its operation and performance measurement. The DRAGON (ELF-SPS-41) effort requires one staff programmer and some graduate student support. Hearsay-2 efforts occupy 6.5 research associates whose contributions cross both the PDP10 and C.mmp versions, with emphasis on the former. Five members of the programming staff are also allocated to Hearsay-2, three on the PDP10 version and two on the C.mmp-L* and speech system. Exclusive of engineering support, the relative SUS manpower allocation to each of the four systems is as follows: Hearsay-1 (3%), DRAGON (12%), Hearsay-2-PDP10 (47%), Hearsay-2-C.mmp (21%), general (17%). The overall program is further supported by three engineering staff members and seven graduate students. A large number of part-timers provide basic low level support for this project so a specific allocation of funds for this group is included in this year's budget.

Facility utilization falls into the two categories of the general facility (PDP10) and the use of specialized processors, C.mmp being included in that category for this discussion. The present effort consumes 46% of the PDP10 facility, virtually one of the two processors. There is little improvement that can be made to this particular configuration. Indeed, the efforts do require a more powerful processor, of the order of a KL10 with 512K memory and paging. This would probably be worth about a factor of 10 when running the ultimate system version with segmentation. Budget constraints did not permit requesting of such an upgrade.

The DRAGON (ELF-SPS-41) system is stable on its dedicated PDP11, and could only profit from a higher powered specialized processor available by early 76 to yield improved performance. The Hearsay-2 effort on C.mmp requires completion of C.mmp, which is dealt with elsewhere in the C.mmp and Budget sections of this proposal.

4. C.MMP: MULTI-MINIPROCESSOR COMPUTER SYSTEM

4.1 Introduction

C.mmp is a multiprocessor system consisting of up to 16 PDP11s operating through a 16*16 crossbar switch into a primary memory of up to 16 memory modules. Each processor can lay its 32K 16-bit word address space anywhere within the larger 2^{20} word physical address space in 4K-word pages. Figure 4-1 shows the structure of the system. [Wulf 71b]

C.mmp has a flexible capability-based operating system, called HYDRA, which permits the operation of many distinct specialized suboperating systems simultaneously. Under HYDRA, C.mmp operates in anarchical mode, which is to say that no physical processors have a distinguished role (e.g., as master or slave).

4.1.1 Goals

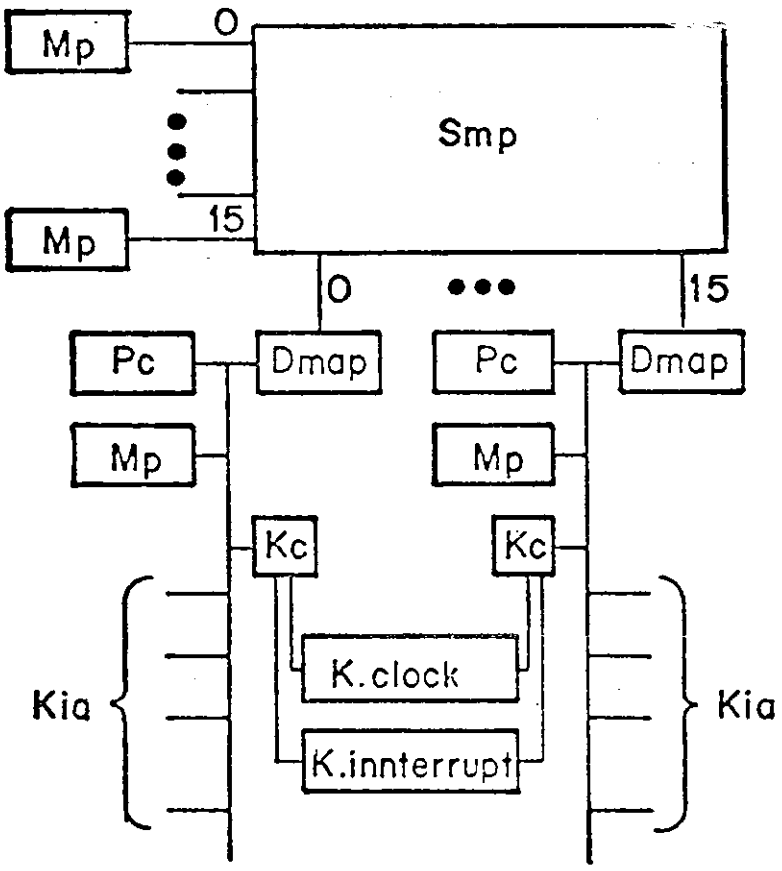
The fundamental reason for a research group to construct a multiprocessor is the general computer science goal of advancing our knowledge of multiprocessors.

Why multiprocessors should be studied -- out of all the possible regions of the computer design space -- lies with some fundamental cost and processing considerations. Multiprocessors do not offer any end-user or programming advantage over a uniprocessor of equivalent parameters: of equal Mips, processor-to-primary-memory bandwidth, I/O bandwidth, etc. Indeed there must be decrease in flexibility and increase in programming complexity. However, for a given state of the art of uniprocessor design, there is no way to obtain the postulated equivalence on performance parameters for equal cost. This arises for two distinct reasons. First, for large N, a processor N times as fast as a currently existing fast uniprocessor exceeds the art. Second, the cost of relatively small processors, if sufficiently mass produced, becomes much less than N times the cost of feasible but low volume processors of N times the performance. Thus cost/benefit drives out uniprocessors in favor of multiprocessors. An entirely independent advantage of multiprocessors is the potential for high reliability due to the multiplicity of components.

The computing world has known this analysis for years, though the number of multiprocessors with more than two processors can be counted on the fingers of one hand. (We do not consider here array or vector processors which have single control stream with a multiple data stream, e.g., ILLIAC-4, CDC Star, etc.) The amount of generally understood scientific knowledge that has emanated from this handful is hardly discernible. It is beyond the bounds of a proposal to develop the fundamental reasons why multiprocessors have not been explored. The reasons are complex and relate to alternative options, to reliability considerations, and to the types of processor systems that funding sources have wanted to see developed. It remains true that we are now moving through a region of technology space where the cost/benefit arguments for multiprocessors become ever more compelling, but without benefit of any of the years of exploration in how to live effectively on such systems.

C.mmp is thus our attempt to generate some of that experience. The two most pressing problems that must be addressed in multiprocessor research are (1) the realization of the initial favorable cost/benefit ratio; and (2) the realization of effective use without consumption of all the cost/performance advantage to obtain it.

Figure 4-1: C.mmp Architecture



The basic cost/performance situation is fundamentally determined by the hardware structure. Figure 4-2 gives some data for C.mmp, using the KL10 processor as the comparison machine. The figures are based on actual measurements of code densities and processing rates for equivalent programs on C.mmp(11/20) and the KA10, with extrapolation to C.mmp(11/40) and to the KL10 as 4*KA10. Primary memory and processor costs are of the same order (and generally more important than switch and I/O channel costs), so that comparisons depend sensitivity on the assumptions about the amount of primary memory required per processing rate (Mp-bits per instructions/sec) and about the cost of memory. The diagram shows Instructions/sec per dollar system cost for C.mmp and KL10 systems, as the cost of memory varies (along the horizontal axis) and as the ratio of Mp-bits per instruction/sec varies (the parallel family of curves). To the right, as memory dominates in cost, all systems begin to look alike, purely as a function of amount of memory. To the left, as memory becomes cheap relative to processing, all variations of amount of memory per instruction/sec become irrelevant and C.mmp becomes equal to about 4 KL10s in efficiency. (In absolute power, C.mmp is 5.95 mips to the KL10 at 1.37 mips, a ratio of 4.35.) Several distinct points are laid out on this curve corresponding to the current KL10 with 256 K words (1 megabyte) of primary memory at current DEC memory prices and at prices equivalent to those paid for C.mmp memory; to the projected 16 11/40 C.mmp with 2 megabytes and 4 megabytes of primary memory.

The cost/performance figures of 4-2 are determined by the hardware structure, and are essentially fixed at design time. The other fundamental question -- using the architecture without using up all the advantage -- is only determined at the end of a more tortuous path. It depends on carrying through the design to a functioning computer system of adequate reliability, providing an operating system, providing software facilities, and bringing the system to full use on a range of applications, where final usefulness can be demonstrated and final net advantage can be measured. This list essentially constitutes the intermediate goals of the C.mmp effort. And in fact the system was realized in conservative technology (e.g., neither the processors nor memory modules employ currently available LSI integrated circuit technology) precisely because all these other steps need to be attained without undue delay, if the total scientific goals are to be realized.

4.1.2 Plans

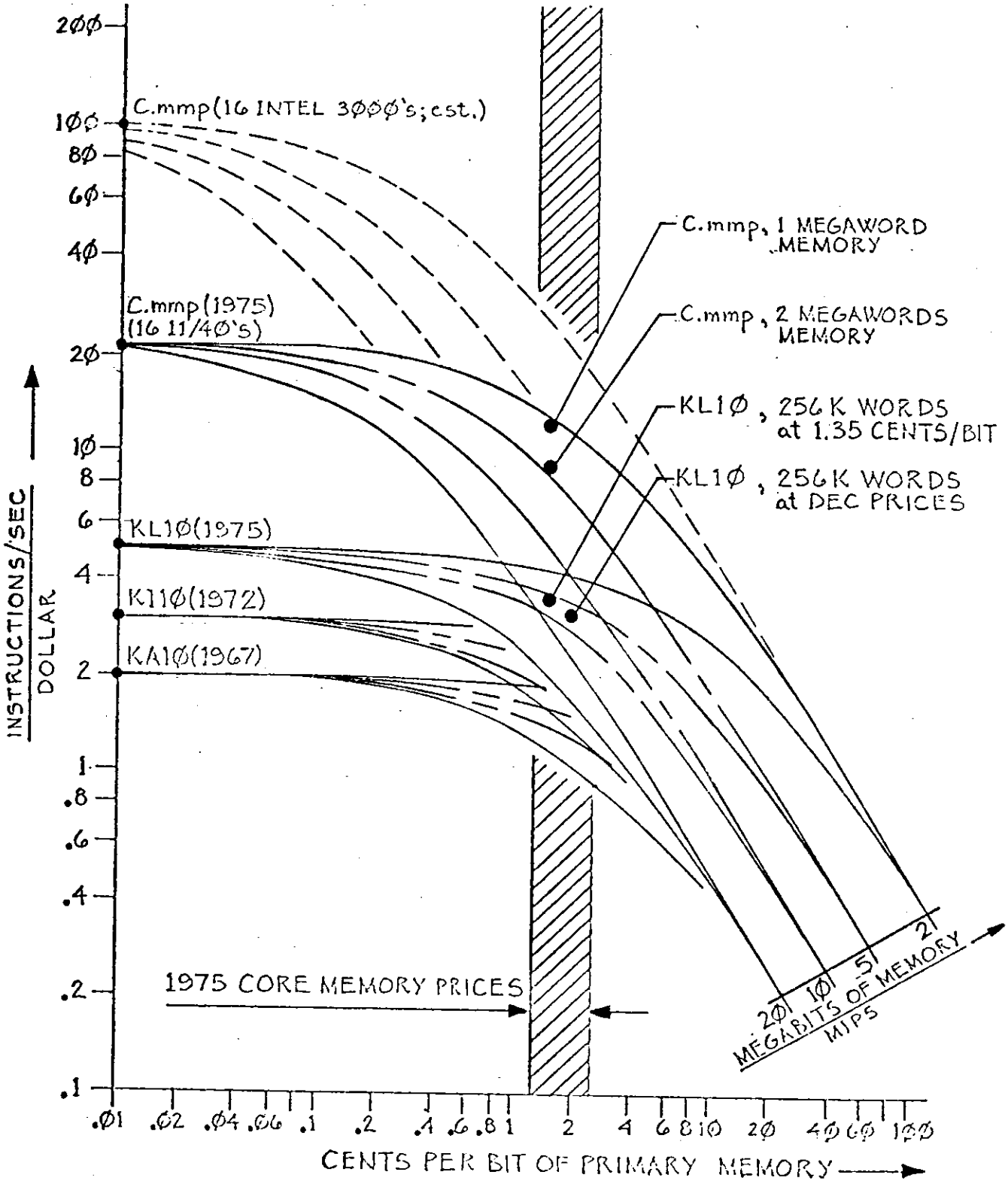
The general plans for C.mmp are dominated by the point in the life cycle at which we now find ourselves. With the system just coming into operational state, all of our concerns are to make it a system on which a large amount of experience can be gained and to shape that experience so that it illuminates the basic issues of multiprocessor structure. We are at that critical juncture where the most important question is the short-term one of how long the transient will be until effective operation.

The rest of this section takes up each of the subcomponents of the total effort in turn: The hardware system; the operating system; the software facilities; and application programs and performance analyses.

4.2 Hardware System

The basic configuration is shown in Figure 4-1. A few additional facts about the system are given in Figure 4-3. The system was designed for the PDP11 processor. Each processor model requires separately designed relocation registers and modifications (not extensive). The original processor was the 11/20. We have since decided that 11/40, which

Figure 4-2: Cost Performance of C.mmp



was announced after the original design was frozen, is a more suitable processor. To capitalize on the 11/40's speed relative to the trip through the switch, it is necessary to add a cache to the relocation registers. (Little extra gain is made by using 11/45s, since the extra high-speed bus cannot be exploited.) As we describe below, the 11/40s we will utilize will have dynamically writeable microcode.

Figure 4-3: Added Facts about C.mmp

(1) Up to 16 PDP11/20 or PDP11/40 processors

(2) PDP11/40's may have writeable microstore

(3) Up to 16 ports each containing up to 10^6 18 bit words

(4) 16x16 crossbar switch allowing up to 16 simultaneous processor-memory connections

(5) 200 nanosecs (roundtrip) delay for going through the switch

(6) Maximum switch bandwidth: $5*10^8$ bits/sec

(7) Each peripheral device associated with a single processor

(8) Non-demand paging to fixed head, zero latency disks; capacity: 512k words (= 128 pages) per disk; transfer rate: 1 page per 16 ms

(9) Other peripherals: standard PDP11 devices

(10) Communication links to front end terminal processor and PDP10A

(11) Imp host (interface under construction)

(12) Current state: 5 PDP11/20 processors functioning; 700k words memory availabe in 12 ports; 4 IMS paging disks; 2 RP03 disks (40 megawords)

The 11/40 is realized as a horizontally microprogrammed processor with 256 56-bit words. Another 1024 words of writeable microcode can be added at any time, without modification to the 11/40. We have designed and augmented the 11/40 system to convert what was originally a rather special microprocessor into a generally useful dynamically microcodable processor.* This has involved adding a general mask-shift unit, a micro-subroutine facility with stack, an emit field and extended branching capability. With these additions there is enough micromemory to do extended arithmetic (both 32 and 64 bit data types), to incorporate many HYDRA kernel functions (with reliability checking), and to form L* or Lisp-specializations.

The design of the IMS fixed head swapping disks should be noted. We achieve zero latency by mapping the C.mmp page (4K words) exactly onto a single track of the disk. The

*DEC supplied the 11/40 and NSF supplied the writeable microstore.

disk controller has been modified to permit the data transfer to begin immediately, whatever the initial position of the platter.

4.2.1 Current State and Recent Results

C.mmp has been available to sympathetic users with the 16*16 switch since Nov74. It currently has 5 11/20s, 700K primary memory through 12 ports (512K through 11 ports routinely functional), and 4 IMS paging disks. The system is connected to terminals via the Communcations Front-end (see Facilities section), in addition goes directly to the PDP10A via a 4800/300 baud link.

The design for the 11/40 relocation registers, cache and processor modifications is complete and the initial version is being implemented. There are 5 11/40s in-house destined for C.mmp. (No more 11/20s will be put on C.mmp.) These additions and modifications are independent of whether the 11/40 is equipped wih writeable microstore.

An initial version of the programmable microcoded 11/40 has been operational since Oct74. We have done a study of its use for the XGP Xerographic Printer. We are currently studying the amounts of speed up available for C.mmp generally, i.e., for HYDRA and basic programming processes. Preliminary results indicate a factor of 4-5 is available in some sections of HYDRA by rewriting these functions directly in microcode. This leads to an estimate of as much as a factor of two in overall performance of the system under HYDRA. These results are very recent (Dec74) and will be checked. They do form the basis of our decision (see Plans below) to go ahead with the proposal to put writeable microstores on all the 11/40s on C.mmp.

[The performance analysis of Figure 4-2 does not take into account that the 11/40s would be microcoded. This would not show up in the raw instructions/sec (which must actually decrease with the microcoded 11/40 if it spends time in the specially microcoded functions). It will show up in the equivalence ratios of how much a KL10 instruction is worth, for a given computation, vs a simple 11 instruction (they happen to be nearly equivalent) vs a microcoded 11/40 instruction (which will increase by some factor large enough to override the lowered instruction rate and yield a net increase in effectiveness). This will in fact produce a more favorable picture for C.mmp, but the estimates of the effectiveness of the microcoded 11/40 are still too unreliable to justify plotting it.]

4.2.2 Plans

By Jul75 we expect to have the total system of components that are in-house in the system: 4 11/20s (The 5th 11/20 now on C.mmp will be swapped for an 11/40) plus 5 11/40s, 1 of which will have a writeable microstore, 700K words of primary memory, 7 IMS paging disks, the IMP connections and a 9600 baud connection to the Front-end.

We plan a complete system that consists of 16 microprogrammable 11/40s with 1 Megaword primary memory, and 9 paging disks (providing a paging ratio of 4-5). The justification for going to a maximum processor system, balanced with respect to other characteristics, rests both on the need to investigate a full multiprocessor and the need for the larger amounts of processing power with respect to applications. (See subsection on Application Programs and Performance Analysis.)

To this must be added the secondary memory requirements. Our current view makes C.mmp heavily dependent on the PDP10s for secondary storage, i.e., for on-line disk storage

(the IMS disks are strictly for paging). This requires high speed communication between C.mmp and the PDP10, to be achieved by two DA28s (which provide a total of $8*10^6$ bits/sec). However a minimal amount of secondary storage must be available on C.mmp itself. This consists of two RP04s (each 40 Megawords), each with its own controller. The capability for two separate systems is required so that the two independent partitions of C.mmp can run simultaneously. (The ability to partition the configuration has already proved to be of major value in making progress.)

The rate at which C.mmp goes to completion depends mostly on obtaining the funds for acquisition (see Resources, below). The acquistions asked for in this proposal will bring the system to 4 11/20s plus 12 microcoded 11/40s, with one million words, and with both RP04 disks. We would expect to have this installed by Dec75. The remainder will come along as fast as funds are obtained.

The creation of the microprogrammable 11/40 offers the possibility of creating highly specialized processors (as opposed to the sort of specialization which we will be putting into most of our "regular" 11/40s to adapt them to HYDRA and general use). We have been studying possibilities, such as L* processors. Plans are not firm at the moment about undertaking any particular project of this sort. (We should point out that HYDRA was designed with an environment of non-homogeneous processors in mind.)

4.3 HYDRA: The Operating System

We know little about multiprocessors. We know even less about multiprocessing operating systems, for the problems they pose can only be understood, and the proposed solutions tested and verified, in the world of routinely running multiprocessors with a community of users.

We do know that the demands of a multiprocessor on an operating system are severe. This follows from the central core functions that operating systems perform: allocation of resources and the creation of safe and sane computing environments. Diversity of resources and of concurrency have always been the instigators of our difficulties with operating systems, both qualitatively and quantitatively. Multiprocessors increase both by a large factor over uniprocessors.

Thus, an operating system for a multiprocessor must itself be a major piece of research into all of the classic operating system issues. At least it must be so if the operating system is not, all by itself, to make it exceedingly difficult to obtain flexible end-use computing without squandering all the cost/performance gain. This is no idle concern. Experience has repeatedly shown that an operating system can take a major share of the cycles of a system (the 90% overheads that formed the horror stories of software history).

We stress this point to underline that research on multiprocessors (and in particular our own research) is not just research on computer structures. Four links, at least, exist in the total research chain: the architecture + the operating system + the software facilities + the task-decomposition and application programs. To these must be added two others: reliability of total operation and analysis of performance. All six require significant research efforts and, despite any problems of keeping the whole in focus, none can be assigned to the status of simply an "auxiliary" or "supportive role". The point is appropriately made here, when describing the second link in the chain after the hardware structure. We will add short reminders as we proceed to the other links.

4.3.1 Goals

Clearly, from the above discussion, the main goal is to create an operating system adequate to the demands of a genuine multiprocessor environment. These demands seem to be (1) adequate protection and security; (2) allocation of resources, including guarantees on the amounts of resources made available in specified time periods; (3) reliability and recoverability; (4) the creation of specialized operating environments; and (5) the support of a general user environment. The first three of these are classical operating system functions; what multiprocessors add are new forms and complexities in the demands. The last two, though perhaps familiar, require some explanation.

As an oversimplification, the exploitation of a multiprocessor can take one of two forms. In one, the operating system effectively creates for the user an uniprocessor system -- except for the performance characteristics (Mips, etc) and charges, he does not know he is on a multiprocessor. In the other, the multiprocessor facilities are not masked. In fact, they are hopefully augmented to be conveniently at the command of the user who is willing to adapt the structure of his algorithms in order to exploit the full capabilities of the particular multiprocessor structure. This latter form is the one intended for C.mmp. It is the one consistent with the attempt to explore how multiprocessors are used; it is the one that maximizes the chances of solving the basic problem of not giving up all the cost/performance advantage to make a multiprocessor system habitable. There must always be a single operating system on a machine at some level, if only to give the system away to various users from time to time. This strategy forces the kernel operating system to permit such specialization by users -- and by different users in different ways. Thus, one arrives at organization that permits multiple specialized operating systems operating simultaneously.

It is a myth that specialized computing avoids providing all the facilities (and facing all the problems) of a general user facility. Specialization buys power and economics; it does not avoid providing facility. To construct and use a complex program on a computer implies the need for all the software and interactive facilities that we have come to associate with a good general purpose system. Big systems, though they may be specialized in their final effects, are general purpose in their demands. Just to make the point we have listed in Figure 4-4 a large number of system and software facilities. Having all these is what characterizes the modern general purpose computing facility. They are needed on C.mmp, if systems of any complexity are to be programmed for it, made to run, and their performance analyzed.

To be sure, the economics may indicate that the general user facilities are not terribly efficient, since the main computing is whatever the central engine does well. But they must still be there. Also, it may be possible to provide these facilities in an associated standard general purpose computer. This is the tactic of systems such as ILLIAC-4, and indeed it is the strategy adopted for C.mmp in several respects (BLISS11 code is compiled and loaded on the PDP10; most C.mmp file space will be on the PDP10 systems). But there are limits to this strategy, for it introduces rigidities which may predispose the use of the specialized system (here C.mmp) to a narrow preconceived style.

A single illustration might suffice to make the point. C.mmp, we assert, must permit general time-shared use. Many people should be able to work on it simultaneously, both on aspects of a single major application system or on many different systems. The classical ploy to avoid this is to think of C.mmp as a glorified processor (much as ILLIAC-4 is thought of) and to allot its time in exclusive intervals to a single user. Without going through a detailed calculation, the rate of utilization of C.mmp under these latter conditions would probably be less than a twentieth of what it will be under the multiple use time-sharing regime -- maybe even much less than that, considering how inefficiently a single user employs a computer.

Figure 4-4: C.mmp Software Facilities

(1) System status reports
(2) User recognition
(3) File manipulation and status reports
(4) File display (terminal and line printer)
(5) Backup of secondary storage
(6) Process creation, monitoring and debugging
(7) Resource allocation
(8) Space management
(9) Time and space accounting
(10) Editor
(11) Assembler
(12) Compilers
(13) Interpreters
(14) Process save and restart
(15) "Intelligent" command interpreter
(16) Network communication
(17) Mail to local and network users
(18) Online documentation
(19) Terminal session recording
(20) Script execution

HYDRA's design incorporates an extensible capability-based protection structure which permits implementation of non-hierarchical protection schemes which are essential to attaining any real diversity of specialized systems. Capability-based operating systems are just coming into being and HYDRA is therefore in this respect also pushing into new territory. Thus, it is a major goal of the research on HYDRA to explore the costs and benefits of a full-scale capability-based system. As will be discussed later, the protection structure of HYDRA facilitates the simple and flexible construction of operating system components as normal user programs.

A distinctly separate subgoal is to explore the role of capability-based operating systems in solving real security problems.

Two methodological goals also exist for HYDRA, both related to general computer science goals of how to produce good software. One is to provide a case example of a large system that was produced by the using the structured programming methodology. It will provide some evidence, though not formally, on how well that methodology works. The second is the goal of constructing operating systems that can be formally verified. This is an important goal of current operating system research. It is not a primary goal of this research, since the verification art has not advanced to where it can be insisted upon. But verification attempts on HYDRA will be made as effort permits.

4.3.2 Structure of HYDRA

HYDRA [Wulf 74, 75a] consists of a Kernel and a collection of Policy Modules and Subsystems. The Kernel is structured around the capability-based protection mechanism it incorporates. [Jones 73, 74] The Kernel includes those basic extensions (of the bare hardware configuration) that will permit multiple operating systems to coexist on the same shared

hardware. Thus the Kernel includes the protection mechanisms, the lowest level resource management which distributes resources to the processes executing under the aegis of different operating systems, and the primitives to permit controlled communication and synchronization of processes (and i/o devices.)

In a capability-based operating system, rights to access a resource are associated with the accessor, not the resource itself as in many second generation operating systems (e.g., the PDP10 System, DEC's TOPS TEN). Because of the HYDRA protection mechanism design, all resources can be protected in a homogeneous fashion. Users can create new 'abstract' resources; and access to these new resources can be protected using the same basic protection mechanism with no extension. The advantage of a capability based system is the fine discriminations it permits with respect to what processes can do to what sorts of things. There are two new dimensions of freedom. Accesses can be tailored to the resources involved--as opposed to the crude uniform read-only/write-only/read-write distinctions allowed in many second generation systems. Capability-based systems also permit non-hierarchial protection schemes so that there need not be a succession of concentric circles of uniformly greater access rights.

The Kernel is quite primitive and for it to be conveniently useful requires the addition of a policy module and subsystems which make up the bulk of what the user views as his operating system.

The distinction between the kernel and the policy modules and subsystems corresponds exactly to the operating system facilities that must always be there and the specializations that are permissible. Thus the key issue is what facilities reside outside the kernel, hence may be separately specialized. A Policy Module contains a scheduler and a pager. Hence these two fundamental aspects are under the complete control of the user. The policy module is also responsible for administering the resource allocation of the processes which run under its aegis.

Along with the policy module the user would probably find it convenient to have a Command Language, a Directory System, and File System. Thus most of the functions one usually associates with an operating system can be specialized for each user system if desired. The exceptions are understandable: There must be a protection system that is primitive to all subsytems -- they must be protected from each other. Specialization of protection occurs because the design of the capability-based system permit the passage of arbitrary forms of access rights to a subsystem, which in turn can then become a subprotection system and pass out further access rights. The other exception is the primitive resource allocator, since someone must own the basic physical resoures and distribute them to the subsystems.

4.3.3 Current State and Recent Results

The Kernel has been operating routinely since early 73 (originally on the 4*4 prototype hardware system that preceeded the 16*16). The first policy module, PM0, and the first set of user subsystems have been available to sympathetic users since Nov74. The facilities in these subsystems are still rather incomplete (e.g., there is no file system, so that all files are kept on the PDP10). Regular user periods have been scheduled every day since Nov74. Mean-time-to-failure is still very low (10 minutes with half a dozen relatively unsophisticated users; about 30 minutes with a single sophisticated user); but is gradually lengthening. Errors are widely distributed across hardware, software and operations.

4.3.4 Plans

The first iteration on HYDRA is complete, but there are a number of improvements and additions we need to make. Short term plans all revolve around getting an adequate user environment up and running routinely. There are several primary targets. One is extending protection to confinement (assuring that no data can leak out of a procedure, especially an unreliable one) and to revocation (taking back a right of access given to someone earlier). Another is to time heavily used functions and to recode to reduce critical overheads. This involves use of the Hardware Monitor (see below). Besides these, the adding of facilities which will be highly responsive to user demands is important. We expect the critical period, when the system appears a monster to work with, to last until about Jun75.

Longer term plans must wait until after the transferral to the small C.mmp Software Group (see below) of the responsibility for satisfying short-term user demands, which are currently all encompassing (and properly so). At that point, a very large number of alternative fundamental issues can be tackled, many corresponding to flexing degrees of freedom in HYDRA that have not yet been touched in the code that is running. Other issues to be addressed include improving HYDRA's resiliance to hardware malfunctions, incorporating resource guarantees and exploiting the 11/40 writeable microcode.

4.4 Software Facilities

There is little question about the need for software in addition to a good operating system, to make a computer system habitable. Nor is there much question about the effort it seems to take to produce it. This, along with similar statements that hold at the level of the substantial application systems, provide the ingredients for the so-called software crises. The problems are especially acute for one-of-a-kind systems, where there are not even the pleasures of amortization to average frustration down to tolerable levels. C.mmp is a one-of-a-kind system, for all momentary purposes. The problem of obtaining software facilities, however, will not go away and it must be faced. Failure to provide adequate software facilities, supporting all the functions implicit in Figure 4-4, will result in failure to develop a user community that can put substantial applications on the system.

It is time to remind the reader that each of the four links (hardware system, operating system, software facilities, application systems) is not only a co-partner in being critical, but is a co-partner in being subject matter for research. Each link has independent research goals, that come from domains of Computer Science not tied to multiprocessor research. This is true of operating sytems. This is perhaps even more true of software facilities, where the software crisis is seen as pervasive. We see several research challenges in solving the software problem, as well as it being just a critical link in the total multiprocessor effort than must get plugged.

4.4.1 Goals

Thus, the goal is to produce adequate software systems for C.mmp. With respect to type, we take this to mean essentially the list of Figure 4-4. With respect to quality, we take this to mean state of the art, which in practice means equivalent to facilities on the PDP10. In some respects demands are more stringent than on the PDP10. With HYDRA we already have an operating system that is fundamentally better than those available on the PDP10, and the software tools must be adequate to support this. For instance, the ability to create schedulers, pagers, file systems, etc., implies systems implementation facilities. Along another dimension,

the extra complexity of multiprocessors implies that debugging and program analysis techniques (e.g, performance monitors) must be better than on a uniprocessor for equivalent rates of progress. In some respects, the demands are less stringent. The collection of application programs can be much smaller, since the range of interests of the C.mmp programming community will be much narrower than a general user community.

Against the background of the known quagmire in constructing substantial software facilities, the key issue is what approaches we have to obtaining the software and at what costs. Our present strategy has six distinct strands, which we outline immediately below. The first four constitute sensible software engineering practice, but do not carry with them any research impact. The last two essentially contain a research component, so that success or failure of them is of interest from a wider view point. As noted in the final paragraph of this subsection, the amount of manpower being devoted directly to the software acquisition is quite small. Consequently, we do believe that the general success of this entire software acquisition will have some value as a case example. Certainly, failure in it (i.e., being finally caught up in the quagmire), will lead to unwillingness on our part to proceed with another such project without having developed a substantially more adequate approach.

The six strands are:

Use of higher level implementation languages: The main implementation language is BLISS11, which is a version of BLISS [Wulf 70, 71a, 72, 75a] (whose original incarnation is as a PDP10 system) for the PDP11, and which produces appropriate code for C.mmp. BLISS (and with it BLISS11) was of course a research venture, involving goals not only of programming ease for system building, but also of highly efficient code (more on this in the SMCD section). But BLISS has been working successfully for several years, and BLISS11 for over a year, and the worth of these systems is not in question in the present effort. HYDRA is implemented in BLISS11 and the code productivity there (26 debugged instructions per man/day for a 56 K instruction system) are very good with respect to the software production art. It should be noted that BLISS11 was a component of the total C.mmp effort and that the decision to produce an 11 version of BLISS was specifically made as part of the total C.mmp research strategy.

Use the PDP10s to avoid building software on C.mmp: In particular, use this strategy to keep the development of C.mmp itself off the critical path. The prime example of this is BLISS11 itself, which resides on the PDP10, compiles and loads there, with only absolute bits flowing into C.mmp. This has been critical to the development of HYDRA, of course, when C.mmp was a completely nascent system. An important part of this strand is having the software exist in both PDP10 and PDP11 forms. Thus, programs can be (often are) coded in both BLISS10 and BLISS11 (there being minor dialectical differences) and run on both machines. (See the discussion of DRAGON in the SUS section for another example.)

Reliance on the natural growth of systems: In an advanced computer science user population, especially one populated by students, systems just naturally grow. This is not a solution open generally in the development of software, but it is of great importance (and has been historically) in the development of software. That it seems managerially untidy is beside the point. Related to capitalizing on this is the first strand -- namely to have good generally available system implementation facilities. [Newell 75]

Acquisition of PDP11 systems from elsewhere and their adaptation to C.mmp: A significant advantage of having adopted a major minicomputer as the C.mmp processor, is that an immense amount of software exists and continues to be generated for it (see the discussion

of algebraic languages, below, for an example). Adaptation comes in two stages. In the first, one must simply bring the new system up under HYDRA. This is not much different (for ease and for troubles) to bringing up any program under a new operating system, with the exception that minicomputer programs tend to assume total occupancy of the computer and direct command over all system functions. The second stage is to develop multiprocessor versions of the system that can simultaneously use multiple processors. This task is generally a creative one and constitutes, from the present vantage point, an exercise in the study of multiprocessing (see the subsection later on benchmark programs). Consequently, our main concern in this strand is with the first stage, letting the second stage take place where someone's research interests evokes their desire to address this problem.

Right Structure Hypothesis: The fifth strand is built around an hypothesis that if the structure of the operating system is right, then the production of the middle level software (i.e., Figure 4-4) will be signficantly easier than under current operating systems. Operationally, the notion of "right structured" is taken to be the structure of HYDRA, and it is predicted that the time it takes to produce software facilities on C.mmp will be much shorter than extrapolations of current experience on systems under second generation operating systems would predict. To understand this hypothesis it is necessary to note that almost all the items in Figure 4-4 have significant interaction with the operating system. This interaction constitutes an important component of the complexities in such systems, though it does not necessarily account for much code. But often such software facilities must be warped in their very design in order to fit them into the constraints of the existing operating system. The type of evidence that will come forth on this will be, naturally, an analysis of cases. Thus, it will be to some extent mixed with other effects, such as the efficacy of the implementation systems, and bv the quality of the data on similar software facilities created on other systems. Still, we are looking for striking results and expect to be able to derive some conclusion about whether this hypothesis is true and why it failed if indeed the expected ease is not substantiated.

Alternative implementation system (L*): L* [Newell 71] is an interactive system in which the user constructs new systems on-line by growing them from the inside, so to speak. The facilities available within L* (as an implementation system) are adapted, augmented and modified to become the facilities of the final application system. L* is itself grown from a small kernel, the central core of which is a general symbolic list processing facility (e.g., as in Lisp), so that all of L*'s facilities are built within the system itself and are open to easy adaptation and growth. The basic L* language is interpretive, and efficiency is to be obtained by selective assembly and compilation (facilities for both being built up within L* itself). In operation, L* has the flavor of a good interactive Lisp.

L* was originally developed on the PDP10, where it has been used for implementing various AI systems (e.g., PSG, a production system language [Newell 72a], Merlin [Moore 73]). It has been undergoing an evolutionary design in order to discover the most appropriate kernel from which to grow the system. Thus, as with BLISS, the implementation language exists on both the PDP10 and C.mmp, with the consequent ability to work back and forth and debug on either machine.

L* and BLISS represent different philosophies for what is needed in an implementation system: L* is interactive, symbolic and grows its target systems; BLISS is compilation-oriented, algebraic and produces its systems as code entities distinct from itself. While BLISS is in the main stream of implementation system development, L* is much less accepted as a scientific hypothesis about how implementation of systems should be accomplished. It seems unlikely that evidence will come forth that one complex set of underlying assumptions (the BLISS set)

dominates or is dominated by another complex set (the L* set). But we expect C.mmp to be an arena in which we may find out a good deal about the space of implementation systems.

One special attraction of L* for C.mmp is the solution it offers to where all the software facilities will come from. With L* they all exist in the implementation system itself, as it is brought up initially, and they become immediately available to the user. Thus, once an L* system is running at all, all the facilities of Figure 4-4 become available (except those which represent functions outside a particular subsystem).

The implementation of the total strategy outlined above for obtaining software facilities is distributed among many groups in the C.mmp community. They all complement each other and can in general proceed along independent developmental paths. (The linkage to the PDP10s is perhaps an exception, but this part is already operational and has been a fundamental component of the C.mmp total system strategy from the beginning.) There is a small software group associated with C.mmp, whose function it is to provide the required user systems, especially at the level of adaptations of Policy Modules and Subsystems. Until now there has been no such group, but with C.mmp developing a community of users, some resources devoted to day to day smoothing of the way is necessary. We think of this small group as critical, but not as having anything like the exclusive burden for carrying out the total strategy. They will, of course, operate to coordinate it, in so far as that is necessary.

4.4.2 Current State and Recent Results

BLISS11, the primary implementation system, has been fully operational since 73. It turns out highly optimized code, namely, as stated, better code for large systems than professionals produce. (This aspect, by the way, is critical, even early on, for its use in HYDRA.)

An L* system is operational on C.mmp that has most of the facilities of Figure 4-4. The first version brought up was a stand-alone version on C.mmp in Apr74, in a month-long software experiment to demonstrate the system generation capabilities of L*. The current version, L*C.(B), works under HYDRA. It is being used for the SUS effort (see below).

4.4.3 Plans

The small software group (1 research associate + 1 programmer + 2 graduate students + part-time students) is just being organized. It is expected to be operational in Apr75. This group is independent of the main continuing work to develop HYDRA. It will be driven by immediate user needs.

Plans for the exact software facilities to be added and in what order are not yet firm, in part because the software group does not yet exist and in part because the facilities will be defined iteratively as a function of need. This is essential in a very small group which is to be highly user-responsive to a user community which is just forming. The following can be said at this time:

The SUS group, which is the major application system being put on C.mmp (see subsection below), is taken care of by L*.

There is a need to make available a regular algebraic language of the Fortran-Algol variety. (BLISS, though an algebraic language, is sufficiently oriented toward implementation issues that it does not serve the required functions; L* is not an algebraic language.) To argue

why a standard algebraic language is absolutely required is essentially to rehearse the argument about why any substantial specialized computer system must include general user facilities. For standard languages one adds the ability to import programs from elsewhere. The decision on which languages is a pragmatic one, depending strongly on the existence of versions for the PDP11 which can be brought up under HYDRA. Candidates are a CMU version of ALGOL68 [Kneuven 75], which currently exists in BLISS11, though no run time system exists yet; FORTRAN, for which interpretive and BLISS11 versions exist, PASCAL, for which multiprogramming versions for the PDP11 are being developed by Brinch Hansen at CalTech; and SAIL, for which versions for the PDP11 are being developed at Stanford.

4.5 Application Programs and Performance Analysis

This subsection stems from a special point of view: For an OK system (either in terms of architecture or operating system) application programs and systems can act as measuring tools. They reveal the properties of the system. Thus, one wants to deliberately generate a controllable set of such application programs and systems. This stands in marked contrast with the normal view about computers, which is that their uses are dictated by the desires and needs of the user community; one does not deliberately seek "applications".

We expect our user community to make use of C.mmp in a diverse set of ways, and in fact expect much of the knowledge about whether and how a multiprocessor can successfully be use to arise from this spontaneous use. However, this section concentrates on our deliberate plans for obtaining useful performance analyses of C.mmp.

There are several important dimensions of analysis. First, are the possibilities for parallel decomposition of a particular computing task, and how this maps onto the structure of a particular multiprocessor under study (here C.mmp). Second is the contribution of the various aspects of the total multiprocessing system to the computing cost. This includes memory contention, process evocation, protection, other operating system overheads and programming efficiencies of various kinds. Third is the degree of processor utilization and memory utilization. Fourth is the bandwidth requirement for communicating between processes.

Measurement tools are required in order to determine actual performance and to permit relating the performance throughout the vertical range of the system -- from hardware contention through the operating system to the program structure imposed by the decomposition scheme. The primary tool is a hardware monitor [Swan 75, Fuller 73a, b] that permits measurements at the level of the memory access, but lets them be correlated with the behavior of the software (the operating system or higher). This more sophisticated measurement is necessary since the current art of hardware monitoring usually permits only a low-level analysis with no correlation of gross level statistics with higher level behavior.

With the exception of the physical measurement tools (and of course the requisite mathematical and statistical techniques), the primary requirement of analyzing the performance of a complex system are the judicious and inspired selection of tasks to program and analyze, and the willingness to spend a substantial amount of effort in carrying the analyses through from task decomposition to programming to measurement to modeling of performance.

4.5.1 Current State and Recent Results

We have designed and built a Hardware Performance Monitor with the requisite properties. It has capabilities as advanced as any monitor in existence or being proposed (and cost us approximately a fifth of existing high performance monitors). It has been operational since Oct74 and is beginning to be used to take performance measures on C.mmp (it was used to obtain some of the C.mmp measures used for Figure 4-2).

Benchmark tasks are small well-understood programs whose behavior can shed light on aspects of multiprocessor performance. To be really useful, benchmarks must also be controllable in the sense that they can be readily adjusted to probe or stress varying parts of the computer system. We currently have a collection of four under investigation, though we are always on the lookout for additional interesting ones:

Technology chess program [Gillogly 72]: This is typical of sophisticated heuristic search procedures. There are important algorithms (e.g., the so called alpha-beta procedure) that exploit the sequentiality of search. The key question is whether parallel search systems (as is natural on multiprocessors) can survive this challenge.

FFT (Fast Fourier Transform): This is a classical and important calculation-intensive task, which appears to be ideal for SIMD machines (Single Instruction stream, Multiple Data stream systems), such as ILLIAC-4 and CDC Star. The key question is whether such vector-oriented task are amenable to effective solution on multiprocessors and how these solutions compare with those on the SIMD machines.

Pattern recognition: The algorithms in question are the classification procedures being used in current image processing work that involve essentially nearest-neighbor calculations in 4-space. The actual procedure being analyzed and programmed is one currently running on ILLIAC-4 working on satellite photographs. The key question is the same as in the FFT: to analyze multiprocessor performance on problems supposedly ideal for array processors (SIMD).

Integer programming and related optimization techniques: The key question is how a task area characterized by a set of simultaneous equations and constraints can be implemented on a multiprocessor. For instance, will it be necessary to shift to branch-and-bound search techniques to achieve the desired parallelism.

We can distinguish application systems from application programs on the basis of the complexity and multiplicity of the functions to be performed (e.g., so that no simple analysis of the "essential" inner loop of the system can suffice). Since these are total systems, it is a major decision to actually implement one and study it, a decision that has to be supported by independent research interest.

The one application system is Hearsay-2, which was described in the SUS section. This is in fact the major driver for bringing C.mmp into the world as a functioning system, since the SUS effort is operating under stringent deadlines. The natural correspondence between the structure of Hearsay-2 and the multiprocessing structure makes it an appealing case. On the other hand, the total system character of Hearsay-2 (for debugging, interaction, experimentation, evolution and multi-person development) puts substantial stress on the entire range of software facilities, and forces them all to be developed simultaneously.

4.5.2 Plans

We expect the FFT benchmark to be implemented and evaluated by Jun75. Of the other benchmarks listed, we expect to have a first iteration completed by Sep75, which will encode them in the straightforward way. Later iterations will look for good, as opposed to obvious, ways to decompose the tasks.

We expect to start a benchmark in the area of fluid dynamics simulation or in structural analysis, which deal with numerical solutions to partial diffential equations. These problems are representative of uses of computers in large government laboratories, and have always been an important class of computations (in fact they have often been the driver in the development of new computer systems).

We are considering substantial application systems in several areas. One is image understanding, for which we have an active interest at the moment as a future research direction, analagous to our work in SUS. The small benchmark task on pattern recognition has been chosen in part to shed some light on this for planning purposes.

We have also been considering the possibility of an Experimental Integer Programming Facility. Our interest stems from an interest by operations research scientists in the Graduate School of Industrial Administration.

Neither of the applications systems have advanced to the place where specific plans, much less dates of accomplishments, are presentable.

4.6 Resources

The C.mmp operating system, HYDRA, and related software development (including diagnostics) has, to date, been carried on by one staff programmer and a dedicated group of 13 graduate students under the direction of one faculty member. It is our intent to shift much of this effort to a programming group whose main responsibility is to be responsive to user demands as the system matures. To this end, a research associate has recently been added and a new full-time programmer is being requested in the new budget. There is also a modest request for part-time programming and operational personnel associated with supporting C.mmp. The performance analysis task so critical to the measurement of success of this effort is presently staffed by 4-5 graduate and undergraduate students under the supervision of one faculty member. No additional manpower is required for this aspect of the project. The engineering staff provides the equivalent of about one engineer dedicated to C.mmp development besides what is extracted from the facility costed expenses for maintenance purposes. This personnel allocation to C.mmp represents 17% of that total budget.

This project does much of its software development compilations on the PDP10 facility, but its total usage represents only about 6%. The greatest facility requirement is in the completion of C.mmp itself and the ability adequately to partition it into two useable systems for developmental and maintenance purposes. We have requested enough hardware to achieve the latter and approach the former.

The funds available in this year's budget are not sufficient to acquire the total system described in the previous section. Thus, we have put in this budget the most important $339K worth of equipment. We have listed in a supplementary section (Section I) the total remaining equipment that we now see as appropriate for C.mmp. This is done in order of desirability so

that any amount of additional funds can be applied to acquire the appropriate additional equipment.

5. SMCD: SYMBOLIC MANIPULATION OF COMPUTER DESCRIPTIONS

5.1 Introduction

The SMCD research effort is the attempt to conduct the study and construction of concrete hardware and software systems at a level that uses a symbolic description of computer systems. Techniques so developed can be applied to a range of computer systems, given only that they have been suitably described.

To understand the force of such a proposal one needs to contemplate that when humans are given a description of something (necessarily, of course, a description in symbols), they use it for whatever problems they wish to address. But the situation is quite otherwise with descriptions destined for use by computers. Here the rule is to use a description for only a single purpose. A FORTRAN program is certainly a description of an algorithm for solving a problem. But it is destined to be used by the computer in only one way: to execute the algorithm. There are many other things that might be done with the algorithm: it might be simplified, it might be verified, it might be made more efficient, it might be generalized, it might be understood, it might be used to illustrate a larger class of algorithms, it might be transformed to yield a counter example to some problem, it might be used to evaluate the interpreter of the algorithm, and so on. It is necessary to make the list rather long, since familiarity with programming languages breeds acceptance of their single purposeness. This is strongly built into the descriptions themselves, not just into the computers that interpret them: Note how few things humans will use a FORTRAN program for.

A second feature should be noted: Most descriptions for use on computers lead to computations (i.e., the computer's use of them) that are one level more instantiated than the original description. We write general algebraic expressions in our computer programs, but they instruct the computer only to make numerical calculations.

There is little caricature in the above, taken generally, though it is wide of the mark in several areas of computer science. Symbolic mathematical manipulation is done on a substantial scale; verification of programs, which implies their use symbolically, is an important field; AI in general is committed to exploring symbolic reasoning in ways reflecting the multiple manipulations carried out by humans.

We are interested in the uses of descriptions of computer systems. Many computer languages have been developed to describe computers. But here the characterization given above stands. All these languages are essentially simulators. They take the symbolic description of a computer system and perform a single calculation, which is instantiated one level more concretely: to compute out a specific behavior path of the system given a completely specified initial memory state (i.e., the exact bits).

Simulation is in fact a useful thing to do with a description of a computer, but it is not the only thing to do. One also wants to design, analyze, verify, abstract, maintain, debug, modify, explore, understand, program, summarize, document, compare, classify, evaluate and so on.

The task of working with computer descriptions is more important to computer science, both pure and applied, than even the remarks made so far indicate. Many of our activities,

both intellectual and practical, are performed in the context of specific computer systems, over and over again. To pick only the most obvious example, repeatedly we program computers, all of which bear a strong family resemblance but differ in their detailed descriptions. Given the stream of new architectures, the problem is actually a pressing one. We need generally to rise above specific computers and deal with them relative to their abstract descriptions. As human scientists we often do this informally and even formally -- on paper. But, as computer scientists know full well, the full assimilation of working at such a level of abstraction requires that we learn to do it with computers.

These remarks sketch out a research focus, one that is clearly in need of much exploration and offers many untrod paths, but not one that is totally new by any means. As mentioned, the thrust to do symbolic calculations and manipulations on computers is a pervasive aspect of computer science and AI. This research area only proposes to carry out that general program in the domain of computer descriptions. Even the specific goals we choose below are not new, for the field has been attempting to do automatic design and automatic compiler writing for a long time. This research only proposes to take a specific cut at this general area commensurate with current art, one that has better chances of success than prior tries → and no doubt less chances than the attempts that will follow at some later time.

With this general background we can now describe the particular research program we have outlined for ourselves in this area.

5.1.1 Goals

The main goal of the SMCD effort is to create symbolic descriptions of computers that can be used in a wide range of tasks. [Barbacci 74] The approach must meet three requirements. First, it must be experimental, involving actual construction of programming systems to carry out the various tasks. Second, the tasks must be ones of actual importance in the conduct of computer science. This requirement is motivated as much by the necessity to face the true sources of complexity and difficulty, as by the possibility for important application, though this latter feature is not unwelcome. Third, there must be several such tasks of diverse nature to avoid the pitfall of again producing one-use computer description schemes.

The second requirement, that of realistic tasks, implies that the total SMCD effort will contain subefforts of substantial magnitude and of a degree of independent importance that may rival the overall goal of SMCD in scientific and practical importance. It also implies that though a number of small individual efforts will exist on various tasks for using symbolic computer descriptions, these larger ones will have to be launched only with due deliberation. Thus, the number (and hence diversity) of tasks (the third requirement) can grow only slowly.

To give a feeling for the tasks we think potentially satisfy our requirements, we produce in Figure 5-1 a list of tasks adopted from the 74-75 proposal (which was the initial proposal for the SMCD research effort). Currently, we are working on just two of these tasks: the Compiler-compiler (1) and the Design of modular systems (5). These two projects provide, in effect, two independent subgoals of the total SMCD effort, with their own independent scientific merit.

A goal of the project is to create the tools, both for analysis and synthesis, to perform these tasks. It can confidently be expected that these tools, if created with sufficient clarity, will be major embodiments of what new computer science knowledge is developed by the research.

Figure 5-1: Tasks for SMCD

(1) Compiler-compiler. A system that takes as input a description of a language and a description of a computer and outputs a compiler for that computer. Given the current art, the language would probably be restricted to be Algol-like.

(2) Machine Relative System Programming. A software production facility that enables one to produce system programs such as editors, display programs, command languages, device handlers, etc. given the description of a machine. (Distinct from compiler-compiler because the type of program; distinct from automatic programming because the concern with the computer description, not with how the problem is specified.)

(3) Verification of I/O Programs. Given an I/O program, such as a device handler, and a description of both the computer and the hardware device controller, verify that the program works. (A specialization of the general verification problem, both in types of machines and types of programs.)

(4) Programming of Microcoded Special Computers. The ability to create specialized computers economically to perform particular narrow classes of algorithms (e.g., signal processing) poses an immense problem in device-dependent one-time programming of highly optimized and hence difficult machines. The task is to construct programming systems that operate relative to descriptions of such machines.

(5) Design of Modular Systems. Given a desired machine in terms of some specification language, and given a space of machines defined by a class of RT-level modules, design a machine according to various constraints and criterion functions. This is a classic design situation, and one that would be well worth making progress on, both in terms of understanding the nature of design and in terms of automating computer design.

(6) Design to Specification. Given a functional specification for a computer and a space of computer systems defined by a computer description language, design a computer that performs to the specification. This is another form of classical design task, where the starting point in a performance specification, not at higher level structure (e.g., the instruction set)

(7) Design Verification. Given a specification for a computer and a description of that computer in the language, verify that the computer satisfies the specification.

(8) Manual Generation. Given a computer defined in a language, create the documentation for the computer, i.e., an operating manual, a maintenance manual, a general introduction, etc. This task is quite different from the ones above, but also involves understanding and manipulating a computer description.

The primary tool of course is the language (or languages) for describing computer systems (they are generally called CDLs). It is a goal of the effort to design one or more CDLs. This activity cannot, however, dominate the research effort, despite its seeming "logical" priority. Many CDLs have been created (and we ourselves have created some, namely ISP, for describing instruction sets and PMS, for describing the major hardware configurational structures). Any attempt to put the design of a CDL at the top of the priority list will simply result in polishing existing languages. Instead, the language design effort must be driven by the major task efforts, i.e., by an analysis of the types of uses (other than simulation) that the CDL must support.

A second tool is that of a good simulation system. Given the historical dominance of simulation as the unique task to be performed given a symbolic description of a computer system, simulation cannot be taken as one of the major tasks to be performed (e.g, it does not appear in Figure 5-1). It is still an important task and must be performed well for any CDL that is developed. A requirement that is relatively new, is that that simulation be possible with mixed level descriptions, namely, with descriptions of computer systems that are not of a uniform degree of detail (some parts being described at a gross parametric level, some at a programming level, some at a register-transfer level and some at a crude (i.e., rise-time) circuit level).

A final tool-goal is that of providing an appropriate global data base so that many researchers can get access to the same body of data and techniques. With several somewhat independent research groups working it is important to find a way of making what one researcher does available to others, not only as a general facility way, but in terms of encodings of machines and the programs that manipulate them.

5.1.2 Current State and General Plans

The SMCD effort started only in Jul74, with the overall goals we have just described. It now has a definite shape, given by the five subsections to follow: three on core tools, Computer Description Languages, Variable Level Simulation and the Global Data Base; and two application areas, Compiler-compiler and Machine Design with Module Sets. The specific plans are given with each subsection.

We do have an overall plan for how the research should develop. The drivers of the research will be the major application tasks we take on. We have two now; we will seek to add others as interest and talent offer the opportunities. The list in Figure 5-1 is suggestive only, and we would be happy to take on others. We have a predilection, stated earlier, for moving toward realistically-sized programming and design tasks, but they must be good bets in terms of the probable light they will throw on basic computer science conceptual issues. We will iterate the design of CDLs and other basic tools, but we see this as happening in response to these applications, rather than proceeding in its own right. (Such a statement of research intent is, of course, somewhat risky, since research is in fact responsive fundamentally to good ideas.)

The SMCD research makes contact with a large fraction of central computer science, especially since some of the efforts that occur "within" it, that is, the arenas in which to attain the goals of SMCD, are within some other domain of computer science, which itself is of independent and fundamental interest. Thus research efforts supported by other funds are likely to become affiliated with SMCD, to the mutual benefit of both. This is already true of the present effort. The work on machine design with module sets is an independent focus of

an NSF grant,* and part of their interest coincides with the application area described below. Work on structured programming is the focus of another NSF grant,* for which the central interest is ALPHARD as a vehicle for exploring structured programming. However ALPHARD has become a main pillar of our effort to construct a new CDL. In all cases there is a need to retain an independent focus of each research effort. But even beyond the standard (though important) assertion that such scientific cooperation is symbiotic, the SMCD project needs such affiliations if it is to get a sufficient set of substantial diverse applications going and maintained, each of which needs to be a first rate research effort in its own right.

5.2 Computer Description Languages

The goal is to produce a language that can describe computer systems at all levels of abstraction. This include both varying levels and mixed levels within a single description. The language shall be computer manipulable, which provides the essential criteria against which success and failure can be determined.

Existing CDLs, of which there are a great many, provide an essentially adequate exploration into the basic representation of fully instantiated computer structures at the logic and Register-Transfer levels. Current work has not dealt successfully with higher levels nor with mixed levels of abstraction. And, as we mentioned, the only use for almost all of these languages has been simulation. (Indeed, many of them are embeddings of descriptive schemes in standard languagues such as FORTRAN and APL.) We ourselves tend to favor a PMS-ISP type of notation, a language system we introduced for purely descriptive (as opposed to simulation) purposes several years ago. [Bell 70, 71]

Our candidate CDL is ALPHARD. [Wulf 74] The central focus in ALPHARD is on stating abstractions of systems, to make possible execution of incompletely defined structures, verification, and extension and manipulation to more complete definitions of sytems (i.e., structured design). The central language construct is the "form", which is a schema for defining an abstract component of a system, either process or a data structure.

ALPHARD is semantically extensible, using the form. It also has a primitive control basis, which permits the description of new control structures. The definition of new control is perhaps the other greatest lack of existing CDLs (in addition to their being locked to a given level of instantiation). (In fact the two are related, since there is no sense having too much in the way of variable and mixed level descriptions if diverse control structures cannot be adequately described.)

To our current way of thinking ALPHARD provides the right ingredients to add to existing CDL langauge constructs to form the next non-trivial extension of CDLs.

---

*Research on Computer Organization and Large Modules, NSF Grant [GJ32758X], D. Siewiorek, principal investigator. The research has been going since mid 1972. About $40,000 of the grant supports work related to the Machine Design with Module Sets section of the SMCD research.

*Software and Programming Systems, NSF Grant [DCR 74-04187], M. Shaw, W. Wulf and A. Jones, principal investigators. The grant has just begun in late 1974. About $30,000 of the grant is expected to support work related to the ALPHARD aspects of the CDL section of the SMCD research.

5.2.1 Current State and Recent Results

We have a fully operational ISP compiler, with two versions (BLISS10 and BLISS11). It compiles into BLISS code of the appropriate type. This will permit experimentation pending an ALPHARD implementation.

ALPHARD specifications exist. We are using ALPHARD for manual analysis to iterate its design.

5.2.2 Plans

We expect to construct an initial compiler for ALPHARD by compiling into the common intersection of BLISS10 and BLISS11 plus POOMAS. (POOMAS is a poor man's SIMULA on the PDP10, which was developed at CMU. It adds the simulation capabilities that are not part of BLISS currently.) It will permit running on either the PDP10 or PDP11. This implementation is a stop-gap solution to obtain a running version of ALPHARD rather quickly. It will, for example, compile slowly. We will eventually construct the compiler correctly. We expect the specification to be complete by May75, with implementation to follow directly. A date for completion does not make sense until the definition is completed, but the total job should not take much more than half a man-year.

5.3 Variable Level Simulation

A very good simulator is a requirement for SMCD. The simulator must be capable of simulating at any level of abstraction possible in the CDL. It must handle a mixed-level simulation, where a system is described by an arbitrary mixture of levels. It must operate in a unifom way over all levels. Finally, it must be highly efficient. Efficiency is required not only because one wants to handle realistically large systems, but because in a mixed-level simulation it takes many cycles at the lower levels to produce consequences of interest at the higher levels.

Many simulators have been constructed, so many in fact that we characterized simulation as the single use of CDLs, and as the contrast point for the initiation of our own work on SMCD. Thus, we do not count simulation as one of the uses to be made of a CDL in enumerating our diverse applications. But it remains a critical tool, and one that we must have meeting the four criteria above.

The task of obtaining a simulator would be strictly a tool building task, except for the requirements of mixed-level simulation and possibly for the requirement of efficiency. (We would in any event have to interface a simulator to our CDLs, e.g., ALPHARD.) There has been almost no work in mixed level simulation (though we know of one effort), and this presents some important problems in how to communicate results across the interfaces between different levels. Due to their recent availabillty, efficiency today is to be obtained by using a microcoded computer. So far there have been few such simulators built. However, a plethora of such simulators over the next few years can be anticipated.

Our general approach is to translate into BLISS code from the CDL (ALPHARD). Those aspects where interactive modification is needed will be run interpretively; compiling the BLISS code will occur where appropriate; and the lowest levels of the simulation will be done on a programmable microcoded machine to obtain the necessary speed. The simulator will be able to handle descriptions done to the gate and signal level.

5.3.1 Current State

The state of ALPHARD was covered in the CDL subsection.

We are currently making a feasibility study of the two microcoded systems to which we have access. [Oakley 75] The MLP900 at ISI is used over the ARPANET is easy to generate code for this vertically microcoded machine which has a wide data-word (36 bits); it is constructed in a fast technology and is a one-of-a-kind machine. The programmable microcoded PDP11/40, which has been discussed already under C.mmp, is an in-house machine. It is relatively difficult to generate code for this horizontally microcoded machine, which has a narrow data-word (16 bits); it is constructed of a somewhat slower technology, and there are potentially many copies of the machine (both locally and at large).

5.3.2 Plans

We plan to complete the benchmarks on the MLP900 and the programmable microcoded 11/40 by Apr75. These will serve as the basis for deciding which microcoded system to use. (We will make this benchmark study generally available.) The microcoded part of the simulator is not on the critical path for the simulator and we will develop a schedule when we make the decision. The simulator design will interact heavily with the compiler design so that higher levels of abstraction can be executed directly as compiled code. More detailed levels will be simulated as a primitive sequential machine. Much will depend on the what we find out about the ease of use of both systems. In any event we do not anticipate that this is more than half of a man-year of work.

The general plans for the system is described indirectly in the discussion of the compilation of ALPHARD.

5.4 Global Data Base

A feature of the SMCD research is the gradual growth of several groups each trying to do SMCD-related work. There arises then a problem about the common use by these various groups of the tools and computer descriptions that have been generated.

It is neither possible nor desirable to force all the tools and descriptions to be formed in a common language, either a common CDL or a common programming language. The diversity of interests will imply that the groups will use different programming languages (e.g., BLISS, SAIL, LISP, L* ...), and indeed these different languages may each tie into uses being made of them in other research efforts. (Like most other advanced computer laboratories, the CMU community is multilingual with over a half dozen languages used substantially.) Even a single CDL cannot be assumed, though one is likely to be dominant to begin with (ALPHARD). CDLs are a topic of research and variant CDLs will get generated to try out new representational ideas. Though diversity is necessary, so is the need for using each others results. Encoding a large computer system should only have to be done once, though it might be necessary to translate and embellish it for new purposes.

The goal of this component of SMCD research, then, is to make the common use of results possible. It is a tool effort. The essential idea is the creation of a common global data base in which everyone's CDL data structures can reside, and can be operated on by programs coded in many different languages.

Given the incompatibility of different representations of the same thing, there is no magic common data structure that can encode information in arbitrarily different languages. More precisely, a common data structure (e.g., bit arrays) is so elementary that it assures nothing about inter-usability. The best one can do, it seems, is to provide a facility with common elementary handling and display functions, with the ability to create shared representational structure to any desirable depth, and with good facilities for mapping representations in one structure into another. To this must be added the ability to let programs in any programming system operate on the data structures of the global data base. This requirement, again, cannot be guaranteed in advance for unknown programming systems, but certainly can be for the common programming systems in the environmment.

The conditions set forth above still pose a challenging software puzzle, whose degree of solution is uncertain. Thus we envision a succession of experimental global data bases, as we try to obtain as much commonality with ease, as possible.

5.4.1 Plans

We currently are planning a global data base in L*. An experimental version should be up and running by Jul75. This effort is not on the critical path for any of the main activities of SMCD, such as ALPHARD or the substudies for the Compiler-compiler and machine designs with module sets.

5.5 Compiler-Compiler

The goal is to construct a production-level compiler-compiler that operates from a description of the target computer. The source language will be a standard algebraic language. This is the only type of language for which there has been extensive development of compiler-compilers.

To be more specific, the system shall take as input: language syntax; language axioms that describe the semantics; a machine description (in a CDL); and machine axioms that describe the behavior (which also will be expressible in the CDL). It will produce a compiler that translates source language to the machine language of the target machine. (What machine the compiler itself runs on is an independent matter, it can be on a third machine itself or on the target machine, though the latter clearly takes two cycles of the system.) The code that the compiler creates shall be production quality, i.e., of the quality of BLISS11 or FORTRAN-H and better than assembly language programmers produce for tasks of moderate size.

The problem of compiler-compilers is an old problem in the area of programming languages, active some years ago and not currently receiving much attention. Such translator writing systems (including variations which are more system generation tools than total automatic systems) are in routine use, especially for producing compilers for standard languages such as FORTRAN on new machines. The chief limitations are that they work only on conventional algebraic languages, that the compilers so produced generate low quality code, and that they are tailor-made for a given target machine. The present work thus proposes to leave basically untouched the scope of the languages accepted, but to push the other two limitations substantially. The code quality issue, in particular, seems a requirement to make the research results of genuine applied interest

5.5.1 Current Structure and Recent Results

The BLISS11 compiler provides a statisfactory model on which to base a compiler-compiler effort. As has been mentioned, it does produce highly optimized code and in creating it we have attended in some detail to the question of how to obtain high quality compilation. A monograph on BLISS11 has just been published, describing this [Wulf 75]. The general structure of the BLISS11 compiler is shown in Figure 5-2. Thus, we start the research into a machine-relative compiler-compiler with what we believe is an adequate position along one of the critical dimensions, namely how to produce high quality code. This is an extremely important point with respect to the total chances of success on the total effort.

Figure 5-2: Structure of the BLISS11 compiler

LEXSYNFLO
- Inputs source program in character string form
- Performs lexical analysis
- Processes declarations
- Analyzes syntax to produce parse tree
- Analyzes flow of control and data

DELAY
- Determines the features that can be used to simplify code
- Estimates the cost of each program segment
- Determines the order of evaluation for expressions

TLA, RANK, PACK
- Allocates memory locations, both fast registers and memory cells.

CODE
- Produces locally optimal code for each node in parse tree.

FINAL
- Analyzes actual code produced by CODE to eliminate inefficiences
- Outputs final listings and code files

As mentioned elsewhere, BLISS11 became fully operational in 74, and is receiving extensive use locally (on C.mmp, HDYRA, etc.). BLISS11 is also a DEC supplied and maintained system, so that it is available to PDP11 users generally and is receiving wide use. Thus, we have substantial confidence in this basic component of the new research effort.

With an overall structure in hand, the important problems are to discover how to do each of the parts of the task, given only a description of the computer, rather than (as in machine-specific compilers such as BLISS11) having knowledge of the target machine permeate the compiler code itself. We are currently considering three of these problems in some detail.

Analysis of the ISP structure of the machine (i.e., its instruction set) to discover the code sequences that provide the potentiality for optimization. Optimization is possible

precisely because there is more than one way to code a desired function. All instructions sets, especially richly endowed ones, provide many special ways of doing things and an important part of "understanding" a specific machine is to know these. The objective here is to produce a running computer program that carries out the discovery procedure. A thesis on this topic is complete [Newcomer 75]

The so-called register allocation problem in compiler design is how to use a limited set of fast registers, minimizing the amount of extra processing that has to be performed to shuffle data in and out of these registers. The problem for the current research effort is to do this for arbitrary types of machines, where the limitations on the scarce resources (the registers) may be of various types. A thesis on this is in mid-stream.

The FINAL pass of the BLISS11 compiler (see Figure 5-2) does what is called "peephole" optimization, looking at the final code and finding ways to improve it locally. There are a well known collection of techniques for doing this. The problem for this research effort is how to do such peephole optimization having only a description of a machine. There is a thesis on this in mid-stream.

These three subtasks illustrate the point that, having a framework for the total compiler organization, we are able to put the major part of our attention into detailed studies of the various components. Such indepth analysis is an absolute requirement for producing, not just a machine-relative compiler-compliler that turns over, but one that produces high quality code.

5.5.2 Plans

One of the three areas (Analysis of ISP structure) is finished. The other two have expected completion dates of Dec75. These three constitute an important fraction of the functions in Figure 5-2, but not all. The additional ones, e.g., syntax analysis, must be initiated. We expect to get the rest of them underway during the spring of 75. Their exact completion dates cannot be specified in advance, though none of them are more difficult than the ones we have done, e.g., about a man-year of effort once the problem is structured.

We will design and specify an initial version of the compiler-compiler after the pieces are pretty much available. We will initiate that during Fall of 75. This could produce a target date for the first version of the compiler-compiler of Aug76. The style of operation will be to start with a system with heavy manual intervention and then gradually eliminate the manual components as the task becomes successively better understood. (This design philosophy of incremental simulation is being used successfully in many complex systems, e.g., in the BBN SUS.) One effect of this methodology is to make it unclear just when a system becomes operational.

The first testbed for the compiler-compiler will probably be the re-implementation of BLISS10. There are several reasons for this choice: (1) We understand both the language and machine thoroughly; (2) the existing implementation will provide a standard against which the quality of the resulting code can be compared; (3) the current implementation of BLISS10 is fairly old, representing our first attempt at a Bliss compiler, and it would be nice to redo it in any case; and (4) the compiler-compiler will be coded in BLISS, hence it should be self-compilable. The last point implies that by altering the machine description it should be possible to have the compiler-compiler compile itself for another machine. Thus the implementation should provide both a test of the ideas as well as an exportable system available for direct use in other environments.

5.6 Machine Design with Module Sets

The basic goal of this component of the SMCD research is to design computers automatically from module sets. [Siewiorek 72, 74] This involves a combination of synthesis, analysis and housekeeping techniques applied to a developing computer description.

This is a version of a classical computer science goal. Its fundamental justification lies, on the pure side, in the understanding of computer structures that will come through analyzing their structure to the point of being able to synthesize them. On the applied side, it lies in the benefits of speed, reliability and cost that can come from successful automation.

The focus on module sets reflects the current state of the art of digital technology. What manufacturers produce today, under the impress of MSI and LSI technology, are collections of large functional units. Except in rare cases, the designer of a digital systems is faced with at most a two stage design process: first, the selection of a module set from among a small set of alternatives made available by semiconductor manufacturers; second, the assembling of these to do the desired task. Thus module sets are the natural givens for the automatic design task.

Much attention has been paid to the automation of computer design and a wide spectrum of work fits under its rubric. The design problems of interest here can be seen from enumerating what is given about the design and what is desired, what are the types of modules that can be employed, and what are the evaluation criteria.

For problems:

(1) Given the ISP description of the final system and the module set, obtain an implementation of that machine.

(2) Given functional and performance specifications for a task, obtain an ISP description of the desired machine. The module set may be a conditioning side factor influencing the ISP indirectly.

For components:

(A) Closed sets of register transfer modules, such as RTM (the DEC PDP16) or Macromodules.

(B) Open sets of register-transfer level components, such as the TI Handbook.

(C) Higher level (PMS level) components, such as CMs (Computer Modules) and computers-on-a-chip. Here the units all have programmed control of some sort.

For evaluation criteria:

(i) Cost (ii) Speed (iii) Reliability (iv) Testability (v) Chargeability

The design task becomes successively more difficult and less well understood as we go up each scale successively, e.g., a (1)-(A)-(i) problem is the easiest and a (2)-(C)-(v) problem is hardly understood at all at this stage. We are interested in this entire space of design problems, though we are working from the better understood end.

From an applied viewpoint, several aspects of the current practical world of computer design and construction are relevant to the present research. Because of rapid evolution, there is a great need to attain independence from any specific technology. This is the hardware analogue of the problem of having to program a succession of new machines. This same rapid rate of evolution increases the need to shorten the delay time between the introduction of a technology and its effective use in new computing systems. An effect that comes from the associated decrease in the cost of technology is the role of ad-hoc and one-time designs, often for exploratory purposes. These need to be done much more rapidly, if the potentiality of the improving technology is to be actualized.

In concordance with the attitudes expressed elsewhere in this project, we are interested in tackling realistic designs. Our motivations are, in part, that only by accepting real problems does one avoid missing the true difficulties, e.g., as can happen by making a bad abstraction In part, there is also a desire for our research to have applied consequences.

5.6.1 Current State and Recent Results

We have developed a system, EXPL,* in which design specifications for specific register-transfer module sets can be explored using heuristic goal-oriented techniques. [Barbacci 73] EXPL engages in modifications of the control flow of the system in order to obtain optimizations (actually, design is done in terms of trade-offs between cost and performance). EXPL is fully operational (late 73). It is tied to a specific module set, RTMs.

A study has been completed of variability in data structures (at the register-transfer level). This is independent of any particular module basis. It has not been implemented in a system.

5.6.2 Plans

We are investigating the extension of EXPL to Macromodules, the other main register-transfer level closed module set. This will force us into a major act of abstraction, and is important in attaining generality in this approach. We can expect to have such an extension by May75. That is, it should be capable of producing designs for the same class of problems as EXPL now produces with RTMs.

A test for well structured control flow (deadlock free, live) that serves to check candidate descriptions is in the advanced debugging stage and should be available by May75. A simulator, a microcode generator (for an RTM microcode controller), and an RTM wire-list generator will be completed by Aug75. A testability measure, which correlates well with actual test generation effort, has been developed and should be running by Aug75.

By Dec75 we intend to have a system that incorporates all the above programs. Given a description in ISP a design, in terms of a wiring list (RTMs or Macromodules) or microcode (RTM), will be developed according to designer-given trade-offs in cost, speed, and testability.

---

*EXPL and, in general, the work in this area prior to the initiation of SMCD, have been carried out under NSF Grant GJ32758X, mentioned earlier. The division of labor between the NSF Grant and SMCD is that the NSF grant is centrally focussed on understanding computer design and the structure of module sets; SMCD is centrally focussed on understanding the ways computer descriptions can be manipulated. Each instance of an application of SMCD, such as this Machine Design subproject, must be driven mostly by its own problems, not by the indirect goal of obtaining another diverse use of symbolic computer descriptions.

A simulation will be possible and the well structuredness of an algorithm will also be guaranteed.

We are beginning the design of a second iteration of the total register-transfer level system for computer aided design. This system must show the ability to: (1) work with variable module sets; (2) optimize over control flow; and (3) to optimize over data flow. We do not have well grounded expectations yet for how large a system it should be able to design (measured, say, in terms of number of modules in the final design). EXPL produces designs of the order of 50 control steps and a dozen data modules, which is enough to reach student exercises in register transfer design (e.g., in [Bell 73]. 1973). An order of magnitude greater than that exceeds the domain in which special purpose register-transfer module systems are practical in today's market. We expect to have an initial specification document by Apr75.

A leading candidate for the structure of this system is the Independent Co-operating Knowledge Sources structure being developed for Hearsay-2. The same conditions seem to prevail, namely that there is a large number of types of knowledge each of which must be represented and handled on its own terms. One desperately wants to be able to build such a design system and then to add to it new knowledge sourcess whose detailed structure was not anticipated in advance, and to revise radically existing knowledge sources -- all the while keeping a system which always does the best with what its got. The data structures appropriate to computer representation and to speech (in the Hearsay-2 global data base) are sufficiently different that it seems doubtful that one can capitalize on the actual program structure. But the design itself may follow very similar lines.

5.7 Resources

This research area draws upon the contributions of four to five faculty members at various levels of effort. There is one research associate and 13 graduate students assigned to it. This group totals only about 12% of all budgeted personnel expenses. The largest subarea is the computer description language devleopment, ALPHARD, which involves three of the faculty, the research associate and two graduate students. No additional manpower is required on this effort. The compiler-compiler subproject is supported solely by graduate students. Should there be a need to produce earlier results here, it would require additional manpower in the form of a research associate and/or staff programmer. Similarly, the variable level simulation subgoal is presently primarily a graduate student effort. Machine design with module sets is again supported by a larger, more diverse group including some effort by two faculty and several of the hardware oriented graduate students.

The individual efforts applied under this proposal to this area interact but do not overlap with two NSF grants in the department. Machine design with module sets relates to efforts under Research on Computer Organization and Large Modules and the ALPHARD group interact with the efforts of an NSF grant for Software and Programming Systems.

To date, SMCD research has been adequately served by the PDP10 facility, absorbing about 11% of it. Its needs should continue to be adequately covered except for some possible requirements for special facilities: large memory for global data base work, network use of the MLP900, and perhaps additional or alternate microprocessors.

6. FACILITIES

6.1 Introduction

The computational facility provides a range of computational resources to the Computer Science research community. In general, resources are available for whatever research directions are being pursued.

Figure 6-1 summarizes the current configuration of the facility. It consists of two large PDP10 KA10 systems, connected to the external world by a communications processor (C.communication) and by the IMP to the ARPANET. C.mmp is listed to make the picture complete, although it does not yet provide generalized computing. Essentially, it is still in a developmental state though it has user time scheduled.

6.2 Current State

The two PDP10s systems are mature, running 99% of a scheduled 22 hour day, 7 day week, for a user community of 250. We have just begun to get response measures (as opposed to load measures, which have always been available) from our system. Table 6-2 gives a few statistics. The average response times of 2.2 seconds during the peak period correspond to a fairly high fraction of annoyingly long delays even when doing trivial things (waiting for editor response, sending a mail message, etc.), say above 10-15 seconds.

The C.communication is just becoming operational (Jan75) and it still handles only a part of its intended function.

In terms of style of operation, the systems run totally in interactive mode. We use the printers pretty much for immediate scratch work, all permanent material is produced on the XGP printer (as was this proposal).

6.3 Plans

The existing PDP10 systems are essentially complete, except for additional disk space. We intend to upgrade our 6 RP02s to RP03s, which essentially doubles our amount of space per disk and adds a total of 30 megawords.

C.communication requires modest upgrading. An IMP interface would relieve the CMU hosts of Telnet protocol overhead. This would actually be significant since there is heavy net traffic between the 10A and 10B systems. Also disk storage is needed for transcription/rollback of terminal i/o scrolled off small-window video terminals. This facility, we believe, would permit more use of video terminals in offices (where they are essentially pure soft-copy devices), with a consequent increase in productivity. No funds are included for this in the current budget, due to the priority of completing C.mmp.

The 10B system contains 5 of the high quality graphics terminals, which are used primarily for speech and vision research, and are available only incidentally for others. We need at least one additional high quality graphics terminal for the 10A, to satisfy a strong

Figure 6-1: Facility configuration

- PDP10 Systems
  - PDP10A
    - KA10 Pc
    - 240K Mp
    - 800K Swapping Drum
    - 60 megawords DSK
    - 5 Dectapes
    - 2 Magtapes
    - IMP host connection
  - PDP10B
    - KA10 Pc
    - 256K Mp
    - 800K Swapping Drum
    - 70 megawords DSK
    - 5 Dectapes
    - 2 Magtapes
    - IMP host connection
    - Real-time speech devices (ADC, DAC, ZCC, AMS) running at 20kHz
    - Link to 11.XGP
  - User system interface
    - Embellished TOPS-10 operating system
      - Full Telnet and FTP protocol implementation
      - CMU graphics support
      - Real-time devices and scheduling support

- C.mmp current state (Mar75)
  - Processors
    - 5 PDP11/20s functioning
    - Designing PDP11/40 relocation box, cache and modifications for 5 PDP11/40s
  - Primary Memory
    - 700K available on 12 ports
      - 512k on 11 ports functioning Dec74
  - Switch
    - 16x16 operating routinely
  - Secondary Memory
    - Paging disk: 4 of 7 IMS disks ($2*10^6$ words) operational
    - Disks for secondary storage: 2 RP03s (40 megawords)
  - Communication
    - 2 4800/300 baud links to PDP10A
    - Front-end connection via 4800 baud asynchronous lines
    - IMP connection under development

Figure 6-2: Facility Performance and Capacity

	Jobs	% Idle	% Overhead & lost time	Terminal Output Response (millisec)*
10A (open system)				
Midday peak period	32.0	1.2	22.5	2192
Prime time (1000-2000)	26.5	29.2	8.6	864
Non-prime (2000-800)	14.8	71.2	4.1	520
10B (administratively controlled to guarantee availabilty to SUS)				
Midday peak period	24.8	5.7	19.2	1449
Prime time (1000-2000)	25.1	36.0	12.0	1087
Non-prime (2000-800)	15.1	2.0	5.2	478

*Average time users wait for their jobs to do terminal ouput after either a command is issued to run a job, or terminal input is given that removes the job from terminal input wait state.

---

demand for such a terminal from the non-speech part of the user community. Again, no funds have been included for this.

As far as regular terminals are concerned, we continue to acquire a small number each year, providing a gradual transition from a purely teletype environment of a few years ago. The most important terminal at this time is the portable hardcopy terminal, which can be used at home in the evening. Distributing these to heavy users produces a noticeable shift in their computing to the late night and early morning hours, with a consequent relief of computing during the peak daytime and evening periods. Thus, these terminals actually generate increased computational efficiency of the facility. We include a small number of such hardcopy terminals along with the addition of a few alphanumeric terminals.

6.4 Resources

Operation and maintenance of the facility accounts for 18% of the total personnel budget. This includes hardware maintenance of all equipment from processors to terminals, and operations on a 24 hours per day, 365 days per year basis. The composition is the equivalent of 3.6 members of the programming staff, 5.3 from the engineering group and 2 in operations, with corresponding part-time budgets in each area.

Approximately 12% of the PDP10 facility is used in maintaining the total facility. This includes the cost of system modification, generation, accounting, batch and peripheral processing programs, and ARPA network server functions.

7. COMPLETION OF C.MMP HARDWARE

The major equipment is concentrated almost exclusively on trying to complete C.mmp. The items in the budget all meet the needs discussed in the proposal proper (C.mmp Resources). The size of the acquisition shown below for C.mmp is dictated by the imposed limit on the size of the total budget and not by the nature of C.mmp itself. We present on the next page a supplementary budget that gives the remaining items we would like to obtain for C.mmp.

38	8k Memory modules 300 k more, provides 1M words	1,300	49,400
4	PDP11/40's bring C.mmp level up to 13 processors 4 11/20's, 9 11/40's	11,696	46,784
8	11/40 C.mmp mods reloc, local clock, proc. mods, cache (4 for above 11/40's, 4 for other 11/40's in house, one will be done on 74-75 budget)	5,000	40,000
2	writeable control stores simulation shows up to a factor of two performance increase over standard 11/40 (obviously depending upon specialized function utilization)	4,800	9,600
1	full duplex DR11B 11 to 11 DMA intf. for AI-11 to C.mmp	5,292	5,292
2	full duplex DR11B 11 to 11 DMA intf. for front-end to C.mmp	5,292	10,584
1	DEC RP04 disk controller	6,397	6,397
1	DEC RP04 disk drives (100 million bytes ea.)	23,310	23,310
5	Video terminals	3,150	15,750
	Total		207,117

The supplement below will yield a completed configuration for C.mmp. The conversion to all PDP11/40s will maximize the capacity and cost effectiveness of the architecture. We also need to attain a swapping ratio of 4-5 to primary memory. Further expansion of the disk system and a small increment in allocated video terminals will serve to make C.mmp a more independently operational system by reducing its reliance on PDP10 facilities.

The itemized list below describes and budgets these items in order of descending importance to achieve a complete system.

4	PDP11/40s Upgrade 11/20's enables writeable control stores on all processors permitting free distribution of operating system functions among them. Allows cache implementation on all processors. Reduces maintenance costs by increasing replication. Upgrades system to 14 11/40's.	11,696	46,784
4	11/40 C.mmp mods Reloc, local clock, processor, mods, cache for above.	5,000	20,000
4	Writeable control stores For above processors. Simulation shows up to a factor of two performance increase over standard 11/40 (obviously depending upon specialized function utilization).	4,800	19,200
2	IMS swapping disks 500k word disk and controller	12,000	24,000
1	DEC RP04 disk drives (100 million bytes each)	23,310	23,310
3	High performance video terminals	3,150	9,450
	Total		142,744

8. RESEARCH INTO MULTIPLE COMPUTER SYSTEMS

8.1 Introduction

We would like to obtain funding for continued explorations into computing with multiple computer structures.

We need to provide one additional item of background information to add to the view of our research as seen in the main body of this proposal. Then we will lay out the total proposal directly, relying on the general background to guarantee that our aims are scientific and that the quality of our science is to be respected. After that we do provide a list of the major types of scientific questions to be asked in multiple computer research. Finally we present a budget.

8.2 CM Machine

We are currently involved in one attempt to move ahead with the investigation of multiple computer structures. [Fuller 73]. We are in the early stages of the design and construction of a 100 processor system with processors of the size of a PDP11/20 (0.18 MIPS). This is part of the research effort to understand modules for computer structures, initiated in 1972 by Gordon Bell, Dan Siewiorek and Sam Fuller and supported by a small NSF grant. The name given to the structures they have investigated are CMs, for Computer Modules. The efforts under NSF grant GJ-32758X are about equally divided between design automation and modular computer structures. The current effort is being carried out jointly with DEC, who is supplying the physical structure, using one of their current LSI implementations of the PDP11 (which realizes the computer in five chips).

The structure, which is now undergoing detailed design, essentially consists of two types of components. First, the computer module: an LSI 11 with 4K to 32K of memory; and second, a component which acts as the inter-CM switch, mapping memory requests across the structure. The switch component is being designed around the Intel3000 microcomputer chips set. The structure permits an arbitrary number of computers to be interconnected in a way that lets them share an address space, hence be a multiprocessor organization. There will be a cost associated with more distant communication, so that issues of programming locality will be of the essence. This is in contrast to the classical multiprocessor architecture in which the random access character of the address space has to be carefully preserved at the hardware level. We attach a summary of an initial design exercise as an appendix to this section. Although the current design is slightly different from those in the appendix, we only intend to create a flavor for CMs sufficient for the remaining discussion.

We expect to come up with a 10 processor version of the system in one year and, if that goes well, to move to a structure with about 100 processors in another 18 months after that.

The sorts of tasks that we will be investigating on the CM machine are similar to the benchmarks and application systems discussed for C.mmp. We do not yet have a major application as a driver for the structure, in part because the design has remained highly variable as we have iterated with different technological specifications.

8.3 The Proposal

We can now express succinctly the essential points of this supplemental proposal. The main proposal has discussed the basic research issues about multiprocessors in the context of C.mmp. These issues apply equally to multiple computers generally. The main proposal has also presented evidence about our way of approaching such research problems.

(1) Multiple small mass-produced LSI computers potentially provide an order of magnitude improvement in cost-effectiveness over conventional uniprocessor computer systems.

(2) We know almost nothing about how to realize the gains of such systems. The technology, which keeps moving ahead, adds to what we know nothing about by creating new architectural possibilities. What we want to know is how to create useful total user systems which realize the cost-performance gains attainable with LSI memory and microprocessor components.

(3) What is needed is to obtain substantial amounts of experience with such structures.

(4) To be worthwhile, that experience must attend to all aspects of the problem -- the architecture, the operating system, the software facilties, the task decomposition and analysis, the total system reliability, and the measurement and analysis of performance. By "attend" we mean "attend scientifically", which implies taking each aspect as a research concern, and producing first rate computer science by studying it. We can do no better than to assert C.mmp as our model of what such total attention amounts to. It implies, we assert, the sort of federated research effort seen there.

(5) To be worthwhile, each proposed computer system must be the focus of an attempt to do at least one substantial task -- one that represents independent accomplishment, either scientific or applied. This provides the guarantee of relevance. With C.mmp the SUS application plays this role. And in general at CMU other significant computer science tasks are being formulated and solved within the same environment in which the new machines are being being conceived. Currently an image understanding system (IUS) is being considered as a potential driving application.

(6) There cannot be simply one multiple computer system built. C.mmp tells us something about the classic multiprocessing structure; it is only one point in a design space. As successive variations and improvements of technology and architecture show up, they pose quite different limiting aspects and offer quite different possbilities for solution. A single research organization, such as CMU, must move deliberately in advancing from system to system, so as to use each system as an occasion for scientific progress, and so as to analyze the end-product use of each machine. Withal, there must still be a cycle time for permitting the next system structure to take shape. This period is determined in part by the rate at which aspects of the technology change.

(7) The current technology implies that substantial experimental systems can be developed for modest costs in an environment which has the facilities and expertise. This is a radical change over even a few years ago. A reasonable way to estimate this is probably by direct hardware costs, from which other costs can be reasonably guessed. We expect the CM system now being designed to have a total hardware cost (for the 100 processor system) of about $360K. If funded appropriately it might have a total cost (hardware, software, support, and core research) over its development lifetime of $900K. (These cost estimates are rough, though more accurate ones could be developed.)

(7.1) If experimental systems prove to be interesting and cost effective then it might be appropriate to initiate an effort to build much larger systems (e.g. a 1000 processor multiple computer). Such an effort, naturally, would cost substantially more than indicated here.

(8) We are not fully supported for the investigation of the CM machine. Indeed, given our goals for the breadth at which such research should be done, we are not even adequately supported. For instance, we have no resources of our own to purchase the hardware; thus we cannot make appropriate choices at times of component selection. There are no full time personnel associated with the project (such as Research Associates, programmers, engineers).

(9) Thus, we propose that we be provided funds sufficient to carry out the present experimental system at an appropriate level and to prepare for the next investigation. The essential ingredients seem to us to be:

(9.1) For each system that we wish to attempt, we will develop a detailed technical proposal that deals with all of the aspects of the problem: architecture, operating systems, software facilities, task decomposition, reliability and performance measurement and analysis.

(9.2) The presumption is that a suitable proposal can be developed for each system. Thus, we can proceed with the timing of the design and development without taking the long lead times that are required for the approval of a design effort when approaching a funding agency as an initial contract or grant. However, it will be possible to deflect a particular project if the proposal is unsuitable.

(9.3) There will be driving tasks associated with each system, to be part of the proposal, as described above. The tasks for such designs can be selected and negotiated to meet a set of mutual goals. (We are currently beginning a year study for the Defense Communication Agency to explore the application of multiple processor systems to digital Communication networks.)

(9.4) We plan to actively work on the Computer Module system for at least three years. Subsequent multiple computer studies may well take us beyond three years, but a minimum duration of three years is needed in order to assure we can push the CM machine well beyond the hardware development stage and into a phase where we can address the problems and opportunites associated with the end use of the multiple computer system.

(10) We are proposing here to seriously explore the multiple computer design space; to only propose to study Computer Module systems (as we now understand them) would limit the project too much in scope to be assured we are addressing the major issues and alternatives. Below we list several topics that need to be explored and may well form the core of experimental systems beyond the Computer Module system proposed here. Items will certainly be added to this list as we gain more experience with C.mmp and Computer Modules.

(10.1) In both C.mmp and now Computer Modules we have limited the architecture of the central processor to the PDP11 instruction set. There have been good reasons for this: primarily, there exists good software

development on the PDP11. However, on a number of counts the 11 architecture is too constrained. For example, the 11 really needs a much larger address space than its current 2^{15} words. Larger integer data-types and a coherent set of interprocessor communication instructions are needed. One direction of future efforts may be to consider the design of the proper central processor architecture for 10 to 100 processor systems.

(10.2) In our discussion with a number of other laboratories specifically, the Univ. of Hawaii and Intel Corp., the prospect of very high bandwidth buses has been discussed. In the current system we are proposing the use of conventional technology to achieve 1 to 2 Megawords per second on the interCM bus. Advances in fiber optics and linear integrated circuits suggest it may be economical to consider systems built around one or two global, high performance buses.

(10.3) As we have mentioned in the main part part of this proposal, the cost of primary memory can be a dominant factor in the cost of the system. A number of interesting technologies are emerging that hold out the promise of a cost-effective high performance secondary store. CCD's, Bubble memories, and electron beam memories are currently attracting the most attention. Only CCD's, however, have been incorporated into a commerical product to date (Intel's CCD 'drum' based on their 16K bits/chip packages. The potential to exploit CCD's and Bubble memories seems particularly attractive on a multiple computer system. The many processors can be used as I/O processors and hence maintain a very high average bandwidth between these solid state secondary stores and primary memory.

8.4 Scientific Questions

We have a strong interest at CMU in exploring the technical problems associated with multiple processor systems. Our work on the C.mmp and Computer Modules projects has helped to clarify some of the major unsolved problems associated with these systems . Several open problems are listed below:

(1) Capacity of Inter-Processor links. Links between computers (or processors) in any network can be characterized by three primary performance parameters: bandwidth, i.e., bits/sec.; latency, i.e., time to execute the transfer of a file, packet, or interrupt request between computers; and the ratio of information bits to control bits. There is currently a dearth of information on the bandwidth and latency required in multiple processor networks. By benchmarking and measuring several applications on both C.mmp and CMs, interprocessor communication levels can be studied. Based on these studies new and improved analytical models can be developed.

(2) Deadlocks. Current multicomputer systems are susceptible to deadlocks. In other words, it is possible for two processors to request links or memories in such a sequence that they lockup for an indefinite length of time. Measurements of the ARPAnet indicate this in fact occurs. Other multiple processor structures such as the BBN Pluribus and DEC PDP11 systems using Unibus Windows also acknowledge the possibility of deadlock. As the next generation of multiple processor systems are used in applications requiring a closer degree of cooperation among processors, the deadlock problem will become a much more serious issue. During the

course of CM design several deadlock prevention techniques were examined. With the scheme adopted it should be possible to measure deadlock potentials and actual deadlocks that were prevented.

(3) Inter-Processor Control Mechanisms. There is presently no concensus on how to effectively pass control among processes in a multiple processor system. C.mmp has a simple inter-processor interrupt unit; the BBN Pluribus has the PID (pseudo-interrupt device); and Computer Modules generalize the memory mapping mechanism to include control mappings as well. Control mechanisms in uniprocessor systems include subroutines, coroutines, goto's (or exits in goto-less programs), and interrupts. An analogous set of control mechanisms is needed for multiple processor systems. Since the initial ten CM testbed will have read/write control stores, various interprocessor control mechanisms can be implemented, measured, and evaluated. [Siewiorek 75]

(4) Process to Processor Binding. As the number of processors in a computing system increases, our current notions and intuition about multiprogramming and processor utilization must be re-examined. Possibly the concept of multiprogramming should be discarded in some multiple processor systems. The simplification in the software could be significant and the time saved by not having to save and switch processor states would result in higher performance in some applications. But what are the guidelines for matching processes to processors and what degree of multiprogramming is appropriate in a particular instance?

(5) Problem Decomposition. In order to take advantage of the potential high reliability, incremental expandability, and very high throughput of a multiple processor system, a problem must be decomposed into parallel, cooperating processes. To date such decomposition has only been achieved for special purpose tasks by individuals intimately familiar with that task. A hill climbing exercise based on developing many benchmarks should give us insight into this and the previous problem.

(6) Addressing Problem. A problem with all multiple processor systems that use 16-bit mini/microcomputers for their processors is that their immediate name space is limited to 64K. One important reason for the development of a read/write control store for the PDP11/40 at CMU is to allow us to expand the architecture of the PDP11 to get larger address types, in addition to the standard 16-bit addresses, or to add instructions to manage efficiently the relocation box. In addition, CMs will have 28 bit network addresses external to the processors.

The problems of reliability, security, and modularity will take an increasing importance as our work progresses.

8.5 Budget

The budget for the major equipment is given on the pages immediately following. Major equipment is divided into a small initial system to test alternative concepts and then the full 100 processor system to test our design on an operational many processor system. Also included in the budget is the expected cost of experimental hardware for high speed bus and memory structures which would permit exploring alternate architectures.

MAJOR EQUIPMENT

Initial 10 Processor CM System

10	LSI 11 with 12K words memory	$1,500	$15,000
15	LSI 11 to Inter-CM bus interfaces	500	7,500
3	Intel 3000 Inter-CM bus controllers with writeable control stores	4,000	12,000
1	PDP11/40 with 28k memory, 30 cps terminal, DECtape drive, and link to PDP10A	27,000	27,000
-	Power supplies, cabinets, and cables	8,000	8,000
	Total		$69,500*

*We are in the final stages of negotiating a joint effort with Digital Equipment Corporation on the initial 10 processor system and DEC will provide CMU with the needed hardware components. The $69,500 for the 10 processor system is not included in the hardware costs of this proposal; the 10 processor CM system is listed here primarily to show what we already intend to build.

Additional Hardware to Make a 100 Processor CM System

90	LSI 11 with 12K words of memory	$1,300	$117,000
120	LSI 11 to Inter-CM bus interfaces	200	24,000
10	Intel 3000 Inter-CM bus controllers	2,000	20,000
-	Power supplies, cabinets and cables		40,000
1	link to PDP10A	5,000	5,000
1	link to C.mmp	5,000	5,000
5	High Speed Drums (mechanical or solid state [e.g. CCD's])	8,000	40,000
1	3330 equivalent disk storage (10^8 word capacity)	40,000	40,000
-	Maintenance and spare parts		18,000
-	Test Equipment		6,000
-	Site Preparation		2,000
	Total		$317,000

9. REFERENCES

Baker, James K. (1974), "The DRAGON System - An Overview", Proc. IEEE Symposium on Speech Recognition, Pittsburgh, Pa., 22-26.

Baker, James K. (1975), Stochastic Modeling as a Means of Automatic Speech Recognition, PhD. Dissertation, Department of Computer Science, Carnegie-Mellon University.

Baker, Janet (1975), A New Time-Domain Analysis of Human Speech and Other Complex Waveforms, PhD. Dissertation, Department of Computer Science, Carnegie-Mellon University.

Barbacci, M. R. (1973), The Automatic Exploration of the Design Space for Register Transfer (RT) Systems, PhD. Dissertation, Computer Science Department, Carnegie-Mellon University.

Barbacci, M. R. and D. P. Siewiorek (1974), "Some Aspects of the Symbolic Manipulation of Computer Descriptions", Technical report, Computer Science Department, Carnegie-Mellon University.

Bell, C. G. and A. Newell (1970), "The PMS and ISP Descriptive Systems for Computer Structures", AFIPS Conference Proceedings, Spring Joint Computer Conference, 351-374.

Bell, C. G. and A. Newell (1971), Computer Structures: Readings and Examples, McGraw-Hill.

Bell, C. G., J. Grason and A. Newell (1973), Designing Computer and Digital Systems Using PDP16 Register Transfer Modules, Digital Press, Maynard, Mass.

Berliner, H. (1973), "Some Necessary Conditions for a Master Chess Program", Proc. of the Third International Joint Conference on Artificial Intelligence, Stanford, California.

Berliner, H. (1975a), Chess as Problem Solving: The Development of a Tactics Analyzer, PhD. Dissertation, Department of Computer Science, Carnegie-Mellon University.

Berliner, H. (1975b), "A Representation and some mechanisms for a Problem Solving Chess Program", in Recent Advances in Computer Chess, Edinburgh University Press, Edinburgh, Scotland.

Bobrow, Daniel B. (1968), "Natural Language Input for a Computer Problem-Solving System", in Marvin Minsky (ed.) Semantic Information Processing, The MIT Press, 146-226.

Chase, W. G. and H. A. Simon (1973), "The Mind's Eye in Chess", in W. G. Chase (ed.) Visual Information Processing, Academic Press, 215-281.

Erman, L., R. D. Fennell, V. R. Lesser and D. R. Reddy (1973), "System Organizations for Speech Understanding: Implications of Network and Multiprocessor Computer Architectures for AI2, Proc. Third International Joint Conference on Artificial Intelligence, Stanford, California,194-199.

Erman, L. D. (1974), An Environment and System for Understanding of Connected Speech, PhD. Dissertation, Stanford University, Stanford, California.

Erman, L. D. and V. R. Lesser (1975), "A Multi-Level Organization for Problem Solving Using Many Diverse Cooperating Sources of Knowledge", Technical Report, Department of Computer Science.

Farley, A. M. (1974), SIPS: A Visual Imager and Preception System: the Result of a Protocol Analysis, PhD. Dissertation, Computer Science Department, Carnegie-Mellon University.

Fennell, R. D. (1975a), Multiprocess Software Architecture for AI Problem Solving, PhD. Dissertation, Department of Computer Science, Carnegie-Mellon University.

Fennell, R. D. and V. R. Lesser (1975), "Parallelism in AI Problem Solving", Technical Report, Department of Computer Science, Carnegie-Mellon University.

Fuller, S., D. Siewiorek and R. J. Swan (1973), "Computer Modules: An Architecture for Large Digital Modules", ACM/IEEE First Annual Symposium on Computer Architecture, Gainesville, Florida, 231-239.

Fuller, S. H., H. J. Swan and W. A. Wulf (1973a), "The Instrumentation of C.mmp: A Multi-Mini-Processor", Proceedings COMPCON 73, New York, 173-176.

Fuller, S. H. and K. Stevenson (1973b), "Performance Monitor for C.mmp", Eleventh Annual Allerton Conference, Urbana, Illinois.

Gerritson, R. (1975), Understanding Data Structures, PhD. Dissertation, Department of Computer Science, Carnegie-Mellon Unversity.

Gillogly, J. J. (1972), "The Technology Chess Program", Artificial Intelligence 3, 145-163.

Gilmartin, K. J., A. Newell and H. A. Simon (1975), "A Program Modeling Short-Term Memory Under Strategy Control", C. N. Cofer (ed.) The Structure of Human Memory, W. H. Freeman, San Francisco (in press). Psychology Department, Carnegie-Mellon University.

Goldberg, H. G., D. R. Reddy and R. L. Suslick (1974), "Parameter-Independent Machine Segmentation and Labeling", Proc. IEEE Symposium on Speech Recognition, Pittsburgh, Pa., 106-111.

Hedrick, C. L. (1974), A Computer Program to Learn Production Systems Using a Semantic Net, PhD. Dissertation, Department of Computer Science, Carnegie-Mellon University.

Knueven, Paul (1975), "The Foundation of a Flexible Run-Time System for Algol 68S", Proc. International Conference on Experience with Algol 68, University of Liverpool Press, Liverpool, UK.

Jones, A. K. (1973), Protection in Programmed Systems, PhD. Dissertation, Department of Computer Science, Carnegie-Mellon University.

Jones, A. K. and W. A. Wulf (1974), "Towards the Design of Secure Systems", Proc. of the International Workshop on Protection in Operating Systems, IRIA, Rocquencourt, France, 121-135.

Lesser, V. R., R. D. Fennell, L. D. Erman and D. R. Reddy (1974a), "Organization of the HEARSAY II Speech Understanding System", Proc. IEEE Symposium on Speech Recognition, Pittsburgh, Pa. 11-21.

Lesser, V. R. (1974b), Parallelism in Speech Understanding Systems: A Survey of Design Problems", R. Reddy (ed.) Speech Recognition: Invited papers of the IEEE Symposium, Pittsburgh, Pa.

Lesser, V. R. (1975), "HSII: A Multiprocessor Speech Understanding System", Interface Workshop on Interprocess Communication, pp.

Moore, J. and A. Newell (1974), "How can Merlin Understand?", in L. Gregg (ed.) Knowledge and Cognition, Lawrence Erlbaum Associates, Potomac, Md., 201-252.

Moran, Thomas P. (1973), The Symbolic Imagery Hypothesis: A Production System Model, PhD Dissertation, Department of Computer Science, Carnegie-Mellon University.

Newcomer, J. M. (1975), Machine-Independent Generation of Optimal Local Code, PhD. Dissertation, Department of Computer Science, Carnegie-Mellon University.

Newell, A., D. McCracken, G. Robertson and L. DeBenedetti, (1971), L*(F) (documentation), Department of Computer Science, Carnegie-Mellon University.

Newell, A. and H. A. Simon (1972a), Human Problem Solving, Prentice-Hall.

Newell, A. (1972b), "A Theoretical Exploration of Mechanisms for Coding the Stimulus", in A. W. Melton and E. Martin (eds.) Coding Processes in Human Memory, Winston and Sons, Washington, D.C., 373-434.

Newell, A. (1973a), "Production Systems: Models of Control Structures", in W. C. Chase (ed.) Visual Information Processing, Academic Press, 463-526.

Newell, A., J. Barnett, J. W. Forgie, C. Green, D. Klatt, J.C.R. Licklider, J. Munson, D. R. Reddy and W. A. Woods (1973b), Speech Understanding Systems: Final Report of a Study Group, (published for Artificial Intelligence), North-Holland/American Elsevier. (Original publication, 1971)

Newell, A. and G. Robertson (1975), "Some Issues in Programming Multi-Mini-Processor", 1974 Conference on the On-Line Use of Computers in Psychology, Journal of Behavior Research Methods and Instrumentation, Psychonomic Society, Inc., Austin, Texas.

Oakley, J. (1975), "A Comparison of Two Microprogrammable Processors: MLP-900 and PDP11/40E", Technical Report, Computer Science Department, Carnegie-Mellon University.

Ohlander, R. (1975), Analysis of Natural Scenes, PhD. Dissertation, Department of Computer Science, Carnegie-Mellon University.

Reddy, D. R. (1973a), "Eyes and Ears for Computers", NTG/GI Fachtagung "Cognitive Verfahran und Systeme", Springer-Verlag, Berlin/N.Y., 1-28.

Reddy, D. R., L. D. Erman and R. B. Neely (1973b), "A Model and a System for Machine Recognition of Speech", IEEE trans. on Audio and Electroacoustics AU-21 (3), 227-238.

Reddy, D. R. and L. Erman (1974), "Systems Organizations for Speech Recognition", R. Reddy (ed.) Speech Recognition: Invited papers of the IEEE Symposium, Academic Press, N. Y. (in press)

Shockey, L. and L. D. Erman (1974), "Sub-Lexical Levels in the HEARSAY II Speech Understanding System", Proc. IEEE Symposium on Speech Recognition, Carnegie-Mellon University, Pittsburgh, Pa., 208-210.

Siewiorek, D. P. and M. B. Barbacci, "Automated Exploration of the Design Space for Register Transfer (RT) Systems", Proc. First Annual Symposium on Computer Architecture, Gainesville, Florida.

Siewiorek, D. P. and M. R. Barbacci (1974), "Some Observations of Modular Design Technology and the Use of Microprogramming", Technical Report, Department of Computer Science, Carngie-Mellon University.

Siewiorek, D. P. (1975), "Process Coordination in Multi-Microprocessor Systems", Proc. of Workshop on the Microarchitecture of Computer Systems, Nice, France.

Simon, H. A. (1971), "The Theory of Problem Solving", Proceedings of the IFIP Congress, Ljubljana, Yugoslavia, 249-266.

Simon, H. A. and J. B. Kadane (1974), "Optimal Problem-Solving Search: All-or-None Solutions", in Artificial Intelligence, (in press).

Swan, R. J. and S. H. Fuller (1975), "K.mon: The C.mmp Hardware Monitor. A Programmer's Manual", Department of Computer Science, Carnegie-Mellon University.

Waterman, D. A. (1974), "Adaptive Production Systems", CIP Working Paper, No. 285, Psychology Department, Carnegie-Mellon University.

Waterman, D. A. (1975), "Serial Pattern Acquisition: A Production System Approach, CIP Working Paper, No. 286, Psychology Department, Carnegie-Mellon University.

Wulf, W. (1970), BLISS: A Systems Programming Language, University of Pittsburgh Press.

Wulf, W., A. N. Habermann and D. Russell (1971a), "BLISS: A Language for Systems Programming", Communications of the ACM, 780-790.

Wulf, W. and C. G. Bell (1971b), "C.mmp: A Multi-Mini-Processor", AFIPS Conference Proceedings of the Fall Joint Computer Computer Conference, Anaheim, Caifornia, 765-778.

Wulf, W. A., J. L. Apperson, C. M. Geschke, R. K. Johnsson, C. B. Weinstock, D. S. Wile, R. F. Brender, P. A. Knueven and M. R. Pellegrini (1972), "Bliss 11 Programmers Manual", Department of Computer Science, Carnegie-Mellon University.

Wulf, W. A.,, E. Cohen, W. Corwin, A. Jones, R. Levin, C. Pierson and F. Pollack (1973), "HYDRA: The Kernel of a Multiprocessor Operating System", Communications of the ACM.

Wulf, W. A. (1974), "ALHARD: Toward a Language to Support Structured Programs", Technical Report, Department of Computer Science, Carnegie-Mellon University.

Wulf, W. A., R. K. Johnsson, C. B. Weinstock, S. O. Hobbs and C. M. Geschke (1975a), The Design of an Optimizing Compiler, American Elsevier, N.Y.

Wulf, W. A. and R. Levin (eds.)(1975b), "The HYDRA Operating System", Technical Report, Department of Computer Science, Carnegie-Mellon University.